Diffusion models

Laura Sánchez García Julio Antonio Soto Vicente

IE University (C4_466671 - Advanced Artificial Intelligence)

Fall 2024

Outline

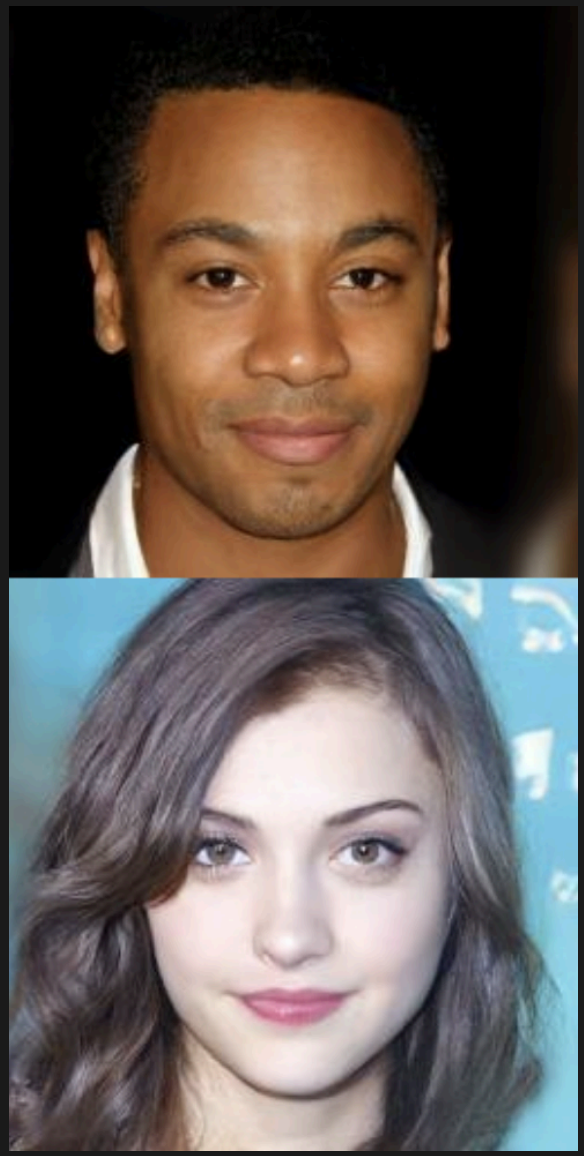
Intro

Generative models

A (certainly not complete) list:

- Latent Variable models (incl. VAEs)
- Autoregressive models (incl. GPT-style Language Models)
- GANs
- Flow-based models (incl. Normalizing Flows)
- Energy-Based Models (incl. Score-based models)
- **Diffusion models** (kind of mix of all previous points)
- Combinations

Image generation



Source: Ho el al. [2020]

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

Source: Saharia et al. [2022]

Video generation

Source: Brooks et al. [2024]

Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models

Outline

- The forward process
- The Nice^{TM} property
- The reverse process
- Loss function
- Training algorithm
- The model
- Sampling algorithm

Denoising Diffusion Probabilistic Models (DDPMs)

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

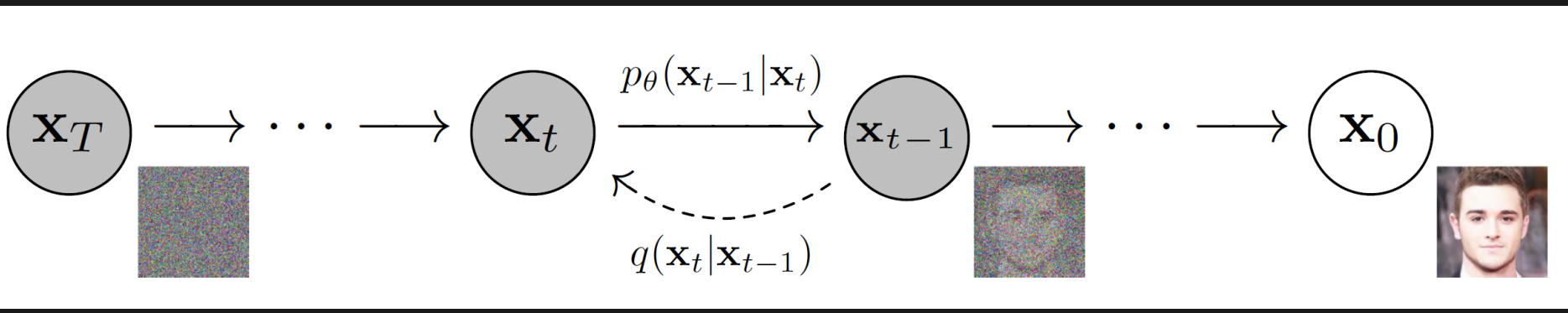
Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

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.

Denoising Diffusion Probabilistic Models (DDPMs)

DDPMs work through many steps t which are $0, 1, \ldots, T$

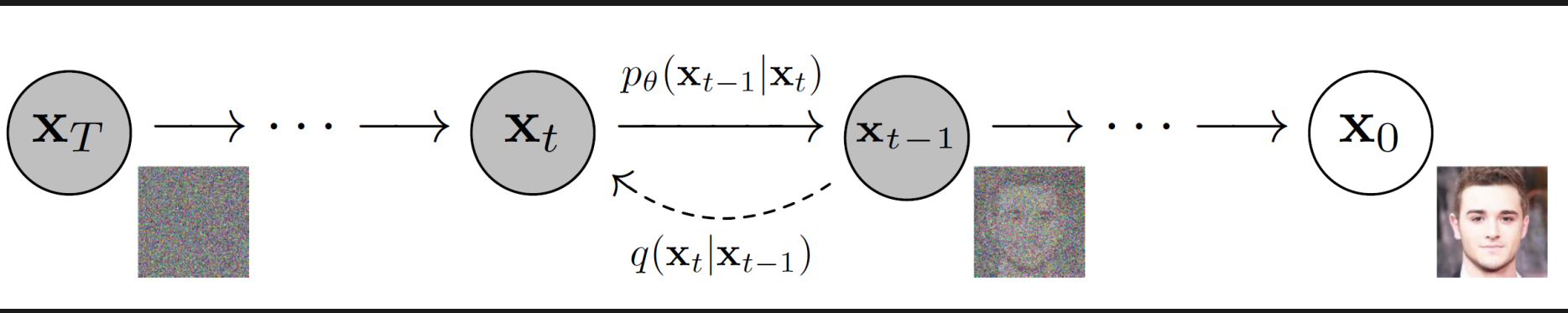


Source: Ho et al. [2020]

- $\mathbf{x}_0$ is the original image
- $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is the **forward** diffusion process
- $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will be the **reverse** diffusion process (learned by our model with weights θ)

Denoising Diffusion Probabilistic Models (DDPMs)

DDPMs work through many steps t which are $0, 1, \ldots, T$



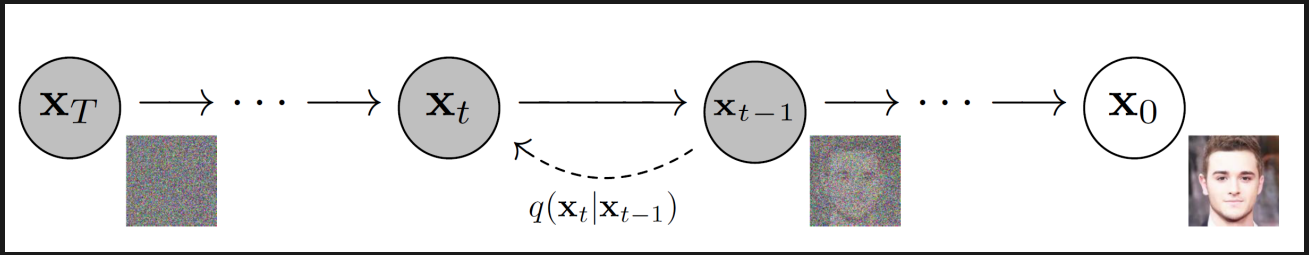
Source: Ho et al. [2020]

- $\mathbf{x}_0$ is the original image
- $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is the **forward** diffusion process
- $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will be the **reverse** diffusion process (learned by our model with weights θ)

During forward diffusion we add Gaussian (Normal) noise to the image in every t, producing noisy images $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_T$

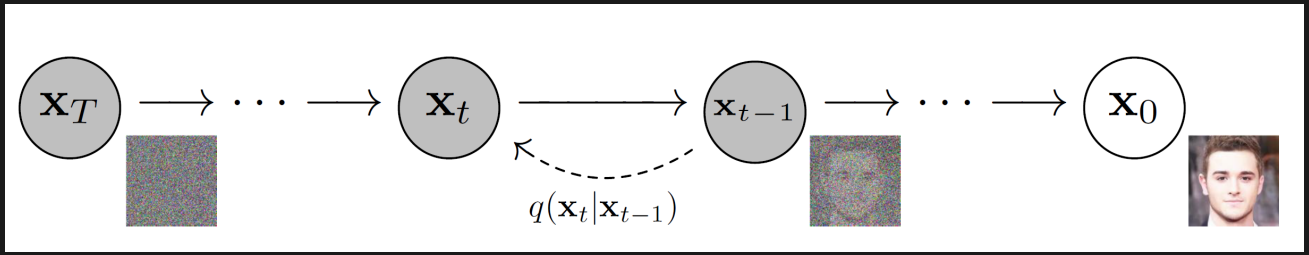
As t becomes higher, the image becomes more and more noisy

The forward process



$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \mathcal{N}(0, \beta_t \mathbf{I})$$

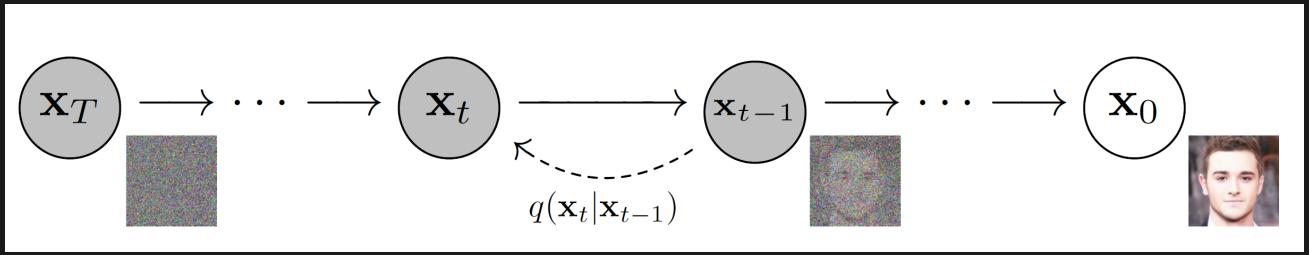
The forward process



$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \mathcal{N}(0, \beta_t \mathbf{I})$$

- Take an image at some point $t-1$ as $\mathbf{x}_{t-1}$
- Generate Gaussian noise from an isotropic multivariate Normal of size $\mathbf{x}_t$, with mean 0 and **variance** β_t
- Scale $\mathbf{x}_{t-1}$ values by $\sqrt{1-\beta_t}$ (so data scale does not grow as we add noise)
- Add the noise the the scaled image

The forward process

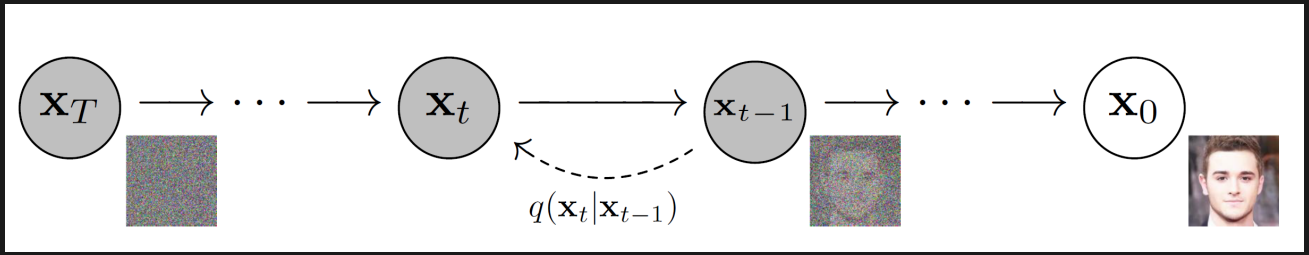


$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \mathcal{N}(0, \beta_t \mathbf{I})$$

- Take an image at some point $t-1$ as $\mathbf{x}_{t-1}$
- Generate Gaussian noise from an isotropic multivariate Normal of size $\mathbf{x}_t$, with mean 0 and **variance** β_t
- Scale $\mathbf{x}_{t-1}$ values by $\sqrt{1-\beta_t}$ (so data scale does not grow as we add noise)
- Add the noise the the scaled image

It can be directly computed as $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$

The forward process



$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \mathcal{N}(0, \beta_t \mathbf{I})$$

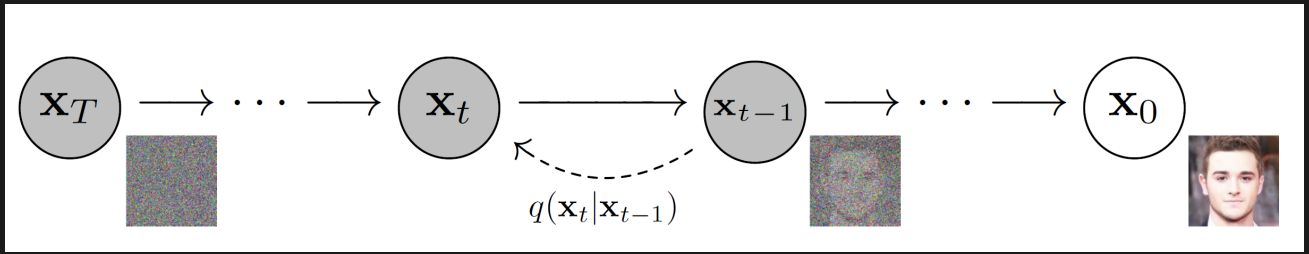
- Take an image at some point $t-1$ as $\mathbf{x}_{t-1}$
- Generate Gaussian noise from an isotropic multivariate Normal of size $\mathbf{x}_t$, with mean 0 and **variance** β_t
- Scale $\mathbf{x}_{t-1}$ values by $\sqrt{1-\beta_t}$ (so data scale does not grow as we add noise)
- Add the noise the the scaled image

It can be directly computed as $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$

β_t is called the *variance schedule* $\beta_1, \ldots, \beta_T$ that effectively controls how much noise is added in each step t [1]

[1] In the paper it is made to grow linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ for $T = 1000$

The forward process



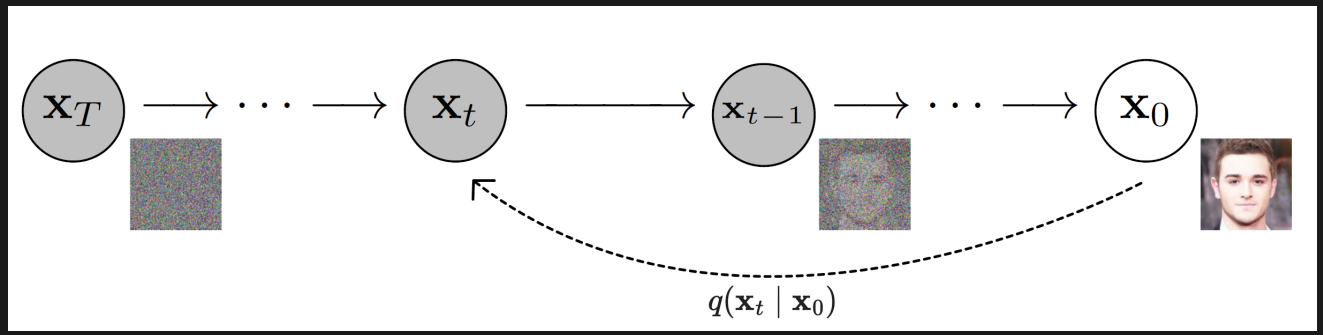
The full forward process is therefore:

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

For a large T, the final image is basically only noise (all original image info is essentially lost), so it becomes roughly $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$

Demo here!

The Nice™ property

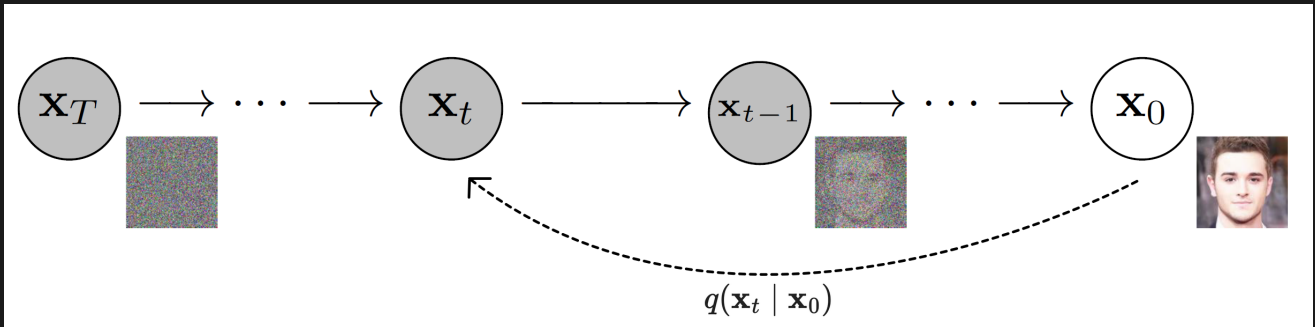


Trick to get any $\mathbf{x}_t$ from $\mathbf{x}_0$ without having to compute the intermediate steps. Let's define $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$. We can use the reparametrization trick for the Normal distribution to get:

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

In the paper this is described as *"A notable property"*. I believe that the first to call this as a nice property was Weng [2021]. We will call it the Nice™ property

The Nice™ property



Trick to get any $\mathbf{x}_t$ from $\mathbf{x}_0$ without having to compute the intermediate steps. Let's define $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$. We can use the reparametrization trick for the Normal distribution to get:

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

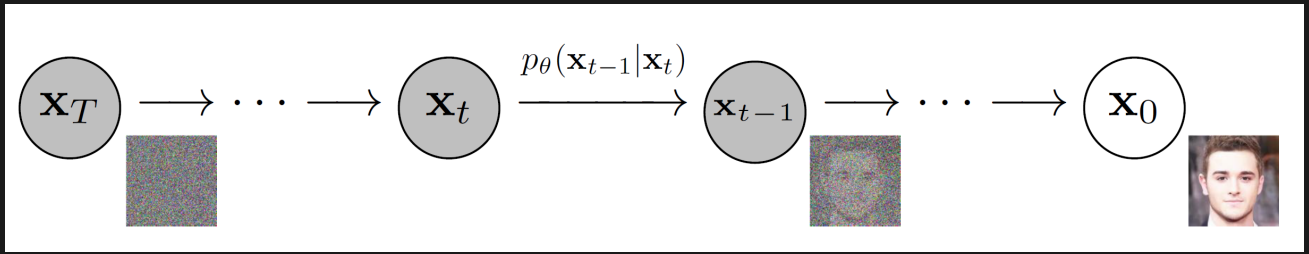
- Easier, faster computation
- Any image state $\mathbf{x}_t$ comes from a (Normal) probability distribution, drastically simplyfing derivations

Details in Appendix A!

