UNIVERSIDAD DE CHILE

Inteligencia Artificial Generativa

Let's talk about hype stuff

Valentin Barriere // Clemente Henriquez

Universidad de Chile – DCC

Diplomado de Postítulo en Inteligencia Artificial, Primavera 2025

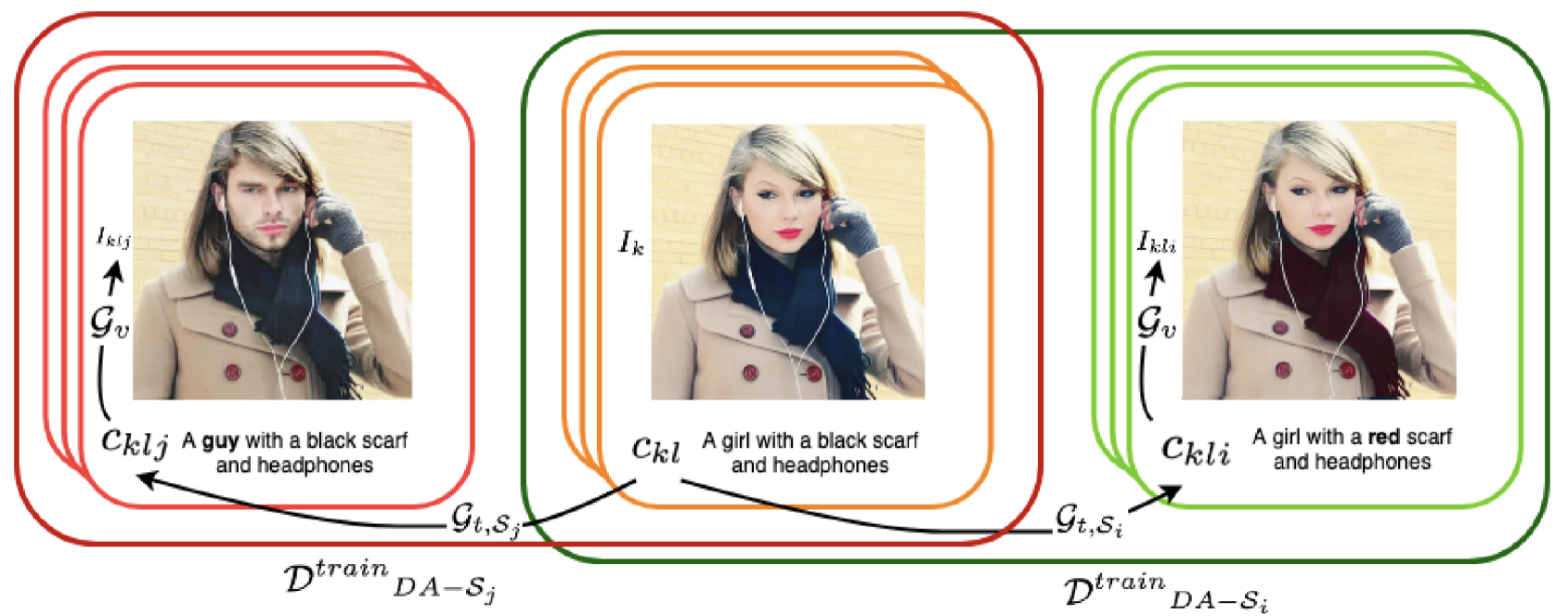
Introduction and Generative Vision Models

Outline : Generative Models

Why Do We Need Generative Models?

AI isn't just about decision-making — it's also about creation.

- Generative models allow machines to **create** realistic data: images, text, audio, etc...: here an example with StyleGanv2
- They enable:
  - Data augmentation
  - Creative content generation
  - Simulation and stress-testing of AI systems using counterfactuals



Why Do We Need Generative Models?

- **Critical insight:** A generative model can fool a discriminative one.
- Example: GANs generate fake images that classifiers confidently mislabel — exposing vulnerabilities in decision systems.
- This interplay drives progress in robustness, security, and evaluation for **building trustworthy AI**.