Demo here!

In the paper this is described as *"A notable property"*. I believe that the first to call this as a nice property was Weng [2021]. We will call it the Nice™ property

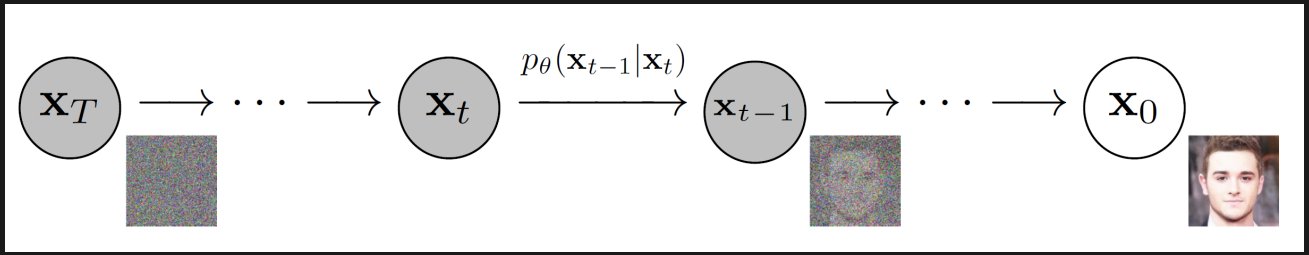
The reverse process



We will train a model p_θ to learn to perform the reverse process

Starting from $p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$, it will try to recreate the image!

The reverse process



We will train a model p_θ to learn to perform the reverse process

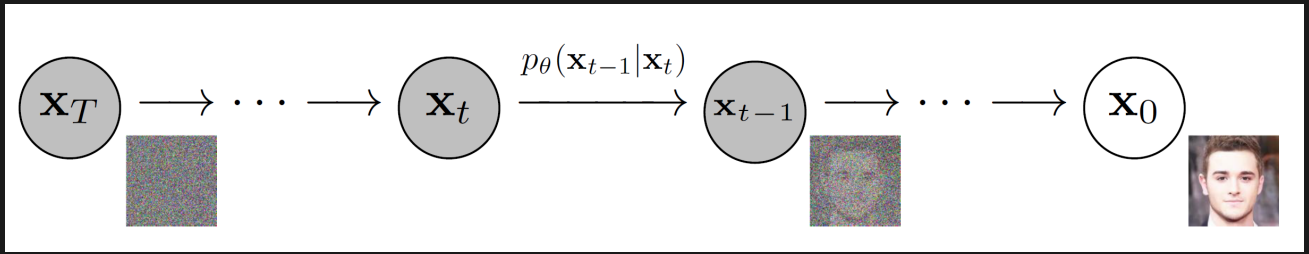
Starting from $p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$, it will try to recreate the image!

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1};\ \mu_\theta(\mathbf{x}_t, t)\ ,\ \Sigma_\theta(\mathbf{x}_t, t)\ )$$

Will be a neural network prediction

Will be set to a value $\sigma_t^2 \mathbf{I}$ based on β_t

The reverse process



We will train a model p_θ to learn to perform the reverse process

Starting from $p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$, it will try to recreate the image!

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1};\ \mu_\theta(\mathbf{x}_t, t)\ ,\ \Sigma_\theta(\mathbf{x}_t, t)\ )$$

Will be a neural network prediction

Will be set to a value $\sigma_t^2 \mathbf{I}$ based on β_t

And

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

Summary

Reverse process

Initial distribution

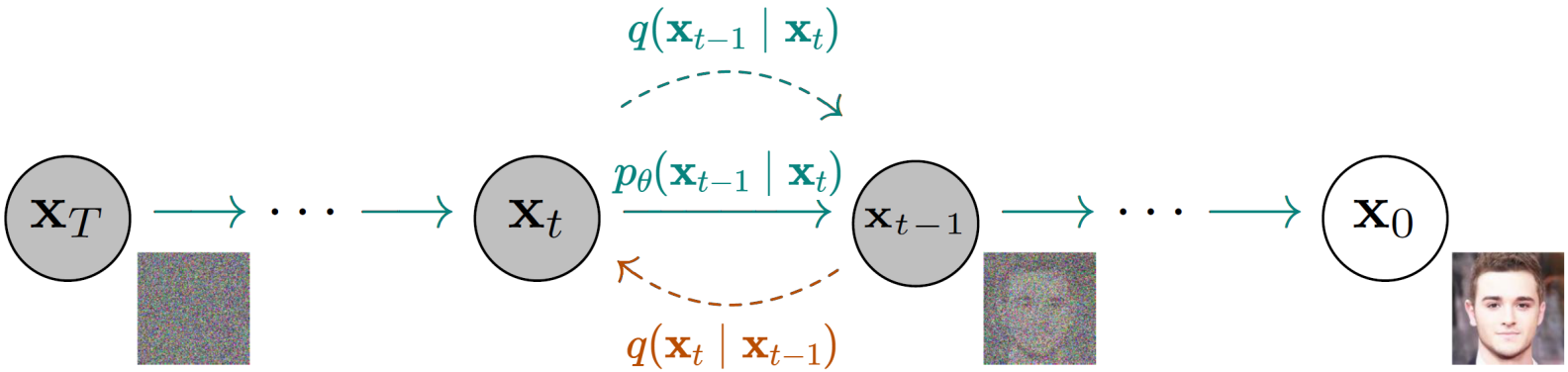
$$p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$$

Learned reverse transformation

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

Approximation of *forward process posterior*

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$



Forward transformation

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

Initial distribution

$$q(\mathbf{x}_0)$$

Forward process

Summary

Reverse process

Initial distribution

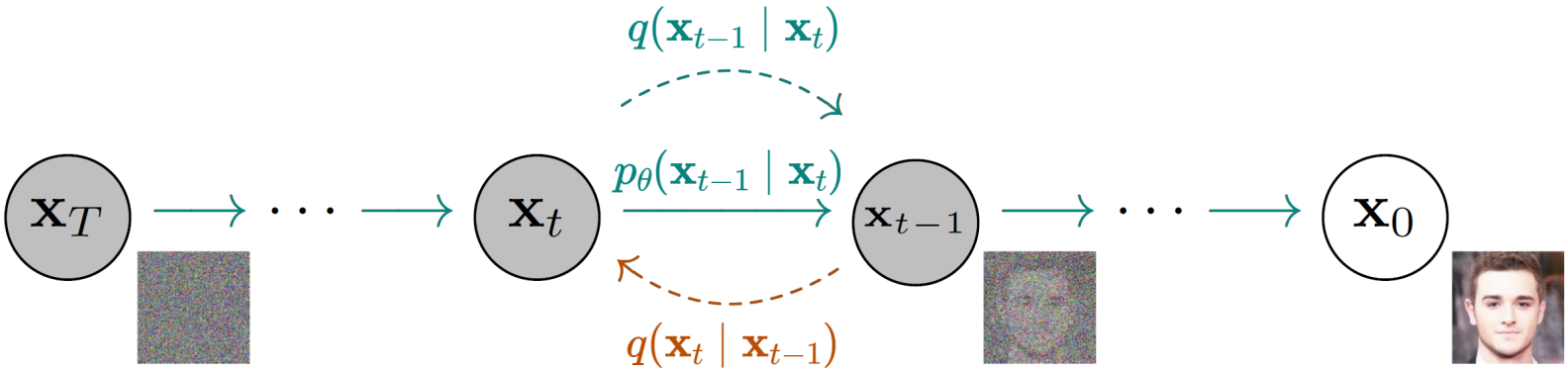
$$p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$$

Learned reverse transformation

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

Approximation of *forward process posterior*

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$



Forward transformation

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

Initial distribution

$$q(\mathbf{x}_0)$$

Forward process

The *forward process posterior* is the ground-truth reverse diffusion process that the model will learn to approximate!

Loss function

Just like in VAEs, the loss function is based on the Evidence Lower Bound (ELBO):

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

Loss function

Just like in VAEs, the loss function is based on the Evidence Lower Bound (ELBO):

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

Which becomes:

$$\mathbb{E}_q\left[\underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}_{L_0}\right]$$

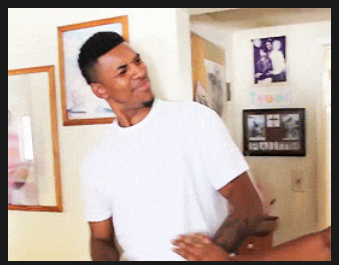
Loss function

Just like in VAEs, the loss function is based on the Evidence Lower Bound (ELBO):

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

Which becomes:

$$\mathbb{E}_q\left[\underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}_{L_0}\right]$$



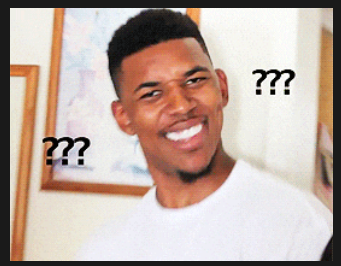
Loss function

Just like in VAEs, the loss function is based on the Evidence Lower Bound (ELBO):

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

Which becomes:

$$\mathbb{E}_q\left[\underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}_{L_0}\right]$$



Details in Appendix B!

Loss function

Just like in VAEs, the loss function is based on the Evidence Lower Bound (ELBO):

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

Which becomes:

$$\mathbb{E}_q\left[\underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}_{L_0}\right]$$

- $L_T \rightarrow$ prior matching term. Has no learnable parameters, so we ignore
- $L_{t-1} \rightarrow$ denoising term
- $L_0 \rightarrow$ reconstruction term. Only learning how to go from $\mathbf{x}_1$ to $\mathbf{x}_0$, so authors ended up ignoring it (simpler and better results)

Details in Appendix B!

Loss function

Loss therefore focuses on L_{t-1}:

$$\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))$$

Where:

- $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ is the *forward process posterior* (i.e. what would be the *prefect*, ground-truth reverse process) conditioned on $\mathbf{x}_0$
- $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will be our learned reverse process as in slide 13

Loss function

Loss therefore focuses on L_{t-1}:

$$\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))$$

Where:

- $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ is the *forward process posterior* (i.e. what would be the *prefect*, ground-truth reverse process) conditioned on $\mathbf{x}_0$
- $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will be our learned reverse process as in slide 13

The forward process posterior is tractable and can be computed as:

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

Where

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$

Loss function

Loss therefore focuses on L_{t-1}:

$$\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))$$

Where:

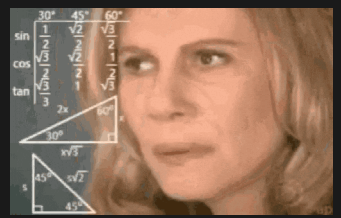
- $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ is the *forward process posterior* (i.e. what would be the *prefect*, ground-truth reverse process) conditioned on $\mathbf{x}_0$
- $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will be our learned reverse process as in slide 13

The forward process posterior is tractable and can be computed as:

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

Where

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$



Details in Appendix C!

Loss function

Loss therefore focuses on L_{t-1}:

$$\mathcal{D}_{\mathsf{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))$$

Where:

- $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ is the *forward process posterior* (i.e. what would be the *prefect*, ground-truth reverse process) conditioned on $\mathbf{x}_0$
- $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will be our learned reverse process as in slide 13

The forward process posterior is tractable and can be computed as:

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

Where

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$

Details in Appendix C!

Loss function

Loss is therefore the KL divergence between two Normals: the forward process posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and the reverse process that our model will learn $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Loss function

Loss is therefore the KL divergence between two Normals: the forward process posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and the reverse process that our model will learn $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Since both are Normal distributions, this KL divergence is:

$$\mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Forward process posterior mean (slide 17)

From model's prediction (slide 14)

Loss function

Loss is therefore the KL divergence between two Normals: the forward process posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and the reverse process that our model will learn $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Since both are Normal distributions, this KL divergence is:

$$\mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Forward process posterior mean (slide 17)

From model's prediction (slide 14)

However: authors decide instead to **predict the noise** added during the forward process. Reformulating:

$$L_{\text{simple}} := \mathbb{E}_q \left[\left\| \epsilon - \epsilon_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Added noise in forward pass

Model predicting the noise using $\mathbf{x}_t$ and t as features

Loss function

Loss is therefore the KL divergence between two Normals: the forward process posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and the reverse process that our model will learn $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Since both are Normal distributions, this KL divergence is:

$$\mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Forward process posterior mean (slide 17)

From model's prediction (slide 14)

However: authors decide instead to **predict the noise** added during the forward process. Reformulating:

$$L_{\text{simple}} := \mathbb{E}_q \left[\left\| \epsilon - \epsilon_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Added noise in forward pass

Model predicting the noise using $\mathbf{x}_t$ and t as features

Details in Appendix D!

Loss function

Loss is therefore the KL divergence between two Normals: the forward process posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and the reverse process that our model will learn $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Since both are Normal distributions, this KL divergence is:

$$\mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Forward process posterior mean (slide 17)

From model's prediction (slide 14)

However: authors decide instead to **predict the noise** added during the forward process. Reformulating:

$$L_{\text{simple}} := \mathbb{E}_q \left[\left\| \epsilon - \epsilon_\theta(\mathbf{x}_t, t) \right\|_2^2 \right]$$

Added noise in forward pass

Model predicting the noise using $\mathbf{x}_t$ and t as features

Details in Appendix D!

Training algorithm

Algorithm 1 Training

1: **repeat**
2: $\quad \mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $\quad t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: $\quad$ 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

Source: Ho et al. [2020]

Training algorithm

Algorithm 1 Training

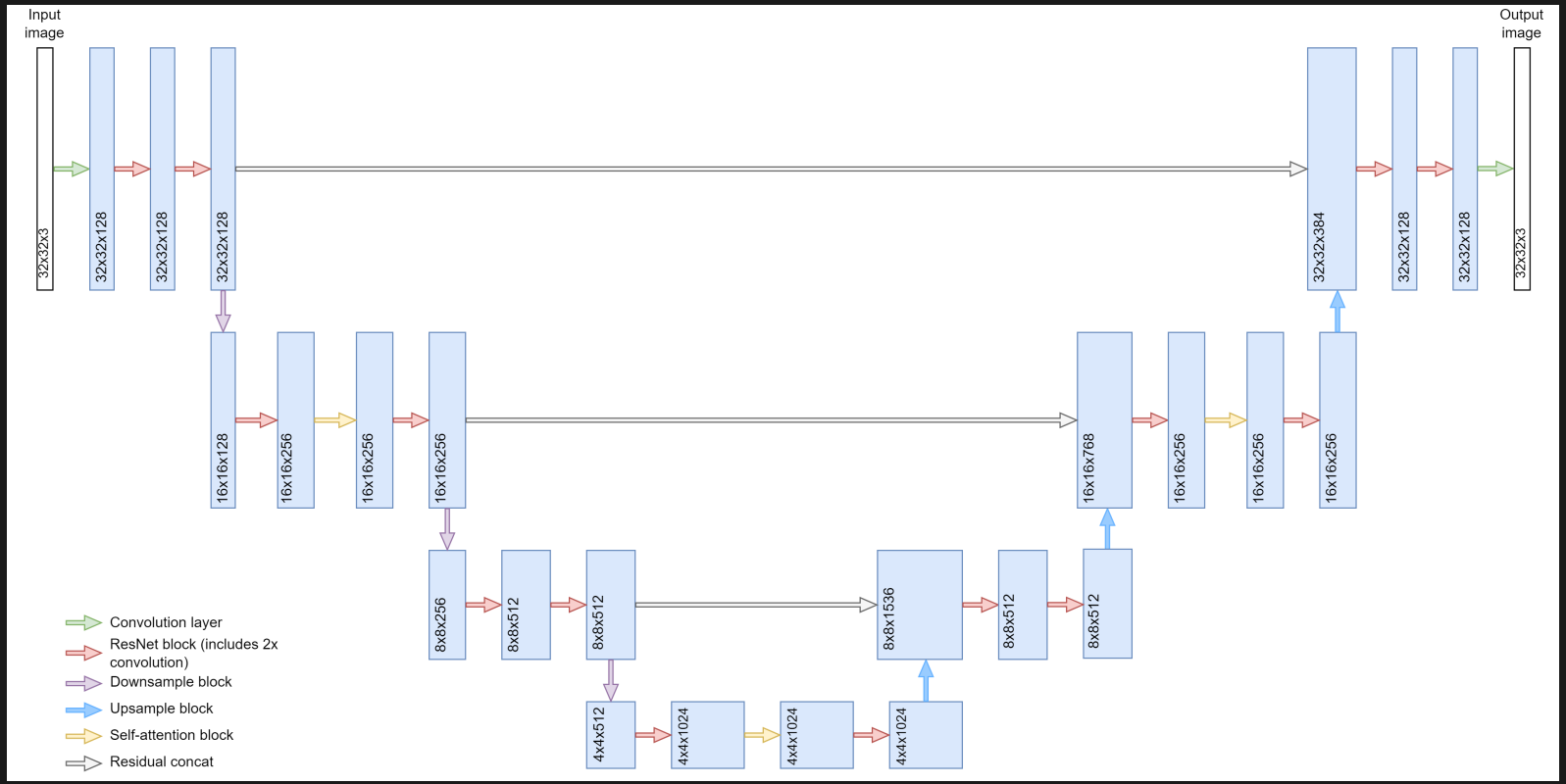
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Source: Ho et al. [2020]

Where $\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1+\bar{\alpha}_t}\epsilon$ is just $\mathbf{x}_t$ computed through the Nice™ property!

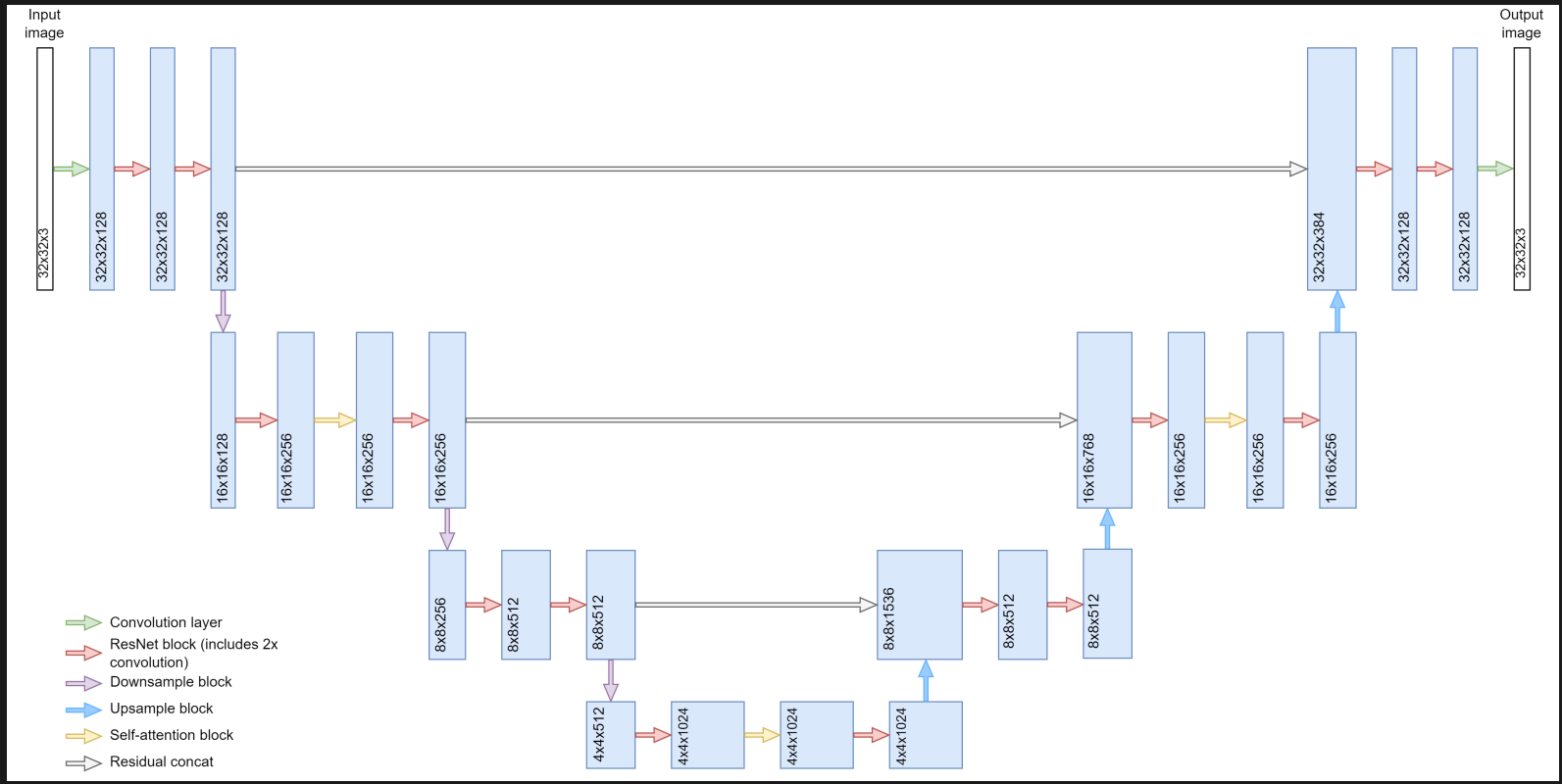
The model

Proposed model is a U-Net architecture (Ronneberger et al. [2015]) that includes self-attention blocks



The model

Proposed model is a U-Net architecture (Ronneberger et al. [2015]) that includes self-attention blocks



They also include GroupNorm (Wu and He [2018]) in ResNet and self-attention blocks

t is added on every ResNet block through positional encoding (Vaswani et al. [2017])

Sampling algorithm

Once the model is trained, we can generate new images by:

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Source: Ho et al. [2020]

Sampling algorithm

Once the model is trained, we can generate new images by:

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Source: Ho et al. [2020]

Step 4 just applies the reparametrization trick to the learned reverse process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$

Sampling algorithm

Sampling is an iterative process: we progressively remove predicted noise

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Sampling algorithm

Sampling is an iterative process: we progressively remove predicted noise

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

You may wonder:

If at any single step we are predicting the full added noise ϵ, why don't we remove it in a single step?

Sampling algorithm

Sampling is an iterative process: we progressively remove predicted noise

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

You may wonder:

If at any single step we are predicting the full added noise ϵ, why don't we remove it in a single step?

Answer:

Details in Appendix E!

Advancements and improvements

Advancements and improvements

Outline

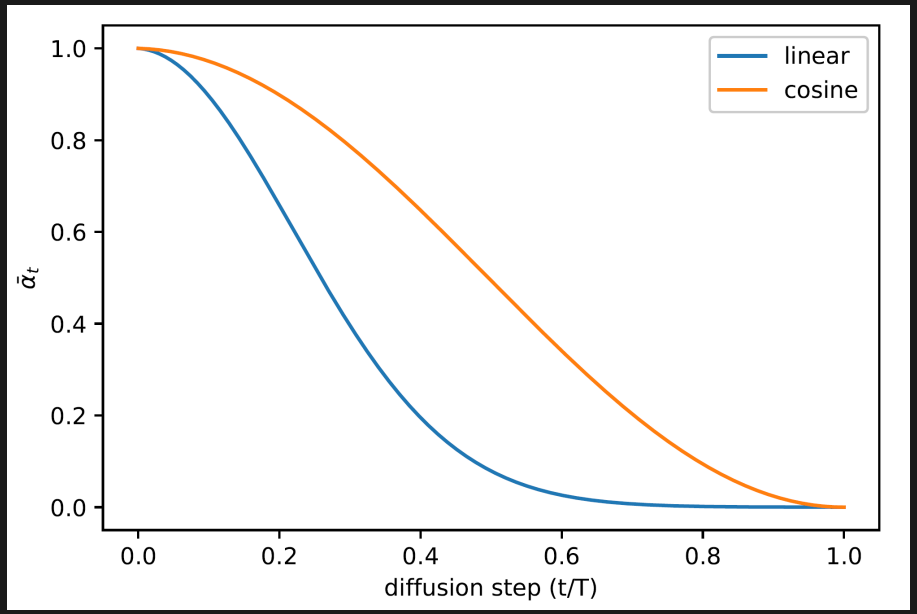
- Variance/noise schedulers
- Learning the reverse process variance
- Faster sampling: DDIMs
- Conditional generation
  - Classifier Guidance
  - Classifier-Free Guidance
  - Conditioning on images
  - ControlNet
  - Conditioning on text

Variance/noise schedulers

Nichol and Dhariwal [2021] propose a different way to set β_t:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \qquad \text{where} f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2$$

And then $\beta_t = \min\left(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999\right)$. s is a small offset to prevent β_t from being tiny when t is close to 0 (they set it to $s = 0.008$)



Comparison between scheduler in DDPM and Nichol & Dhariwal's cosine scheduler proposal. Source: Nichol & Dhariwal [2021]

Variance/noise schedulers

Source: Nichol & Dhariwal [2021]

More progressive forward diffusion process, especially for large values of t

Demo here!

Learning the reverse process variance

Nichol & Dhariwal [2021] also proposed to learn the reverse process variance $\Sigma_\theta(\mathbf{x}_t, t)$ instead of setting it upfront based on β_t:

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

So the learned variance is an interpolation between β_t and $\tilde{\beta}_t$ controlled by v, which is a mixing vector predicted by the model

Learning the reverse process variance

Nichol & Dhariwal [2021] also proposed to learn the reverse process variance $\Sigma_\theta(\mathbf{x}_t, t)$ instead of setting it upfront based on β_t:

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

So the learned variance is an interpolation between β_t and $\tilde{\beta}_t$ controlled by v, which is a mixing vector predicted by the model

Loss function is changed accordingly to include this (called now L_{hybrid} in the paper)

Faster sampling: DDIMs

The sampling algorithm in DDPMs is slow, as it requires T iterations $\rightarrow$ slow image generation

In *Denoising Diffusion Implicit Models* or **DDIMs**, Song et al. [2020] made the diffusion process non-Markovian, so each step t does not depend only on last step

Faster sampling: DDIMs

The sampling algorithm in DDPMs is slow, as it requires T iterations $\rightarrow$ slow image generation

In *Denoising Diffusion Implicit Models* or **DDIMs**, Song et al. [2020] made the diffusion process non-Markovian, so each step t does not depend only on last step

This allows us to "skip" steps during the sampling process

Faster sampling: DDIMs

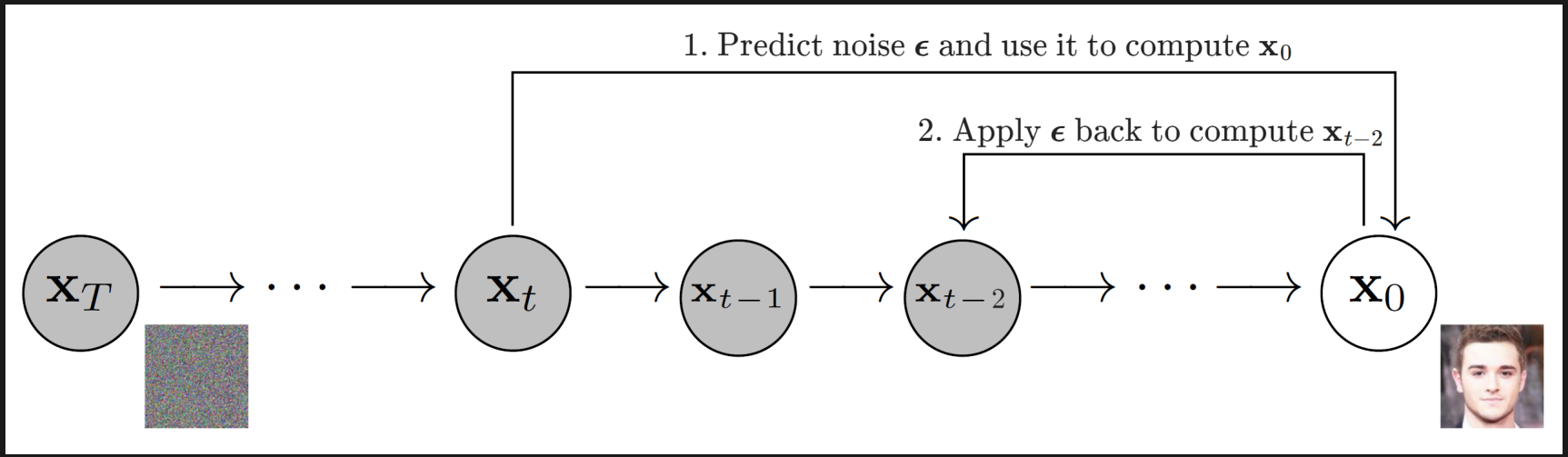
The sampling algorithm in DDPMs is slow, as it requires T iterations $\rightarrow$ slow image generation

In *Denoising Diffusion Implicit Models* or **DDIMs**, Song et al. [2020] made the diffusion process non-Markovian, so each step t does not depend only on last step

This allows us to "skip" steps during the sampling process

How? Predicting how to get from $\mathbf{x_t}$ to $\mathbf{x}_0$, and then "go back" from $\mathbf{x}_0$ to for instance a $\mathbf{x}_{t-2}$. Therefore, that iteration brings us from $\mathbf{x_t}$ directly to $\mathbf{x}_{t-2}$ (skipping 1 sampling step)

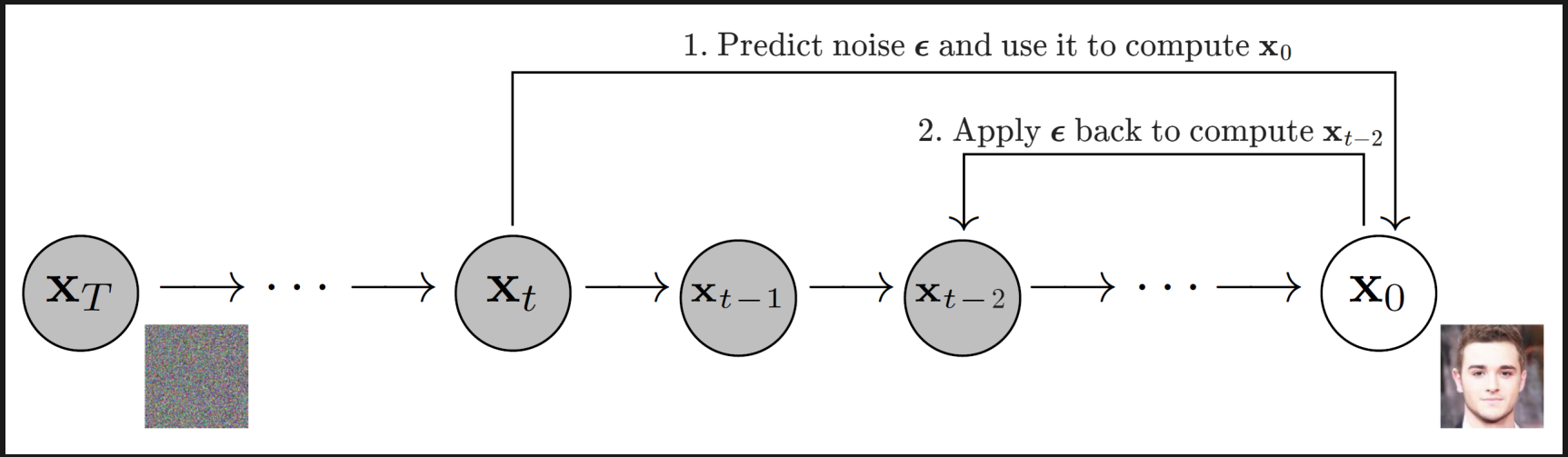
Faster sampling: DDIMs



DDIM for accelerated sampling (in this diagram it is used to jump from $\mathbf{x}_t$ directly to $\mathbf{x}_{t-2}$)

$$\mathbf{x}_{t-2} = \sqrt{\bar{\alpha}_{t-2}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-2}-\sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t, t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t \mathbf{z}}_{\text{random noise}}$$

Faster sampling: DDIMs



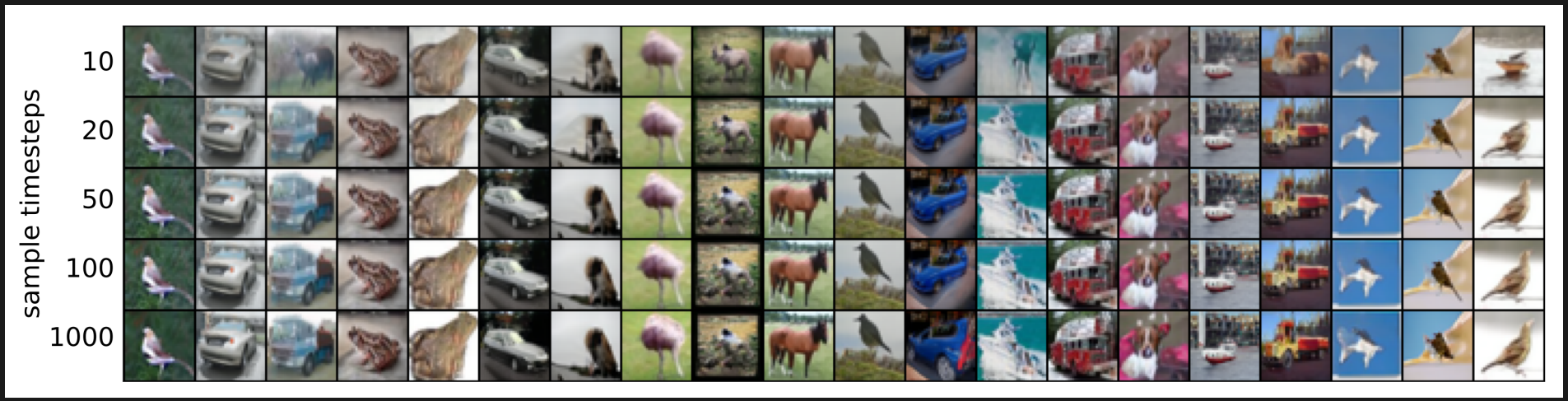
DDIM for accelerated sampling (in this diagram it is used to jump from $\mathbf{x}_t$ directly to $\mathbf{x}_{t-2}$)

$$\mathbf{x}_{t-2} = \sqrt{\bar{\alpha}_{t-2}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-2}-\sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t, t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t \mathbf{z}}_{\text{random noise}}$$

We can choose freely how many steps to skip per iteration (but quality can decrease if we skip too many!)

Faster sampling: DDIMs

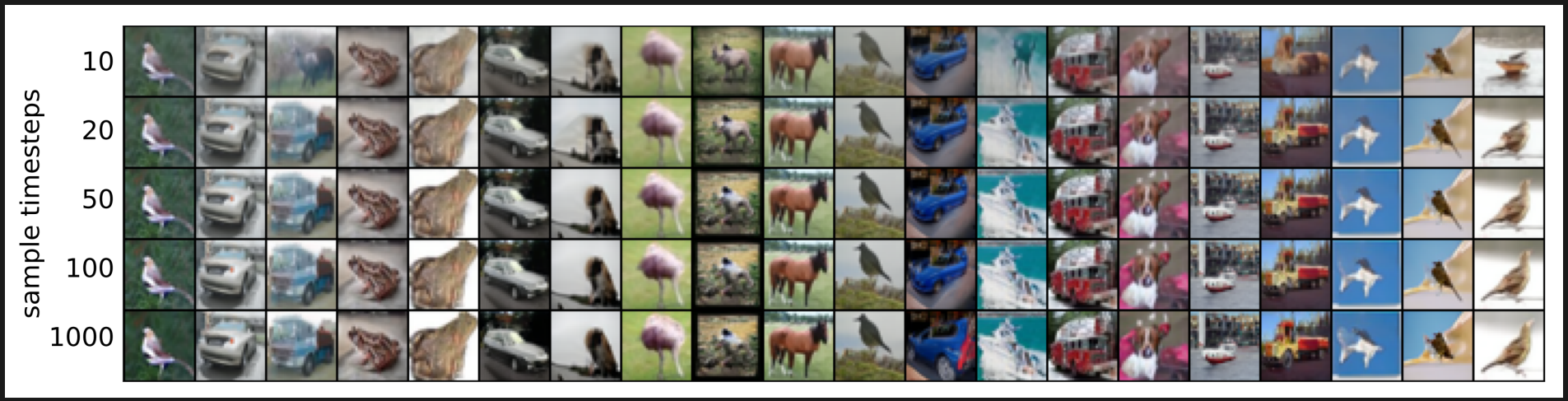
Most models can generate good quality images with few steps (such as 20 or 50)



Source: Song et al. [2020]

Faster sampling: DDIMs

Most models can generate good quality images with few steps (such as 20 or 50)



Source: Song et al. [2020]

Many other sampling algorithm variants have been proposed after DDIM—especially for Stable Diffusion models!