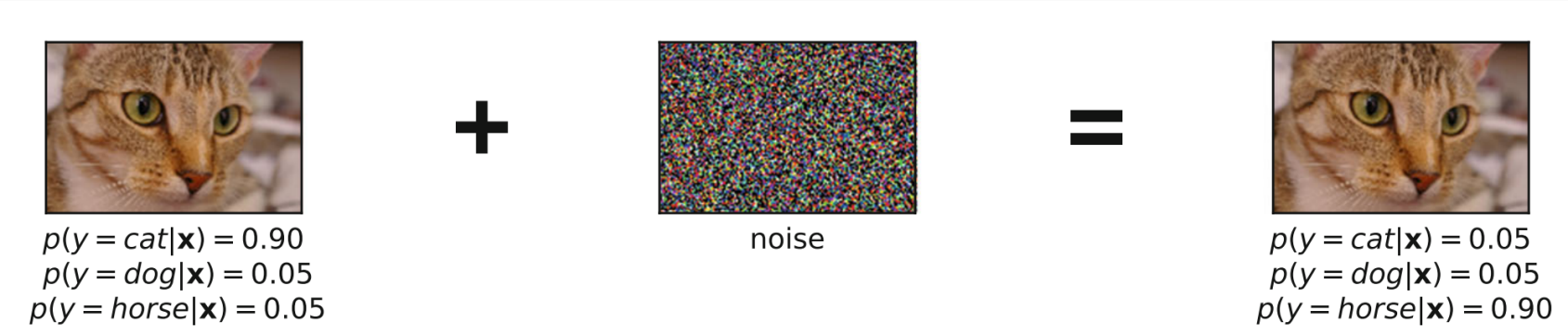


Figure 2: Adding noise results in a shift of predicted label.

Generative vs. Discriminative Models

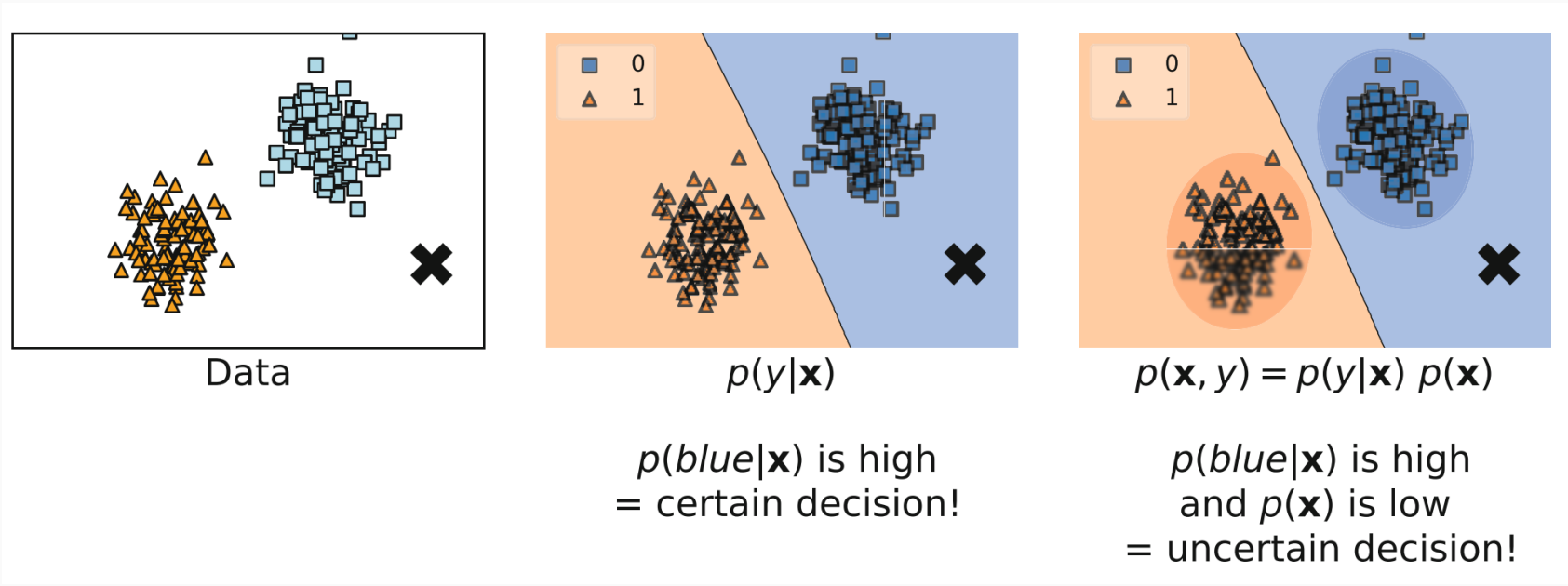


Figure 3: Example of data and two approaches to decision-making: a discriminative approach and a generative approach.

- **Discriminative**: Learn $p(y|x)$: "Given input, what label?"
- **Generative**: Learn $p(x, y)$ or $p(x)$: "How is data generated?"