Faster sampling: DDIMs

$$\mathbf{x}_{t-2} = \sqrt{\bar{\alpha}_{t-2}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-2}-\sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t, t)}_{\text{direction pointing to } \mathbf{x_t}} + \underbrace{\sigma_t \mathbf{z}}_{\text{random noise}}$$

Authors also state that σ_t can be set to 0, making the sampling algorithm deterministic

Faster sampling: DDIMs

$$\mathbf{x}_{t-2} = \sqrt{\bar{\alpha}_{t-2}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-2}-\sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t, t)}_{\text{direction pointing to } \mathbf{x_t}} + \underbrace{\sigma_t \mathbf{z}}_{\text{random noise}}$$

Authors also state that σ_t can be set to 0, making the sampling algorithm deterministic

If done, the model is an *implicit probabilistic* one, hence the "I" in the name DDIM

Faster sampling: DDIMs

$$\mathbf{x}_{t-2} = \sqrt{\bar{\alpha}_{t-2}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-2}-\sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t, t)}_{\text{direction pointing to } \mathbf{x_t}} + \underbrace{\sigma_t \mathbf{z}}_{\text{random noise}}$$

Authors also state that σ_t can be set to 0, making the sampling algorithm deterministic

If done, the model is an *implicit probabilistic* one, hence the "I" in the name DDIM

Training does not change (same as in regular DDPMs)

Conditional generation

So far the diffusion models we have seen generate *any* image from pure noise; we don't have control over it

But it is much more useful to tell the model in some way what kind of image we want to generate (e.g. dog, house, "a polar bear with sunglasses surfing in space"...)

Conditional generation

So far the diffusion models we have seen generate *any* image from pure noise; we don't have control over it

But it is much more useful to tell the model in some way what kind of image we want to generate (e.g. dog, house, "a polar bear with sunglasses surfing in space"...)

Conditional generation: allows us to "inject" additional data to the model to obtain a specific kind of image

Conditional generation

So far the diffusion models we have seen generate *any* image from pure noise; we don't have control over it

But it is much more useful to tell the model in some way what kind of image we want to generate (e.g. dog, house, "a polar bear with sunglasses surfing in space"...)

Conditional generation: allows us to "inject" additional data to the model to obtain a specific kind of image

That additional data can be a class label (e.g. 0→dog, 1→cat, 2→house), a text prompt (a polar bear with sunglasses surfing in space), another image...

Conditional generation: Classifier Guidance

To perform conditional generation, Dhariwal and Nichol [2021] trained a regular image classifier $p_\phi(y \mid \mathbf{x}_t, t)$ using partially noisy images

They used the classifier gradients $\nabla_{\mathbf{x}_t} \log p_\phi(y \mid \mathbf{x}_t, t)$ to guide the sampling algorithm towards y (the class label e.g. 0→dog, 1→cat, 2→house)

Conditional generation: Classifier Guidance

To perform conditional generation, Dhariwal and Nichol [2021] trained a regular image classifier $p_\phi(y \mid \mathbf{x}_t, t)$ using partially noisy images

They used the classifier gradients $\nabla_{\mathbf{x}_t} \log p_\phi(y \mid \mathbf{x}_t, t)$ to guide the sampling algorithm towards y (the class label e.g. 0→dog, 1→cat, 2→house)

To do so, for the sampling algorithm they replace ϵ_θ with $\hat{\epsilon}$, which is:

$$\hat{\epsilon}(\mathbf{x}_t, t) = \epsilon_\theta(\mathbf{x}_t, t) - \sqrt{1 - \bar{\alpha}_t} \cdot s \cdot \nabla_{\mathbf{x}_t} \log p_\phi(y \mid \mathbf{x}_t, t)$$

Where s is a weight/scale factor that controls the guidance strength. Higher s→higher fidelity (but less diverse) images

Conditional generation: Classifier-Free Guidance

Cons of Classifier Guidance $\rightarrow$ having to train a separate classifier. Also: most information in the image $\mathbf{x}_t$ is not relevant for predicting y, and therefore taking its gradient w.r.t. $\mathbf{x}_t$ can yield somehow arbitrary guidance

Conditional generation: Classifier-Free Guidance

Cons of Classifier Guidance $\rightarrow$ having to train a separate classifier. Also: most information in the image $\mathbf{x}_t$ is not relevant for predicting y, and therefore taking its gradient w.r.t. $\mathbf{x}_t$ can yield somehow arbitrary guidance

Ho and Salimans [2022] proposed Classifier-Free Guidance, which avoids training a separate classifier

Conditional generation: Classifier-Free Guidance

Cons of Classifier Guidance $\rightarrow$ having to train a separate classifier. Also: most information in the image $\mathbf{x}_t$ is not relevant for predicting y, and therefore taking its gradient w.r.t. $\mathbf{x}_t$ can yield somehow arbitrary guidance

Ho and Salimans [2022] proposed Classifier-Free Guidance, which avoids training a separate classifier

Instead, we can get an "implicit classifier" by jointly training a conditional and unconditional diffusion model. Applying Bayes' theorem:

$$\underbrace{p(y \mid \mathbf{x}_t, t)}_{\text{classifier}} \propto \underbrace{p_\theta(\mathbf{x}_t \mid y, t)}_{\substack{\text{conditional} \\ \text{diffusion model}}} \; / \; \underbrace{p_\theta(\mathbf{x}_t \mid t)}_{\substack{\text{unconditional} \\ \text{diffusion model}}}$$

Both conditional and unconditional models can be the same one by training a conditional model $p_\theta(\mathbf{x}_t \mid y, t)$ for which the class y gets dropped at random during training with some probability (similar to what happens in dropout)

Conditional generation: Classifier-Free Guidance

Specifically, we can say that our model becomes $\epsilon_\theta(\mathbf{x}_t, t, y)$ with some probability of $y = \emptyset$ (this is, we sometimes feed the model with a special null class identifier instead of the real one during training)

The paper uses slightly different notation: they use $\mathbf{c}$ instead of y and t for the class and timestep feature respectively (they group them into $\mathbf{c}$), and use $\mathbf{z}_\lambda$ instead of $\mathbf{x}_t$

Conditional generation: Classifier-Free Guidance

Specifically, we can say that our model becomes $\epsilon_\theta(\mathbf{x}_t, t, y)$ with some probability of $y = \emptyset$ (this is, we sometimes feed the model with a special null class identifier instead of the real one during training)

In the sampling algorithm they replace ϵ_θ with $\tilde{\epsilon}_\theta$, which is a combination of the unconditional and conditional predictions:

$$\tilde{\epsilon}_\theta(\mathbf{x}_t, t, y) = (1 + w) \cdot \epsilon_\theta(\mathbf{x}_t, t, y) - w \cdot \epsilon_\theta(\mathbf{x}_t, t, y = \emptyset)$$

Where w controls the guidance strength

The paper uses slightly different notation: they use $\mathbf{c}$ instead of y and t for the class and timestep feature respectively (they group them into $\mathbf{c}$), and use $\mathbf{z}_\lambda$ instead of $\mathbf{x}_t$

Conditional generation: Classifier-Free Guidance

Source: Ho & Salimans [2022]

No guidance

$w = 1$

$w = 3$

Conditional generation: Conditioning on images

We can condition a diffusion model on something other than a class label. For instance, with other images

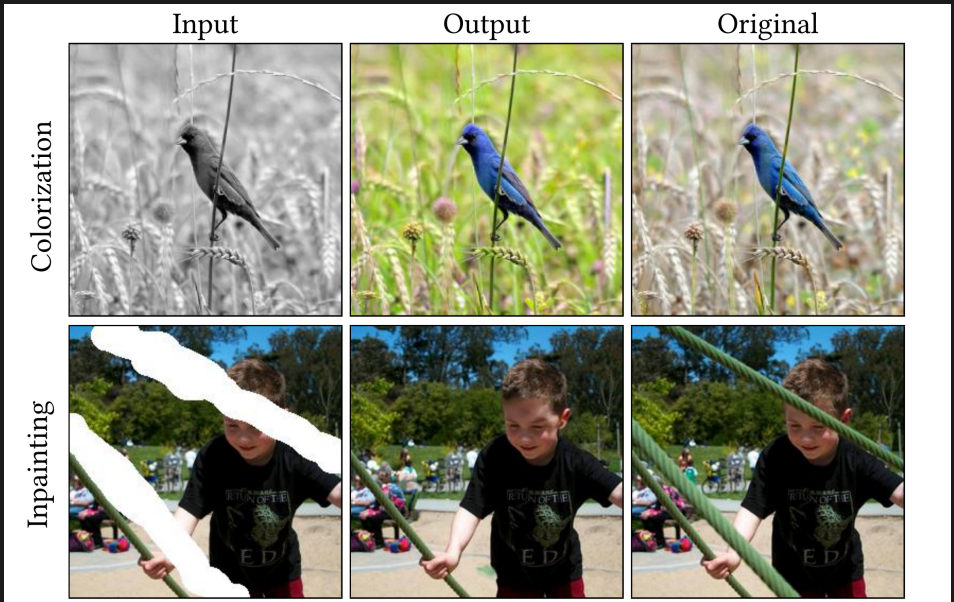
Palette by Saharia et al. [2022] performs image-to-image translation by training $p_\theta(\mathbf{x}_t \mid y, t)$ where y is an image we use to condition generation (given as input data to the model both during training and sampling)

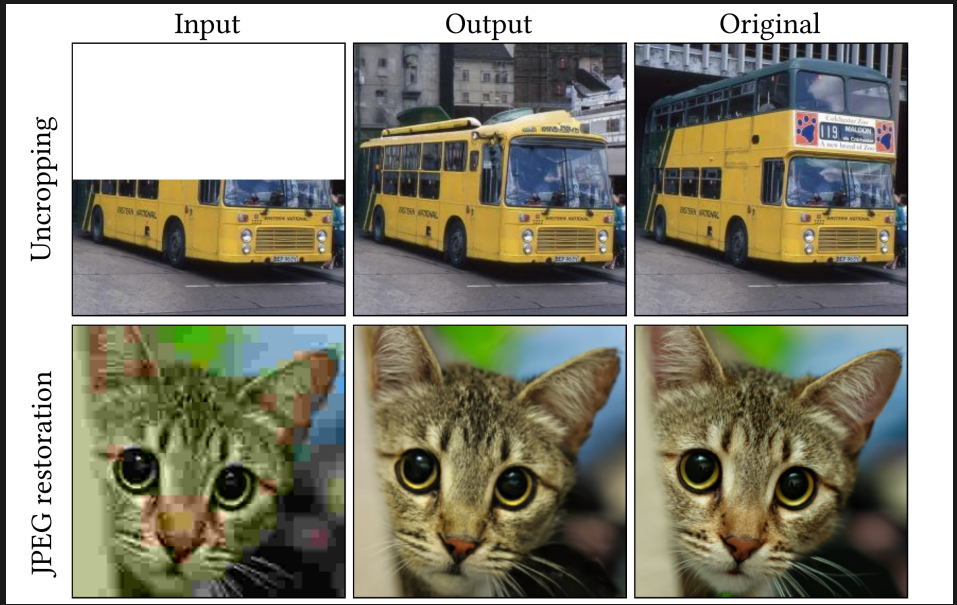
Conditional generation: Conditioning on images

We can condition a diffusion model on something other than a class label. For instance, with other images

Palette by Saharia et al. [2022] performs image-to-image translation by training $p_\theta(\mathbf{x}_t \mid y, t)$ where y is an image we use to condition generation (given as input data to the model both during training and sampling)

They use it to perform image colorization, where y is a black and white image that we want to colorize. Also for inpainting, image restoration...





Source: Saharia et al. [2022]

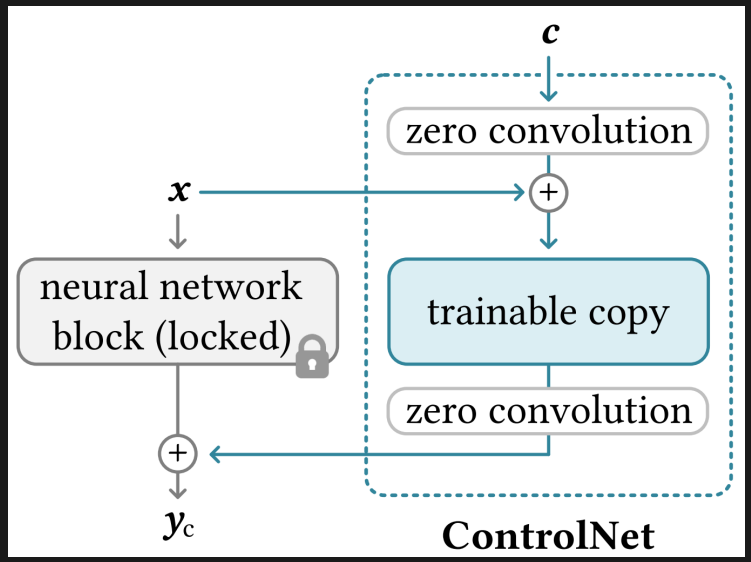
Conditional generation: ControlNet

Zhang et al. [2023] proposed ControlNet, where they fine-tuned a diffusion model to include extra conditioning such as sketch images, depth maps, human pose data, image segmentation data and others

Conditional generation: ControlNet

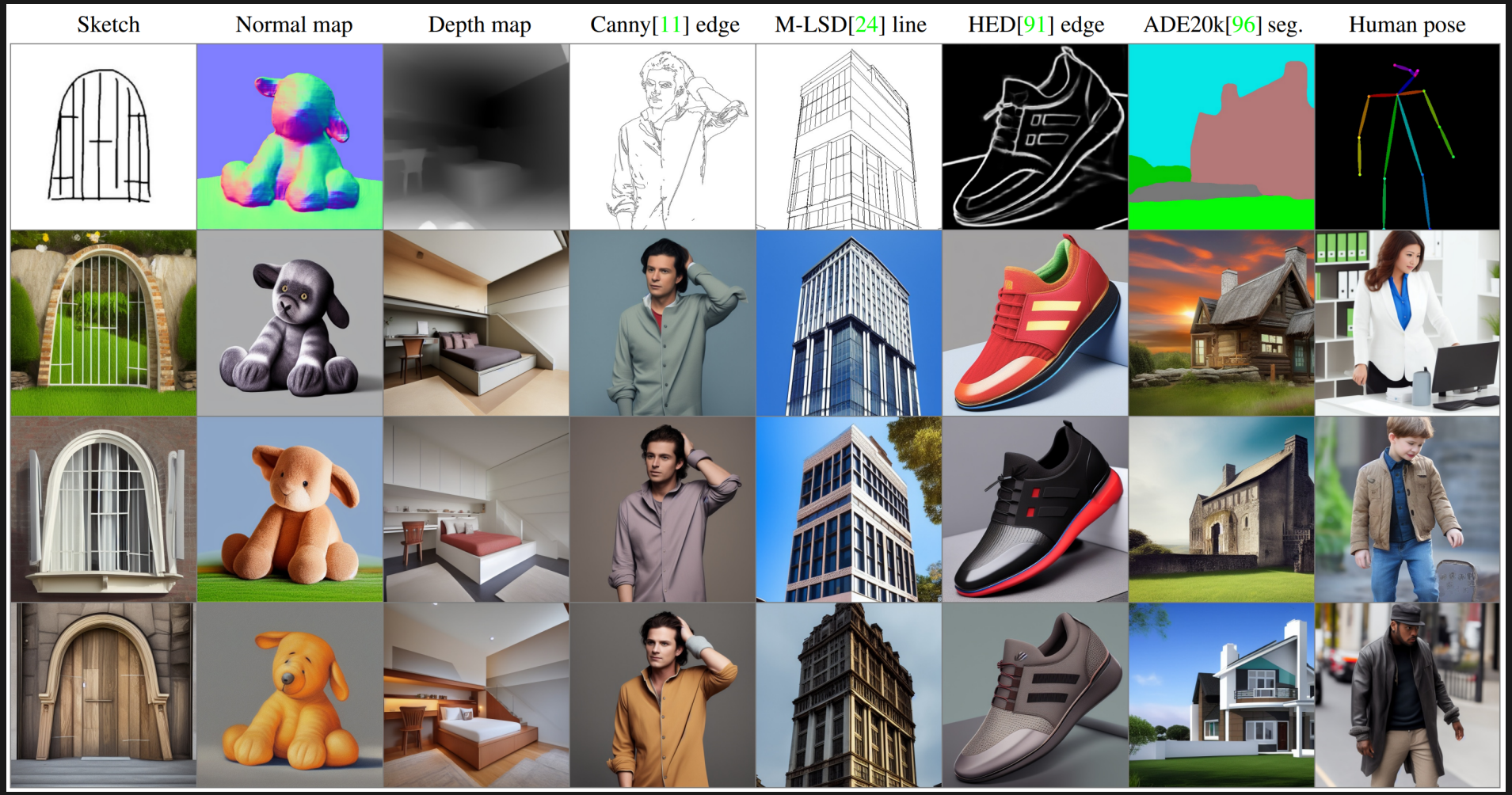
Zhang et al. [2023] proposed ControlNet, where they fine-tuned a diffusion model to include extra conditioning such as sketch images, depth maps, human pose data, image segmentation data and others

They did architectural changes to the diffusion model to accomodate the additional conditioning info, and then fine-tuned the model with datasets that include that extra conditioning data:



ControlNet block. They freeze the original pre-trained model layer, and make a trainable copy adding zero convolutions (1x1 conv layers with weights intialized to zeros) before and after the copied layer. x is what we know as $\mathbf{x}_t$ and $\mathbf{c}$ is the extra conditioning image/data. Frozen and ControlNet block outputs are added together. Source: Zhang et al. [2023]

Conditional generation: ControlNet



Source: Zhang et al. [2023]

Conditional generation: Conditioning on text

Diffusion models can be also conditioned in natural language through a *prompt* e.g. “a polar bear with sunglasses surfing in space”

This can be done with Classifier-Free Guidance, but making y be text (a list of tokens) instead of a class label, and adding some text-processing specific layers to the diffusion model (such as Transformer-like layers)

Conditional generation: Conditioning on text

Diffusion models can be also conditioned in natural language through a *prompt* e.g. "a polar bear with sunglasses surfing in space"

This can be done with Classifier-Free Guidance, but making y be text (a list of tokens) instead of a class label, and adding some text-processing specific layers to the diffusion model (such as Transformer-like layers)

Another option is to use a pre-trained text model and use its outputs to condition the diffusion model

A way to do this is to use *CLIP guidance*

Conditional generation: Conditioning on text

CLIP by Radford et al. [2021] is a discriminative (non-generative) model that uses contrastive learning to match images with their natural language descriptions

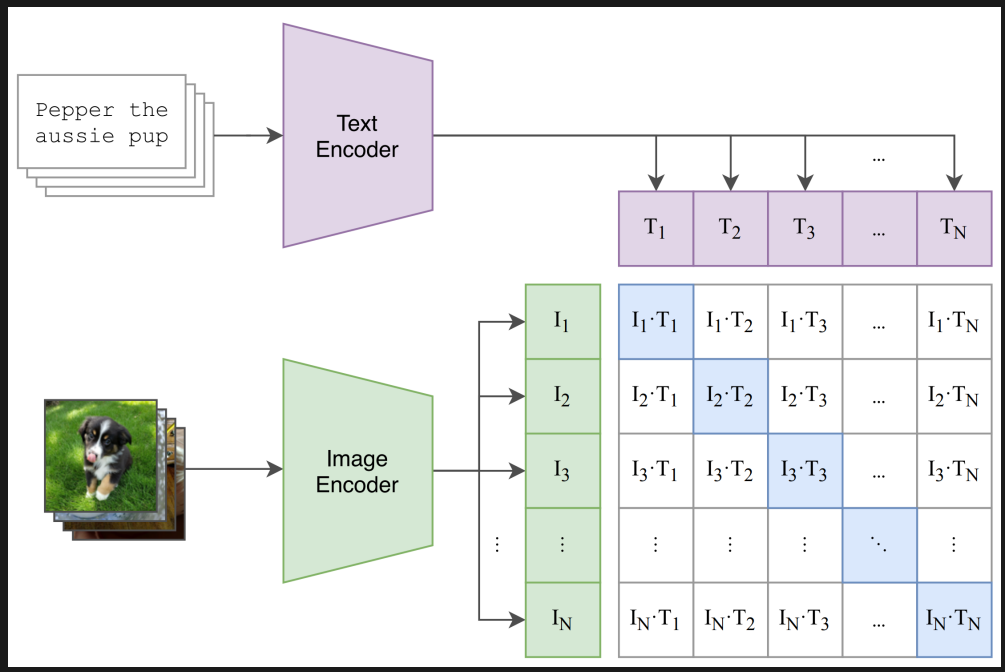
Made of a text encoder (Transformer) and an image encoder (Vision Transformer), it embeds both text and images into a common dimensional space

Conditional generation: Conditioning on text

CLIP by Radford et al. [2021] is a discriminative (non-generative) model that uses contrastive learning to match images with their natural language descriptions

Made of a text encoder (Transformer) and an image encoder (Vision Transformer), it embeds both text and images into a common dimensional space

The model is trained to maximize the (cosine) similarity between matching (image, text) pairs in this dimensional space



CLIP learns to maximize the cosine similarity (which can be thought of as correlation) between images and texts that go together. It therefore learns a sort of "correlation matrix" between training images $I_1, I_2, \ldots, I_N$ and training textual descriptions $T_1, T_2, \ldots, T_N$. Source: Radford et al. [2021]

Conditional generation: Conditioning on text

Once the CLIP model is trained, we can feed a text prompt to the text encoder and use the output as the conditioning for a diffusion model

CLIP guidance is applied in large diffusion models such as GLIDE and DALL-E 2, which we will cover next

Large diffusion models

Large diffusion models

Outline

- GLIDE
- DALL-E
- Imagen
- Stable Diffusion
- Flux
- Sana

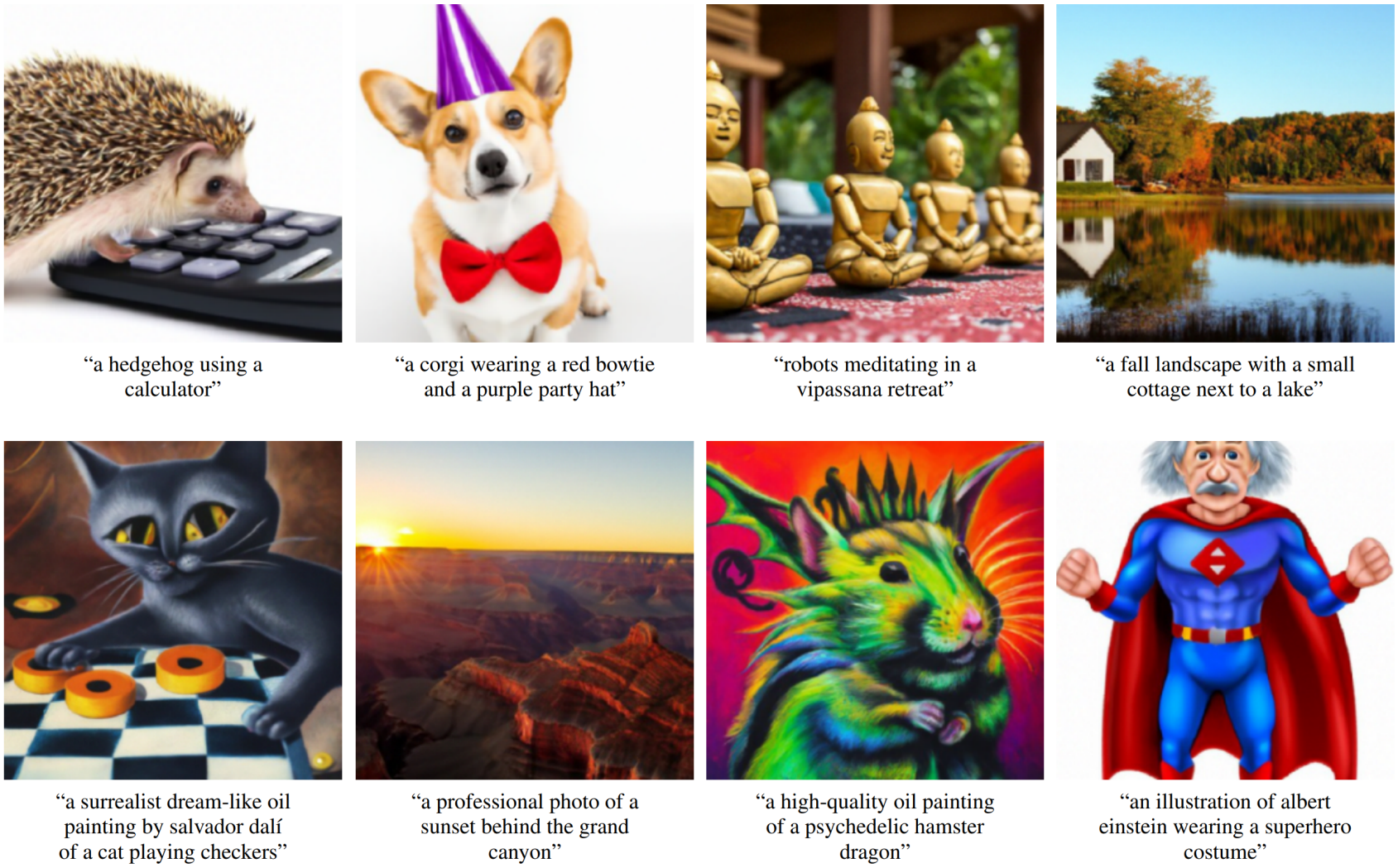
GLIDE

GLIDE (**G**uided **L**anguage to **I**mage **D**iffusion for generation and **E**diting) by Nichol et al. [2021] is a text-to-image model made of three sub-models:

- A base diffusion model that generates 64x64 images
- An upsampling diffusion model that increases the resolution to 256x256
- A Transformer-based text encoder to condition generation on text prompts (Classifier-Free Guidance)

They also tried to replace the Transformer with a pre-trained CLIP, but the Classifier-Free Guidance method worked better

GLIDE



Source: Nichol et al. [2021]

DALL-E

Dall-E 1 by Ramesh et al. [2021] was not a diffusion model, but Dall-E 2 and 3 are

Dall-E 2 (aka unCLIP) by Ramesh et al. [2022] uses a pre-trained CLIP model, and the text-to-image model itself is made of two main components:

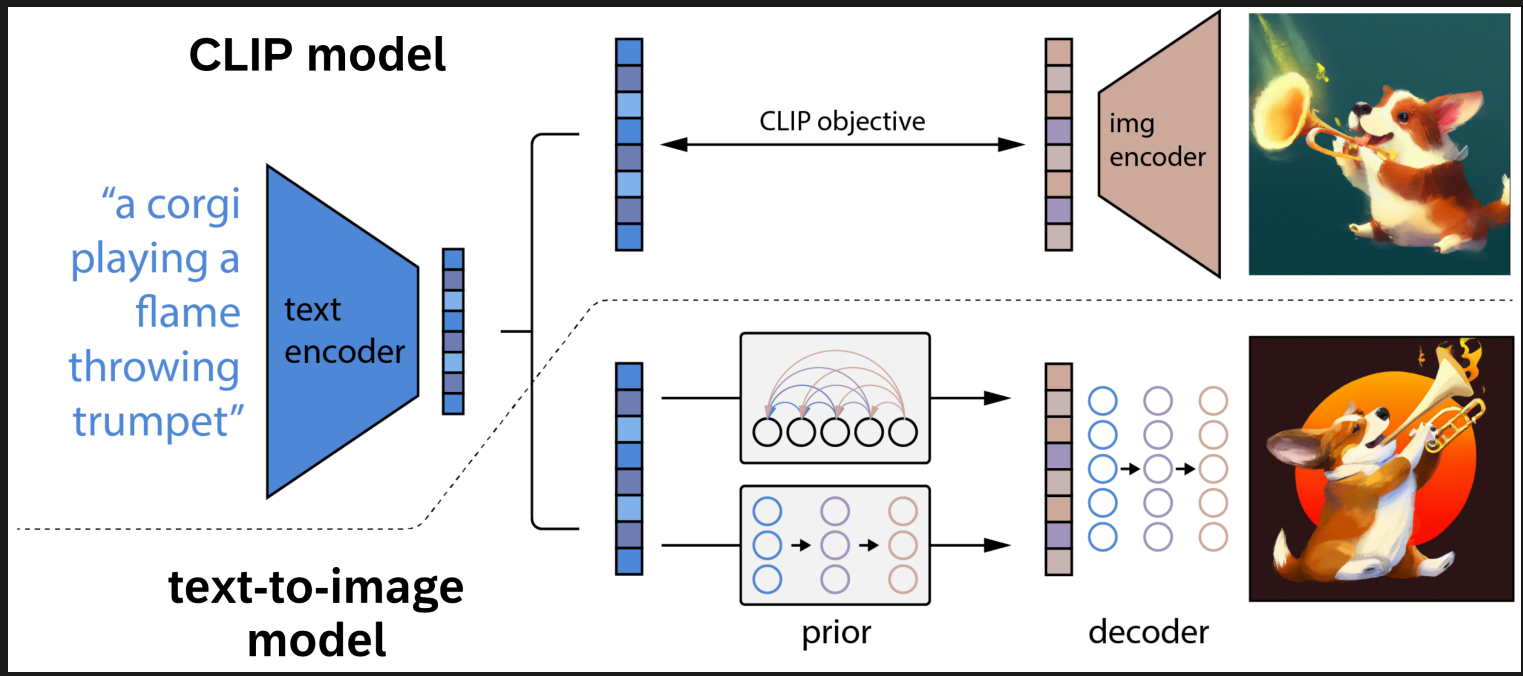
- A *prior* model which produces image embeddings based on the text encoded by CLIP
- A *decoder* model that generates the actual image based on the Prior's output and the original prompt

DALL-E

Dall-E 1 by Ramesh et al. [2021] was not a diffusion model, but Dall-E 2 and 3 are

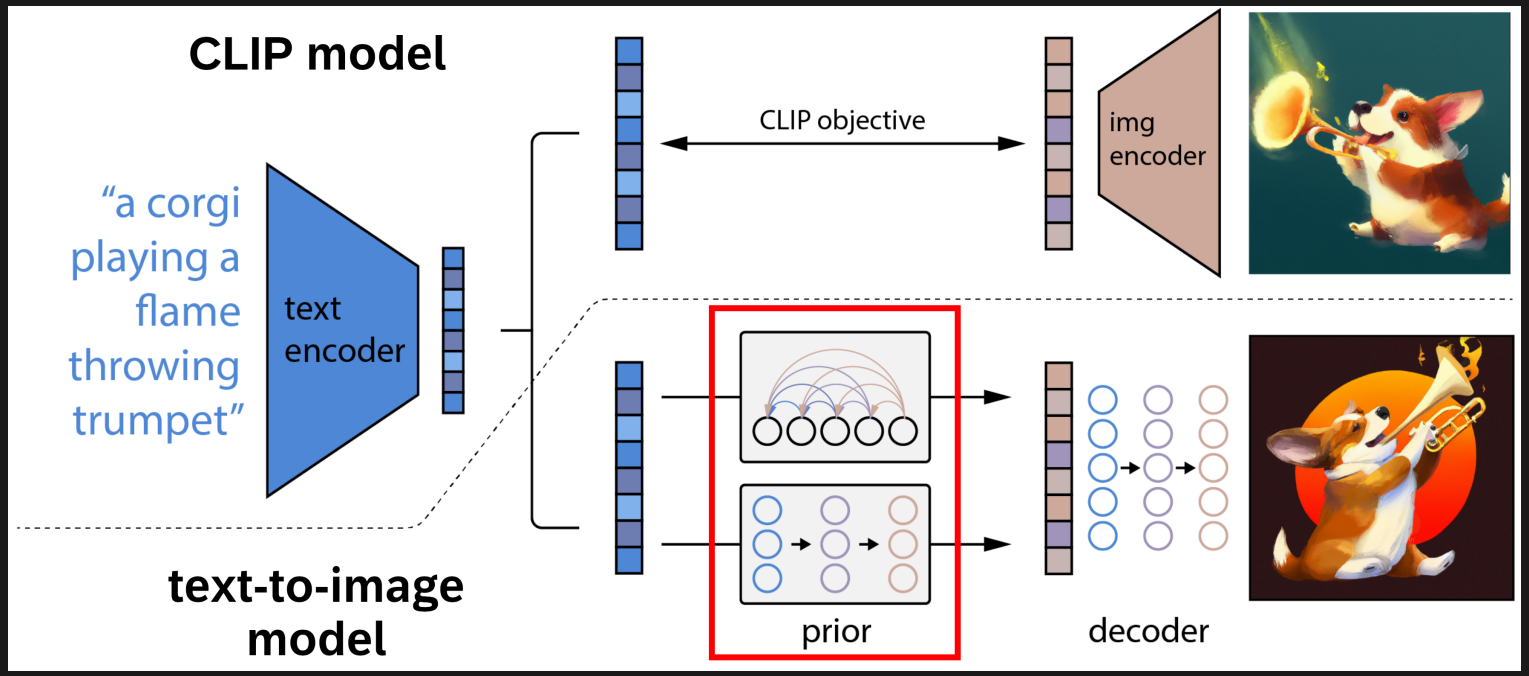
Dall-E 2 (aka unCLIP) by Ramesh et al. [2022] uses a pre-trained CLIP model, and the text-to-image model itself is made of two main components:

- A *prior* model which produces image embeddings based on the text encoded by CLIP
- A *decoder* model that generates the actual image based on the Prior's output and the original prompt



Source: adapted from Ramesh et al. [2022]

DALL-E

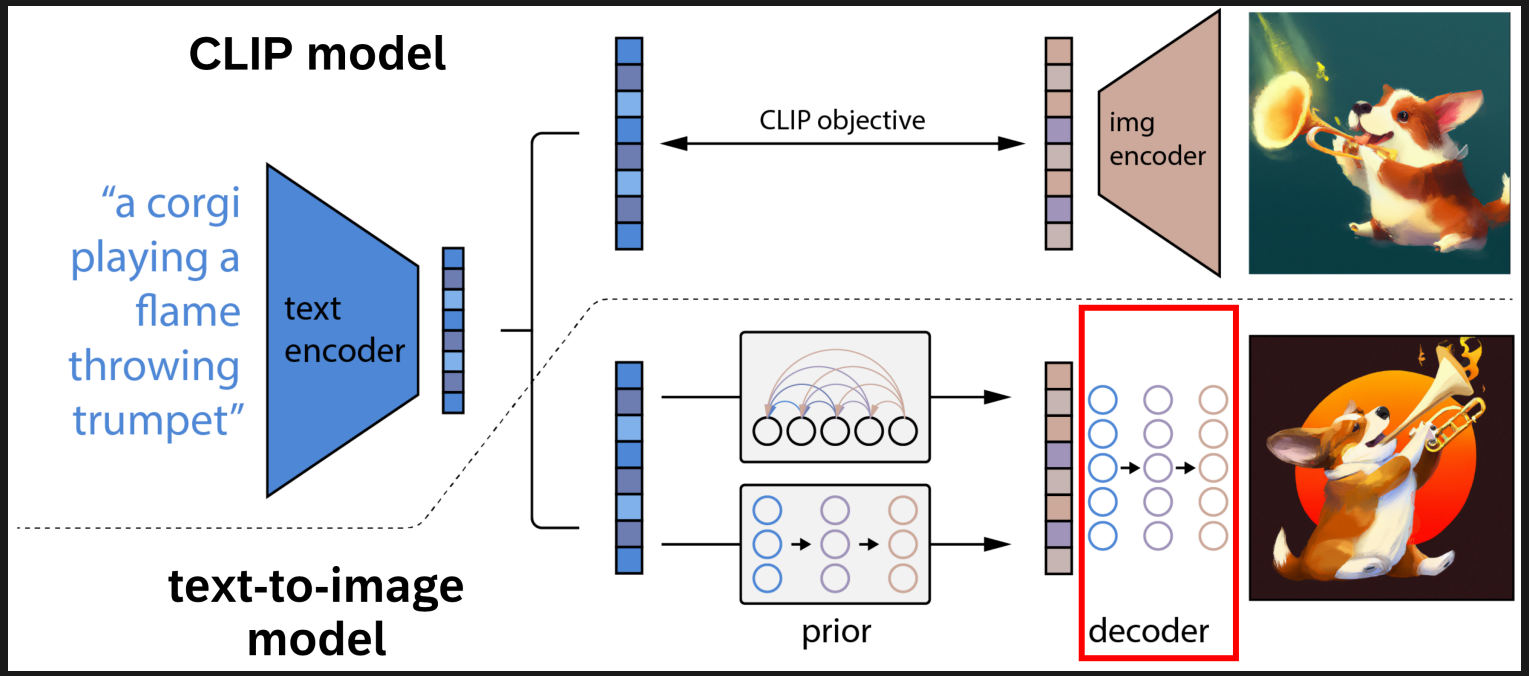


Authors tried two variants for the *prior* model:

- An autoregressive one (top one inside the red rectangle), where the image embeddings are generated as a discrete set of tokens (like in LLMs)
- A diffusion one (bottom on inside the red rectangle)

The diffusion prior worked better for them

DALL-E

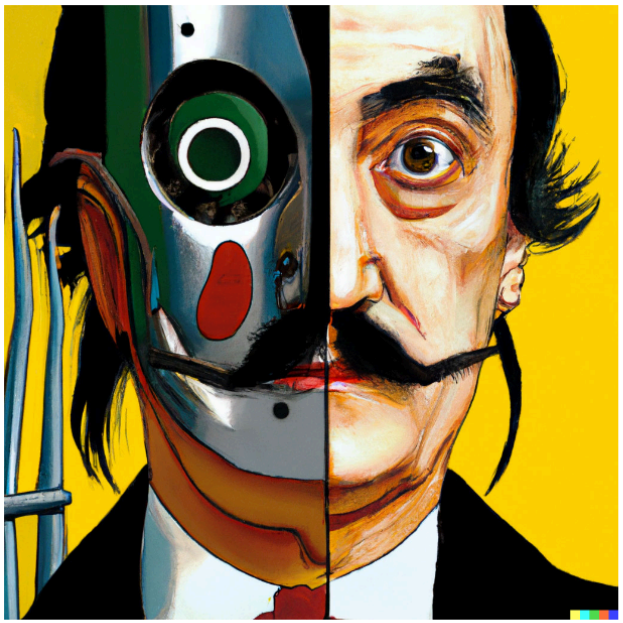


The decoder model is actually 3 diffusion models:

- One to generate 64x64 images
- Another one to upscale to 256x256
- A third one to upscale to 1024x1024

The prior output intermediate representation can be tweaked, allowing for text-guided image manipulation tasks

DALL-E



vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

a close up of a handpalm with leaves growing from it

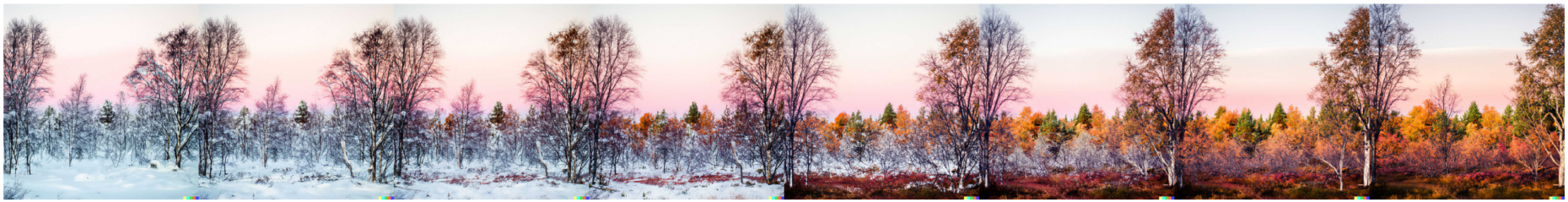
Source: Ramesh et al. [2022]

DALL-E

a photo of a cat → an anime drawing of a super saiyan cat, artstation

a photo of a victorian house → a photo of a modern house

a photo of an adult lion → a photo of lion cub



a photo of a landscape in winter → a photo of a landscape in fall

Source: Ramesh et al. [2022]

DALL-E

DALL-E 3 by Betker et al. [2023] improves on DALL-E 2 mostly by curating a better dataset

They train an image captioner to generate better textual descriptions of images

They also use GPT-4 to "upsample" (extend) text prompts before generating the image

DALL-E

DALL-E 3 by Betker et al. [2023] improves on DALL-E 2 mostly by curating a better dataset

They train an image captioner to generate better textual descriptions of images

They also use GPT-4 to "upsample" (extend) text prompts before generating the image

Model architecture is undisclosed

Imagen

Imagen by Saharia et al. [2022] also uses "cascaded" diffusion models

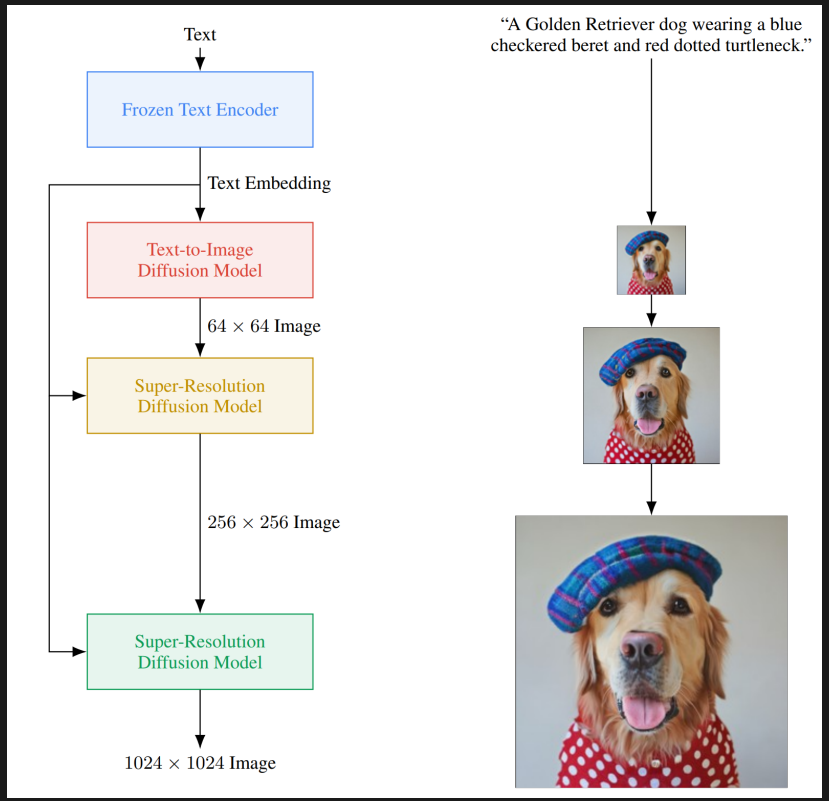
Uses a pre-trained text encoder LLM (T5-XXL by Raffel et al. [2019]) to generate text embeddings $\rightarrow$ much more effective than CLIP and quality improves with the LLM size (increasing text LLM size $>$ increasing diffusion models size)

Imagen

Imagen by Saharia et al. [2022] also uses "cascaded" diffusion models

Uses a pre-trained text encoder LLM (T5-XXL by Raffel et al. [2019]) to generate text embeddings $\rightarrow$ much more effective than CLIP and quality improves with the LLM size (increasing text LLM size $>$ increasing diffusion models size)

Used models are *Efficient U-Nets* which slightly alter the original U-Net



Source: Saharia et al. [2022]

Imagen

Sprouts in the shape of text 'Imagen' coming out of a fairytale book.

A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.

A high contrast portrait of a very happy fuzzy panda dressed as a chef in a high end kitchen making dough. There is a painting of flowers on the wall behind him.

Teddy bears swimming at the Olympics 400m Butterfly event.

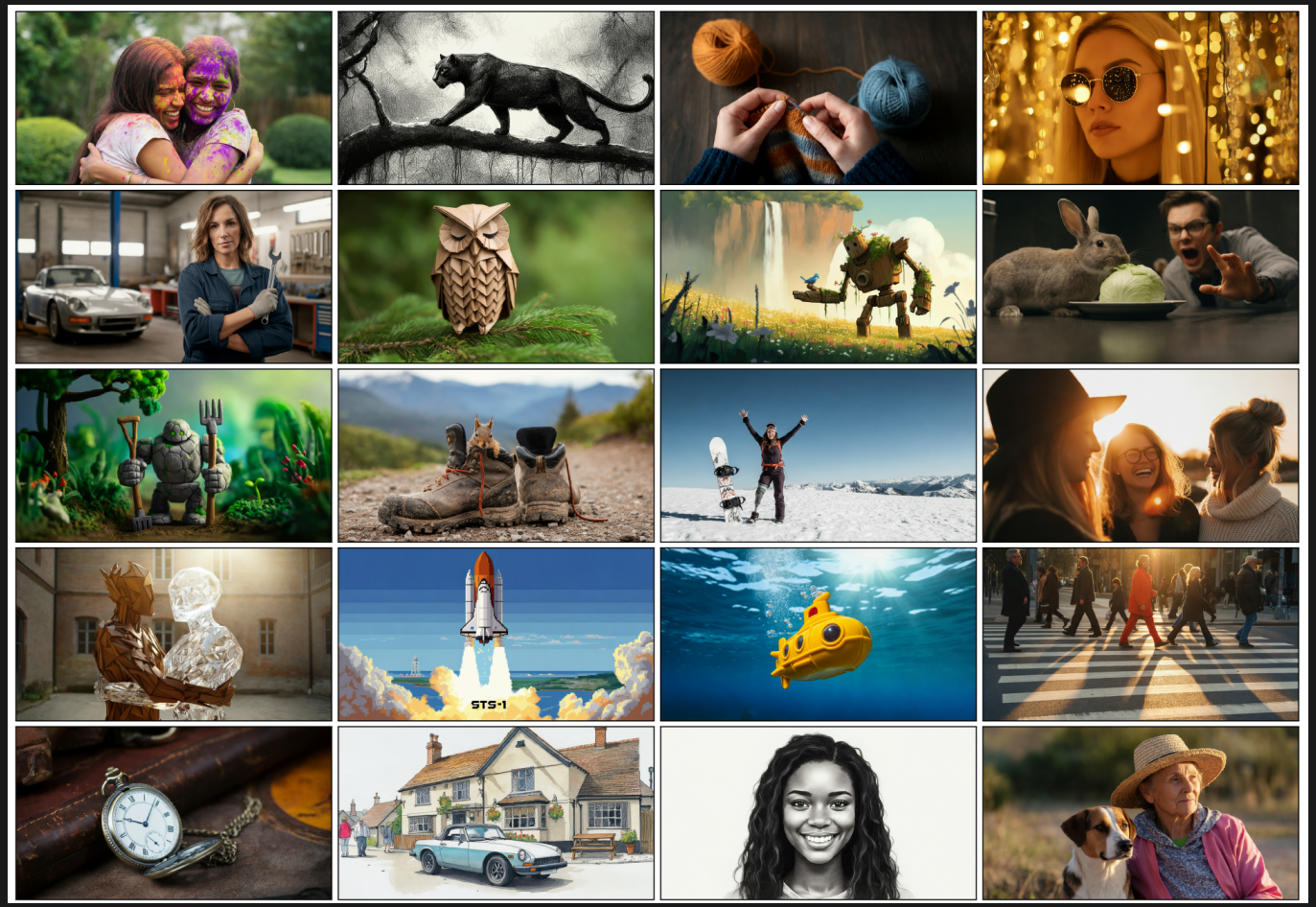
A cute corgi lives in a house made out of sushi.

A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.

Source: Saharia et al. [2022]

Imagen

Imagen 2 (2024) and 3 (Imagen 3 team [2024]) are more recent, but their architectures remain largely undisclosed



Source: Imagen 3 team [2024]

Stable Diffusion

Rombach et al. [2022] published a **Latent Diffusion Model** (LDM), which would serve as the basis for Stable Diffusion

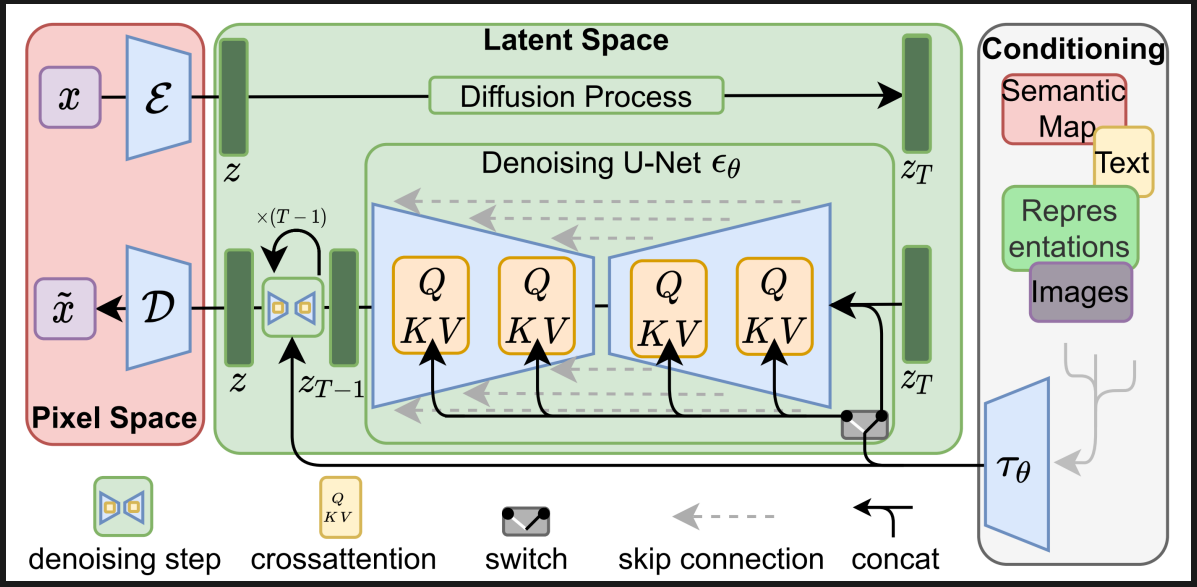
LDM makes image generation more efficient by running the diffusion process in a compressed, latent space instead of in the original image space

Stable Diffusion

Rombach et al. [2022] published a **Latent Diffusion Model** (LDM), which would serve as the basis for Stable Diffusion

LDM makes image generation more efficient by running the diffusion process in a compressed, latent space instead of in the original image space

It uses an Autoencoder to project images into that latent space, and runs the diffusion there



(Read in order: top-left→top-right→bottom-right→bottom-left): An encoder $\mathcal{E}$ projects an image x into a latent representation of it named as z, which then goes through the forward diffusion process (called Diffusion Process in the image) to get the noisy latent z_T. Then, the U-Net performs the reverse diffusion process to try to recover z, which gets unprojected using the decoder $\mathcal{D}$ back to a generated image $\tilde{x}$. Different conditionings (text, other images, etc) can be used to guide the generation process thanks to the condition encoders τ_θ and the crossattention layers. Source: Rombach et al. [2022]

Stable Diffusion

In June 2022, Stability AI and Runway released an open LDM which they called Stable Diffusion

Versions 1.X (June - Oct 2022) used OpenAI's pretrained CLIP for text conditioning and were capable of generating 512x512 images

Stable Diffusion

In June 2022, Stability AI and Runway released an open LDM which they called Stable Diffusion

Versions 1.X (June - Oct 2022) used OpenAI's pretrained CLIP for text conditioning and were capable of generating 512x512 images

Versions 2.X (Nov - Dec 2022) used a CLIP model trained by themselves, which was open sourced as OpenCLIP. Also increased resolution to 768x768

Stable Diffusion

In June 2022, Stability AI and Runway released an open LDM which they called Stable Diffusion

Versions 1.X (June - Oct 2022) used OpenAI's pretrained CLIP for text conditioning and were capable of generating 512x512 images

Versions 2.X (Nov - Dec 2022) used a CLIP model trained by themselves, which was open sourced as OpenCLIP. Also increased resolution to 768x768

XL versions (July - Nov 2023, Podell et al. [2023]) made the U-Net 3x larger and introduced a two-stage process: a base model that generates an image, and a *refiner* model that additional high-quality details to it. Resolution was increased to 1024x1024

Stable Diffusion

In June 2022, Stability AI and Runway released an open LDM which they called Stable Diffusion

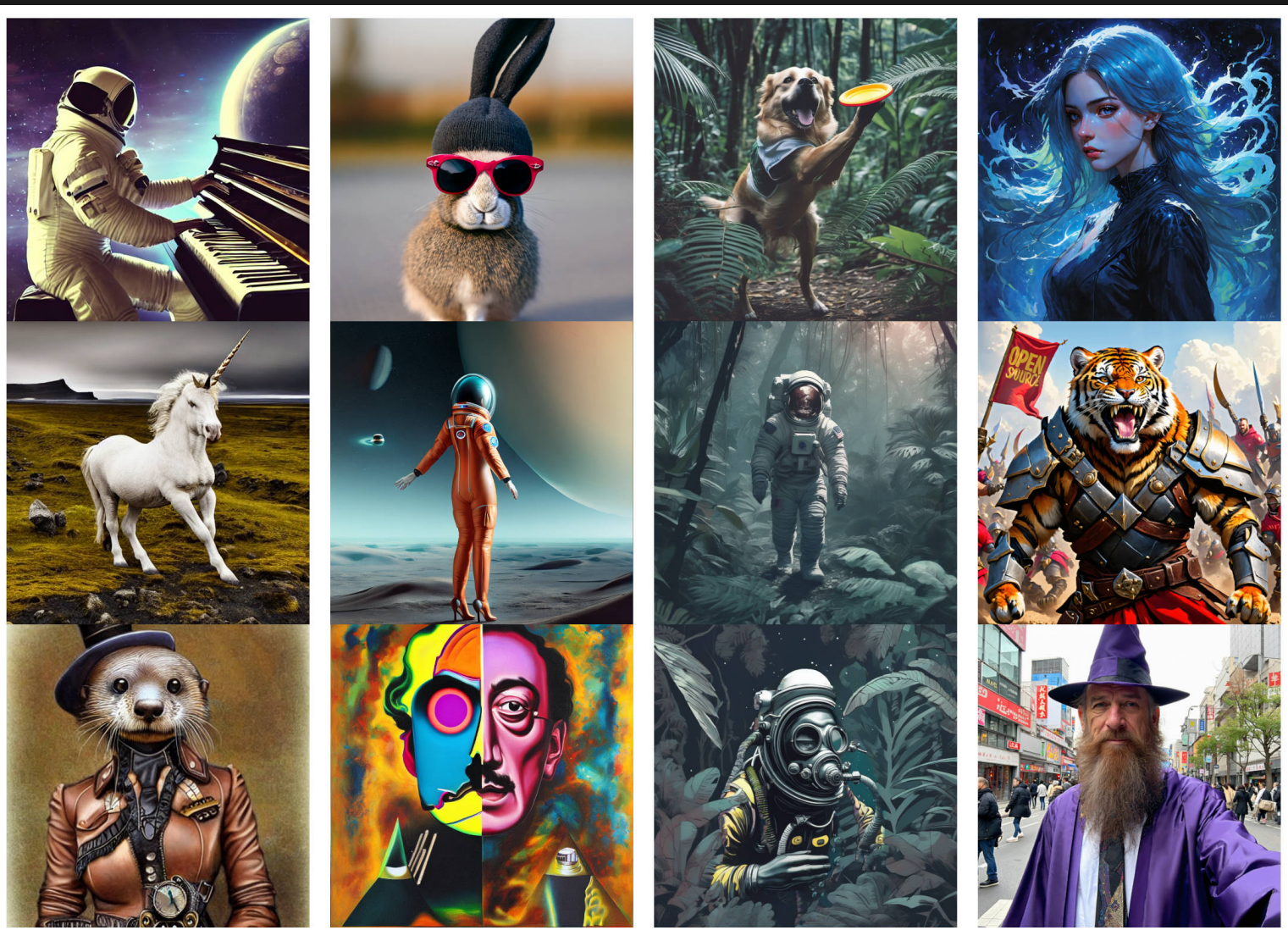
Versions 1.X (June - Oct 2022) used OpenAI's pretrained CLIP for text conditioning and were capable of generating 512x512 images

Versions 2.X (Nov - Dec 2022) used a CLIP model trained by themselves, which was open sourced as OpenCLIP. Also increased resolution to 768x768

XL versions (July - Nov 2023, Podell et al. [2023]) made the U-Net 3x larger and introduced a two-stage process: a base model that generates an image, and a *refiner* model that additional high-quality details to it. Resolution was increased to 1024x1024

Version 3 (Esser et al. [2024]) made many architectural changes, such as replacing CLIP as the text encoder with T5-XXL (Raffel et al. [2020]). But its release was controversial (both because of its unclear licensing and generation issues)

Stable Diffusion



SD 1 **SD 2** **SD XL** **SD 3**

Sources: https://github.com/CompVis/stable-diffusion, https://github.com/Stability-AI/stablediffusion, https://github.com/huggingface/diffusers/blob/main/docs/source/en/using-diffusers/sdxl.md, https://huggingface.co/stabilityai/stable-diffusion-3-medium [2024]

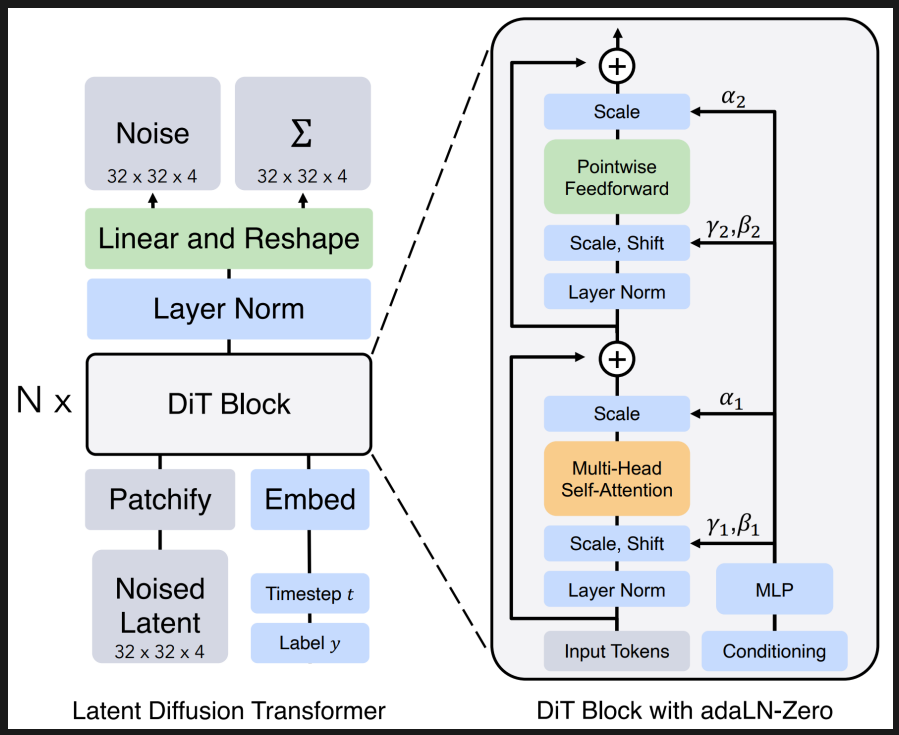
Flux

Authors of the LDM model founded Black Forest Labs in Aug 2024 and released inference code for a model called Flux

Flux

Authors of the LDM model founded Black Forest Labs in Aug 2024 and released inference code for a model called Flux

Architecturally very similar to Stable Diffusion 3, it replaces the U-Net with a **Diffusion Transformer (DiT)** (Peebles and Xie [2023])



Just like LDM, DiT operates on the latent space. Like Vision Transformers (ViTs), it transforms the data into a sequence of "patches", which go through N DiT blocks. They tried three different block architectures, being adaLN(adaptive Layer Norm)-Zero the one which gave the best results, as depicted in the right. Source: Peebles and Xie [2023]

Flux

Like Stable Diffusion 3, Flux uses **Flow Matching** (Lipman et al. [2023]) to formulate, train and sample from the model

Flow matching is based on Continuous Normalizing Flows, and leverages generative modeling by progressively morphing a distribution into another one

Flux

Like Stable Diffusion 3, Flux uses **Flow Matching** (Lipman et al. [2023]) to formulate, train and sample from the model

Flow matching is based on Continuous Normalizing Flows, and leverages generative modeling by progressively morphing a distribution into another one

Diffusion is a specific case of Flow Matching, where the morphing is done by adding random Normal noise to the data

Other Flow Matching formulations (like when using *Conditional Optimal Transport*) can be more efficient to train and sample from, at the expense of a bit of quality (which can be offset by more data)

Flux

Like Stable Diffusion 3, Flux uses **Flow Matching** (Lipman et al. [2023]) to formulate, train and sample from the model

Flow matching is based on Continuous Normalizing Flows, and leverages generative modeling by progressively morphing a distribution into another one

Diffusion is a specific case of Flow Matching, where the morphing is done by adding random Normal noise to the data

Other Flow Matching formulations (like when using *Conditional Optimal Transport*) can be more efficient to train and sample from, at the expense of a bit of quality (which can be offset by more data)

We refer to the paper above and this video by Lipman for further details

Flux

Source: https://blackforestlabs.ai/announcing-black-forest-labs/ [2024]

Sana

Sana by Xie et al. [2024] includes innovations such as:

- A *Deep Compression Autoencoder* (DC-AE), which compresses/decompresses aggresively input images up to 32x $\rightarrow$ efficient generation of 4K images
- A linear DiT that performs efficient sub-quadratic attention
- Uses Google's pre-trained Gemma-2 (Gemma Team [2024]) decoder-only LLM to perform text conditioning $\rightarrow$ they extract last layer's output as textual embeddings

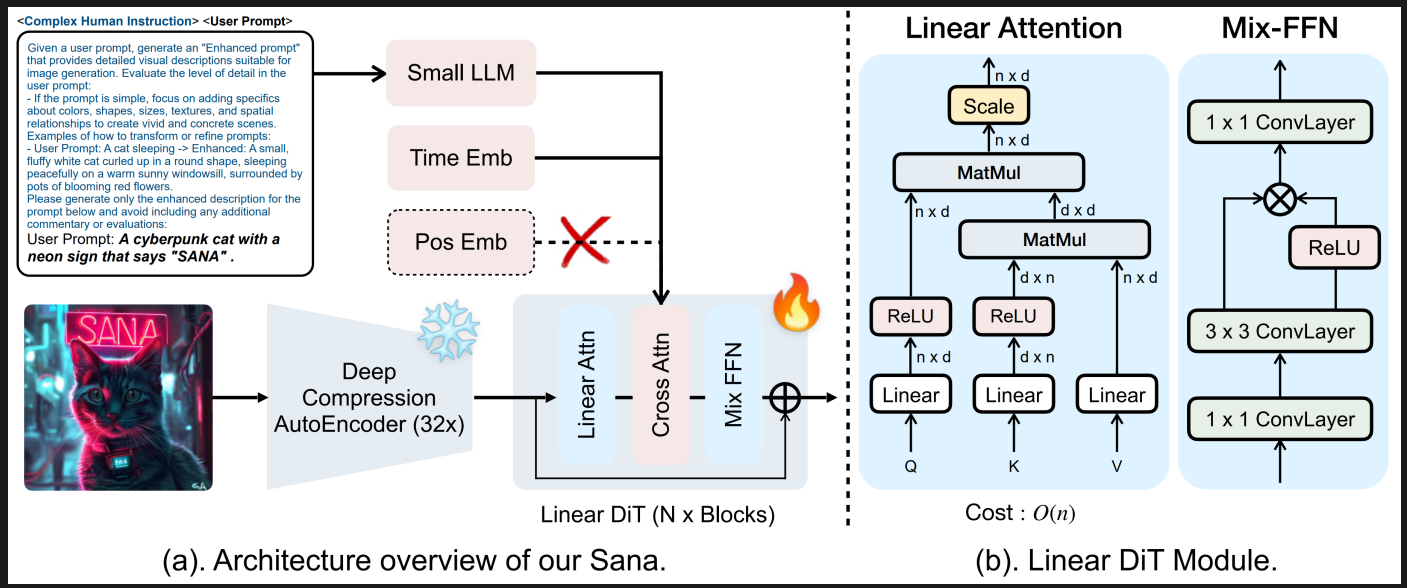
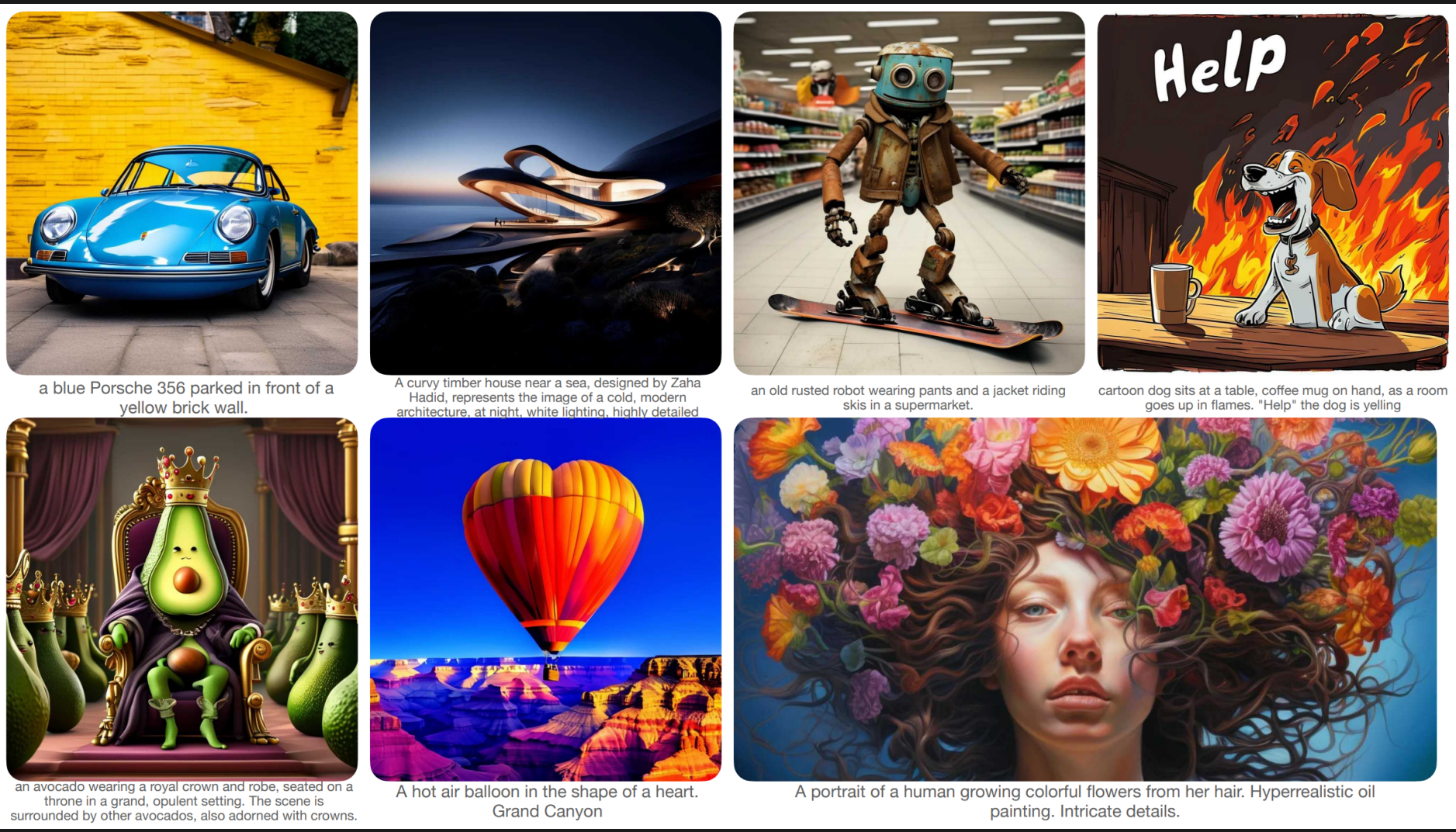


Figure (a) shows the model architecture, including a sample prompt for Gemma-2. Positional embedding is not required in the linear DiT. Figure (b) shows details on the linear DiT block. Source: Xie et al. [2024]

Sana



Source: Xie et al. [2024]

Beyond image generation

Diffusion for video

Image generator diuffusion models work with 4D data in the form of (*Batch*, *Channels*, *Height*, *Width*)

Diffusion-based video generation models work by mostly adapting the model to handle 5D data (*Batch*, *Time*, *Channels*, *Height*, *Width*) in various ways

Diffusion for video

Image generator diuffusion models work with 4D data in the form of (*Batch*, *Channels*, *Height*, *Width*)

Diffusion-based video generation models work by mostly adapting the model to handle 5D data (*Batch*, *Time*, *Channels*, *Height*, *Width*) in various ways

Some of the most prominent publications are:

- Video Diffusion Models by Ho et al. [2022]
- Imagen Video: High Definition Video Generation with Diffusion Models by Ho et al. [2022] and Veo by Veo team [2024]
- Photorealistic Video Generation with Diffusion Models by Gupta et al. [2023]
- Sora by Brooks et al. [2024]
- Movie Gen by The Movie Gen Team @ Meta [2024]
- CogVideoX by Yang et al. [2024]
- Pyramid Flow by Jin et al. [2024]
- Mochi 1 by Genmo Team [2024]

Diffusion for video

Source: The Movie Gen Team @ Meta [2024]

Diffusion for other applications

Some examples (by no means a thorough list):

3D rendering and reconstruction:

- DreamFusion: Text-to-3D using 2D Diffusion by Poole et al. [2022]
- CAT3D: Create Anything in 3D with Multi-View Diffusion Models by Gao et al. [2024]

Text generation:

- Diffusion-LM Improves Controllable Text Generation by Lisa Li et al. [2024]

Music and audio generation:

- Noise2Music: Text-conditioned Music Generation with Diffusion Models by Huang et al. [2023]
- Fast Timing-Conditioned Latent Audio Diffusion by Evans et al. [2024]
- QA-MDT: Quality-aware Masked Diffusion Transformer for Enhanced Music Generation by Li et al. [2024]

Diffusion for other applications

Text-to-speech:

- Better speech synthesis through scaling by Betker [2023]
- NaturalSpeech 3: Zero-Shot Speech Synthesis with Factorized Codec and Diffusion Models by Ju et al. [2024]
- F5-TTS: A Fairytaler that Fakes Fluent and Faithful Speech with Flow Matching by Chen et al. [2024]

Life sciences:

- DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking by Corso et al. [2022]
- Protein generation with evolutionary diffusion: sequence is all you need by Alamdari et al. [2023]
- Accurate structure prediction of biomolecular interactions with AlphaFold 3 by Abramson et al. [2024]

Diffusion for other applications

Robotics:

- Diffusion Policy: Visuomotor Policy Learning via Action Diffusion by Chi et al. [2023]

Videogame engine creation:

- Diffusion Models Are Real-Time Game Engines by Valevski et al. [2024]

And many more (as there are more than 4,000 papers on arXiv at the time of writing!)

Recent survey papers like A Survey on Generative Diffusion Models by Cao et al. [2023] can be useful to stay up to date

How to use diffusion models?

Do-It-Yourself:

- PyTorch

Training and fine-tuning of existing architectures:

- 🤗 Diffusers
- Ostris AI toolkit
- Bghira's SimpleTuner
- Kohya

Inference and image generation GUIs (no-code or low-code), from simpler to more complex:

- Automatic1111
- ForgeUI → very similar to A1111, but faster and more modern
- SwarmUI
- ComfyUI → tons of features; professional tool

This list gets outdated very quickly! Check for community updates on r/StableDiffusion and Civitai

Some interesting topics we skipped

Connection to score-based generative models (what if t was continuous instead of discrete?). Diffusion expressed as Ordinary and Stochastic Differential Equations:

- Generative Modeling by Estimating Gradients of the Data Distribution by Song and Ermon [2019]
- Improved Techniques for Training Score-Based Generative Models by Song and Ermon [2020]
- Score-Based Generative Modeling through Stochastic Differential Equations by Song et al. [2020]
- Elucidating the Design Space of Diffusion-Based Generative Models by Karras et al. [2022]

Perform sampling in even less steps:

- Progressive Distillation for Fast Sampling of Diffusion Models by Salimans and Ho [2022]
- Consistency Models by Song et al. [2023]
- Simple and Fast Distillation of Diffusion Models by Zhou et al. [2024]

Zero-terminal Signal-to-Noise Ratio (SNR):

- Common Diffusion Noise Schedules and Sample Steps are Flawed by Lin et al. [2024]

Some interesting topics we skipped

Improve resolution through cascaded generation, used in e.g. DALL-E 2 and Imagen:

- Cascaded Diffusion Models for High Fidelity Image Generation by Ho et al. [2022]

Subject-driven image generation through fine-tuning (add specific characters to images):

- An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion by Gal et al. [2022]
- DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation by Ruiz et al. [2022]

Image and video editing:

- SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations by Meng et al. [2021]
- Prompt-to-Prompt Image Editing with Cross Attention Control by Hertz et al. [2022]
- RePaint: Inpainting using Denoising Diffusion Probabilistic Models by Lugmayr et al. [2022]
- Differential Diffusion: Giving Each Pixel Its Strength by Levin et al. [2023]
- Tune-A-Video: One-Shot Tuning of Image Diffusion Models for Text-to-Video Generation by Wu et al. [2023]
- Dreamix: Video Diffusion Models are General Video Editors by Moland et al. [2024]

Some interesting topics we skipped

Diffusion Transformer variants and improvements:

- PixArt-α: Fast Training of Diffusion Transformer for Photorealistic Text-to-Image Synthesis by Chen et al. [2024]
- Dynamic Diffusion Transformer by Zhao et al. [2024]

ControlNet variants:

- Uni-ControlNet: All-in-One Control to Text-to-Image Diffusion Models by Zhao et al. [2023]
- ControlNet++: Improving Conditional Controls with Efficient Consistency Feedback by Li et al. [2024]
- CtrLoRA: An Extensible and Efficient Framework for Controllable Image Generation by Xu et al. [2024]

Flow Matching variants and improvements:

- InstaFlow: One Step is Enough for High-Quality Diffusion-Based Text-to-Image Generation by Liu et al. [2023]
- Rectified Diffusion: Straightness Is Not Your Need in Rectified Flow} by Wang et al. [2024]

Some interesting topics we skipped

General multi-task diffusion models:

- OmniGen: Unified Image Generation by Xiao et al. [2024]

References I

[1] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J. Ballard, Joshua Bambrick, Sebastian W. Bodenstein, David A. Evans, Chia-Chun Hung, Michael O'Neill, David Reiman, Kathryn Tunyasuvunakool, Zachary Wu, Akvilė Žemgulytė, Eirini Arvaniti, Charles Beattie, Ottavia Bertolli, Alex Bridgland, Alexey Cherepanov, Miles Congreve, Alexander I. Cowen-Rivers, Andrew Cowie, Michael Figurnov, Fabian B. Fuchs, Hannah Gladman, Rishub Jain, Yousuf A. Khan, Caroline M. R. Low, Kuba Perlin, Anna Potapenko, Pascal Savy, Sukhdeep Singh, Adrian Stecula, Ashok Thillaisundaram, Catherine Tong, Sergei Yakneen, Ellen D. Zhong, Michal Zielinski, Augustin Žídek, Victor Bapst, Pushmeet Kohli, Max Jaderberg, Demis Hassabis, and John M. Jumper. 2024. Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature* 630, 8016 (June 2024), 493–500. https://doi.org/10.1038/s41586-024-07487-w

[2] Sarah Alamdari, Nitya Thakkar, Rianne van den Berg, Alex X. Lu, Nicolo Fusi, Ava P. Amini, and Kevin K. Yang. 2023. Protein generation with evolutionary diffusion: Sequence is all you need. *bioRxiv* (2023). https://doi.org/10.1101/2023.09.11.556673

[3] James Betker. 2023. Better speech synthesis through scaling. Retrieved from https://arxiv.org/abs/2305.07243

[4] James Betker, Gabriel Goh, Li Jing, † TimBrooks, Jianfeng Wang, Linjie Li, † LongOuyang, † JuntangZhuang, † JoyceLee, † YufeiGuo, † WesamManassra, † PrafullaDhariwal, † CaseyChu, † YunxinJiao, and Aditya Ramesh. Improving image generation with better captions. Retrieved from https://api.semanticscholar.org/CorpusID:264403242

[5] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. 2024. Video generation models as world simulators. (2024). Retrieved from https://openai.com/research/video-generation-models-as-world-simulators

[6] H. Cao, C. Tan, Z. Gao, Y. Xu, G. Chen, P. Heng, and S. Z. Li. 2024. A survey on generative diffusion models. *IEEE Transactions on Knowledge & Data Engineering* 36, 07 (July 2024), 2814–2830. https://doi.org/10.1109/TKDE.2024.3361474

[7] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. 2023. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. Retrieved from https://arxiv.org/abs/2310.00426

[8] Yushen Chen, Zhikang Niu, Ziyang Ma, Keqi Deng, Chunhui Wang, Jian Zhao, Kai Yu, and Xie Chen. 2024. F5-TTS: A fairytaler that fakes fluent and faithful speech with flow matching. *arXiv preprint arXiv:2410.06885* (2024).

[9] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. 2023. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of robotics: Science and systems (RSS)*, 2023.

[10] Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi S. Jaakkola. 2023. DiffDock: Diffusion steps, twists, and turns for molecular docking. In *The eleventh international conference on learning representations*, 2023. Retrieved from https://openreview.net/forum?id=kKF8_K-mBbS

References II

[11] Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion models beat GANs on image synthesis. In *Advances in neural information processing systems*, 2021. Curran Associates, Inc., 8780–8794. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa-Paper.pdf

[12] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. 2024. Scaling rectified flow transformers for high-resolution image synthesis. Retrieved from https://arxiv.org/abs/2403.03206

[13] Zach Evans, CJ Carr, Josiah Taylor, Scott H. Hawley, and Jordi Pons. 2024. Fast timing-conditioned latent audio diffusion. Retrieved from https://arxiv.org/abs/2402.04825

[14] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. 2022. An image is worth one word: Personalizing text-to-image generation using textual inversion. https://doi.org/10.48550/ARXIV.2208.01618

[15] Ruiqi Gao*, Aleksander Holynski*, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul P. Srinivasan, Jonathan T. Barron, and Ben Poole*. 2024. CAT3D: Create anything in 3D with multi-view diffusion models. *arXiv* (2024).

[16] Agrim Gupta, Lijun Yu, Kihyuk Sohn, Xiuye Gu, Meera Hahn, Li Fei-Fei, Irfan Essa, Lu Jiang, and José Lezama. 2023. Photorealistic video generation with diffusion models. Retrieved from https://arxiv.org/abs/2312.06662

[17] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. 2022. Prompt-to-prompt image editing with cross attention control. (2022).

[18] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P. Kingma, Ben Poole, Mohammad Norouzi, David J. Fleet, and Tim Salimans. 2022. Imagen video: High definition video generation with diffusion models. Retrieved from https://arxiv.org/abs/2210.02303

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239* (2020).

[20] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. 2022. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research* 23, 47 (2022), 1–33. Retrieved from http://jmlr.org/papers/v23/21-0635.html

[21] Jonathan Ho and Tim Salimans. 2022. Classifier-free diffusion guidance. Retrieved from https://arxiv.org/abs/2207.12598

[22] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. 2022. Video diffusion models. In *Advances in neural information processing systems*, 2022. Curran Associates, Inc., 8633–8646. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2022/file/39235c56aef13fb05a6adc95eb9d8d66-Paper-Conference.pdf

[23] Qingqing Huang, Daniel S. Park, Tao Wang, Timo I. Denk, Andy Ly, Nanxin Chen, Zhengdong Zhang, Zhishuai Zhang, Jiahui Yu, Christian Frank, Jesse Engel, Quoc V. Le, William Chan, Zhifeng Chen, and Wei Han. 2023. Noise2Music: Text-conditioned music generation with diffusion models. Retrieved from https://arxiv.org/abs/2302.03917

References III

[24] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. 2021. OpenCLIP. https://doi.org/10.5281/zenodo.5143773

[25] Yang Jin, Zhicheng Sun, Ningyuan Li, Kun Xu, Kun Xu, Hao Jiang, Nan Zhuang, Quzhe Huang, Yang Song, Yadong Mu, and Zhouchen Lin. 2024. Pyramidal flow matching for efficient video generative modeling. Retrieved from https://arxiv.org/abs/2410.05954

[26] Zeqian Ju, Yuancheng Wang, Kai Shen, Xu Tan, Detai Xin, Dongchao Yang, Yanqing Liu, Yichong Leng, Kaitao Song, Siliang Tang, Zhizheng Wu, Tao Qin, Xiang-Yang Li, Wei Ye, Shikun Zhang, Jiang Bian, Lei He, Jinyu Li, and Sheng Zhao. 2024. NaturalSpeech 3: Zero-shot speech synthesis with factorized codec and diffusion models. Retrieved from https://arxiv.org/abs/2403.03100

[27] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. 2022. Elucidating the design space of diffusion-based generative models. In *Advances in neural information processing systems*, 2022. 26565–26577. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2022/file/a98846e9d9cc01cfb87eb694d946ce6b-Paper-Conference.pdf

[28] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. Retrieved from https://arxiv.org/abs/1312.6114

[29] Eran Levin and Ohad Fried. 2023. Differential diffusion: Giving each pixel its strength. Retrieved from https://arxiv.org/abs/2306.00950

[30] Chang Li, Ruoyu Wang, Lijuan Liu, Jun Du, Yixuan Sun, Zilu Guo, Zhenrong Zhang, and Yuan Jiang. 2024. QA-MDT: Quality-aware masked diffusion transformer for enhanced music generation. Retrieved from https://arxiv.org/abs/2405.15863

[31] Ming Li, Taojiannan Yang, Huafeng Kuang, Jie Wu, Zhaoning Wang, Xuefeng Xiao, and Chen Chen. 2024. ControlNet++: Improving conditional controls with efficient consistency feedback. In *European conference on computer vision (ECCV)*, 2024.

[32] Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. 2022. Diffusion-LM improves controllable text generation. In *Advances in neural information processing systems*, 2022. Curran Associates, Inc., 4328–4343. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2022/file/1be5bc25d50895ee656b8c2d9eb89d6a-Paper-Conference.pdf

[33] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. 2024. Common diffusion noise schedules and sample steps are flawed. In *2024 IEEE/CVF winter conference on applications of computer vision (WACV)*, 2024. 5392–5399. https://doi.org/10.1109/WACV57701.2024.00532

[34] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. 2023. Flow matching for generative modeling. In *The eleventh international conference on learning representations*, 2023. Retrieved from https://openreview.net/forum?id=PqvMRDCJT9t

[35] Xingchao Liu, Xiwen Zhang, Jianzhu Ma, Jian Peng, and qiang liu. 2024. InstaFlow: One step is enough for high-quality diffusion-based text-to-image generation. In *The twelfth international conference on learning representations*, 2024. Retrieved from https://openreview.net/forum?id=1k4yZbbDqX

References IV

[36] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. 2022. RePaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2022. 11461–11471.

[37] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. 2022. SDEdit: Guided image synthesis and editing with stochastic differential equations. Retrieved from https://arxiv.org/abs/2108.01073

[38] The Movie Gen team @ Meta. 2024. Movie gen: A cast of media foundation models. (2024). Retrieved from https://ai.meta.com/static-resource/movie-gen-research-paper

[39] Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav-Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. 2024. Dreamix: Video diffusion models are general video editors. Retrieved from https://openreview.net/forum?id=2vAhX71UCL

[40] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. 2022. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. In *Proceedings of the 39th international conference on machine learning* (*Proceedings of machine learning research*), 2022. PMLR, 16784–16804. Retrieved from https://proceedings.mlr.press/v162/nichol22a.html

[41] Alex Nichol and Prafulla Dhariwal. 2021. Improved denoising diffusion probabilistic models. Retrieved from https://arxiv.org/abs/2102.09672

[42] William Peebles and Saining Xie. 2023. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision (ICCV)*, 2023. 4195–4205.

[43] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. 2023. SDXL: Improving latent diffusion models for high-resolution image synthesis. Retrieved from https://arxiv.org/abs/2307.01952

[44] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2022. DreamFusion: Text-to-3D using 2D diffusion. *arXiv* (2022).

[45] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th international conference on machine learning* (*Proceedings of machine learning research*), 2021. PMLR, 8748–8763. Retrieved from https://proceedings.mlr.press/v139/radford21a.html

[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. Retrieved from http://jmlr.org/papers/v21/20-074.html

[47] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with CLIP latents. Retrieved from https://arxiv.org/abs/2204.06125

References V

[48] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *Proceedings of the 38th international conference on machine learning* (*Proceedings of machine learning research*), 2021. PMLR, 8821–8831. Retrieved from https://proceedings.mlr.press/v139/ramesh21a.html

[49] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2022. 10684–10695.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention – MICCAI 2015*, 2015. Springer International Publishing, Cham, 234–241.

[51] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. 2022. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. (2022).

[52] Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. 2022. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH 2022 conference proceedings* (*SIGGRAPH '22*), 2022. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3528233.3530757

[53] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. 2022. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in neural information processing systems*, 2022. Curran Associates, Inc., 36479–36494. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2022/file/ec795aeadae0b7d230fa35cbaf04c041-Paper-Conference.pdf

[54] Tim Salimans and Jonathan Ho. 2022. Progressive distillation for fast sampling of diffusion models. In *International conference on learning representations*, 2022. Retrieved from https://openreview.net/forum?id=TIdIXIpzhoI

[55] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd international conference on machine learning* (*Proceedings of machine learning research*), 2015. PMLR, Lille, France, 2256–2265. Retrieved from https://proceedings.mlr.press/v37/sohl-dickstein15.html

[56] Jiaming Song, Chenlin Meng, and Stefano Ermon. 2020. Denoising diffusion implicit models. *arXiv:2010.02502* (2020). Retrieved from https://arxiv.org/abs/2010.02502

[57] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. 2023. Consistency models. *arXiv preprint arXiv:2303.01469* (2023).

[58] Yang Song and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. In *Proceedings of the 33rd international conference on neural information processing systems*. Curran Associates Inc., Red Hook, NY, USA.

[59] Yang Song and Stefano Ermon. 2020. Improved techniques for training score-based generative models. In *Proceedings of the 34th international conference on neural information processing systems* (*NIPS '20*), 2020. Curran Associates Inc., Red Hook, NY, USA.

References VI

[60] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. Score-based generative modeling through stochastic differential equations. In *International conference on learning representations*, 2021. Retrieved from https://openreview.net/forum?id=PxTIG12RRHS

[61] Gemma Team. 2024. Gemma 2: Improving open language models at a practical size. Retrieved from https://arxiv.org/abs/2408.00118

[62] Imagen Team. 2024. Imagen 3. Retrieved from https://arxiv.org/abs/2408.07009

[63] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. 2024. Diffusion models are real-time game engines. Retrieved from https://arxiv.org/abs/2408.14837

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, 2017. Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[65] Fu-Yun Wang, Ling Yang, Zhaoyang Huang, Mengdi Wang, and Hongsheng Li. 2024. Rectified diffusion: Straightness is not your need in rectified flow. Retrieved from https://arxiv.org/abs/2410.07303

[66] Lilian Weng. 2021. What are diffusion models? *lilianweng.github.io* (July 2021). Retrieved from https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[67] Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Stan Weixian Lei, Yuchao Gu, Yufei Shi, Wynne Hsu, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. 2023. Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation. In *Proceedings of the IEEE/CVF international conference on computer vision (ICCV)*, 2023. 7623–7633.

[68] Yuxin Wu and Kaiming He. 2018. Group normalization. In *Computer vision – ECCV 2018: 15th european conference, munich, germany, september 8-14, 2018, proceedings, part XIII*, 2018. Springer-Verlag, Berlin, Heidelberg, 3–19. https://doi.org/10.1007/978-3-030-01261-8_1

[69] Shitao Xiao, Yueze Wang, Junjie Zhou, Huaying Yuan, Xingrun Xing, Ruiran Yan, Shuting Wang, Tiejun Huang, and Zheng Liu. 2024. OmniGen: Unified image generation. Retrieved from https://arxiv.org/abs/2409.11340

[70] Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, and Song Han. 2024. SANA: Efficient high-resolution image synthesis with linear diffusion transformers. Retrieved from https://arxiv.org/abs/2410.10629

[71] Yifeng Xu, Zhenliang He, Shiguang Shan, and Xilin Chen. 2024. CtrLoRA: An extensible and efficient framework for controllable image generation. *arXiv preprint arXiv:2410.09400* (2024).

[72] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, Da Yin, Xiaotao Gu, Yuxuan Zhang, Weihan Wang, Yean Cheng, Ting Liu, Bin Xu, Yuxiao Dong, and Jie Tang. 2024. CogVideoX: Text-to-video diffusion models with an expert transformer. Retrieved from https://arxiv.org/abs/2408.06072

References VII

[73] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In *2023 IEEE/CVF international conference on computer vision (ICCV)*, 2023. 3813–3824. https://doi.org/10.1109/ICCV51070.2023.00355

[74] Shihao Zhao, Dongdong Chen, Yen-Chun Chen, Jianmin Bao, Shaozhe Hao, Lu Yuan, and Kwan-Yee K. Wong. 2023. Uni-ControlNet: All-in-one control to text-to-image diffusion models. *Advances in Neural Information Processing Systems* (2023).

[75] Wangbo Zhao, Yizeng Han, Jiasheng Tang, Kai Wang, Yibing Song, Gao Huang, Fan Wang, and Yang You. 2024. Dynamic diffusion transformer. Retrieved from https://arxiv.org/abs/2410.03456

[76] Zhenyu Zhou, Defang Chen, Can Wang, Chun Chen, and Siwei Lyu. 2024. Simple and fast distillation of diffusion models. Retrieved from https://arxiv.org/abs/2409.19681