Generative vs. Discriminative Models

Discriminative Models	Generative Models
Focus on decision boundary	Model full data distribution
Examples: Logistic Regression, CNN classifiers	Examples: Naive Bayes, VAEs, GANs, AR models
High accuracy for classification	Can generate samples, handle missing data, detect anomalies
Cannot generate new data	Often more complex to train

Key Insight:

Generative models ask: *"What does the world look like?"*

Discriminative models ask: *"What is this specific thing?"*

Generative vs. Discriminative Models

Discriminative Models	Generative Models
Focus on decision boundary	Model full data distribution
Examples: Logistic Regression, CNN classifiers	Examples: Naive Bayes, VAEs, GANs, AR models
High accuracy for classification	Can generate samples, handle missing data, detect anomalies
Cannot generate new data	Often more complex to train

Key Insight:

Generative models ask: *"What does the world look like?"*

Discriminative models ask: *"What is this specific thing?"*

Generative models are more ambitious — they model the full process of data creation.

Applications of Deep Generative Models

Where Are They Used? Real-World Impact

Deep generative models are transforming multiple domains:

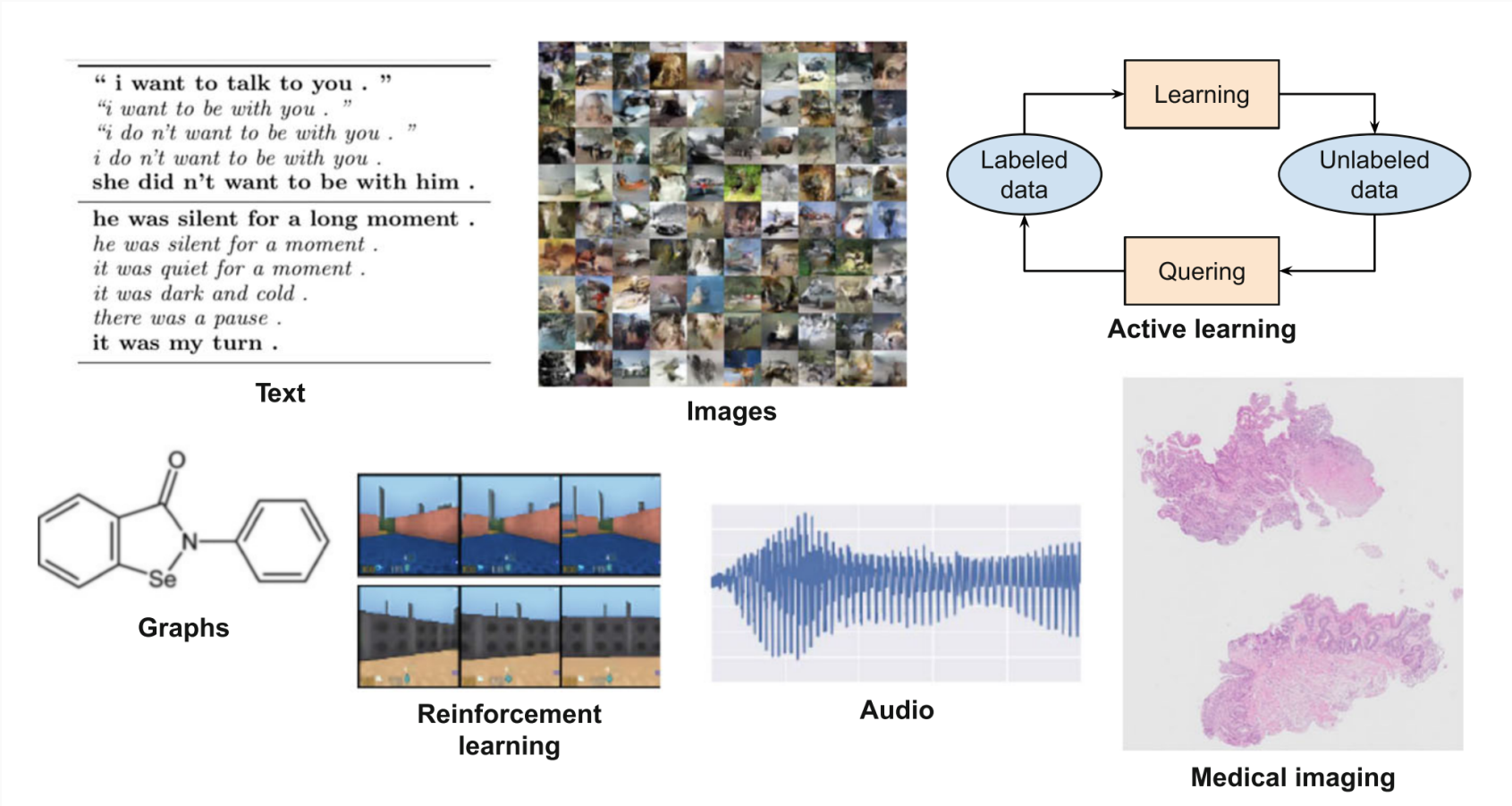


Figure 4: Drug discovery, DeepFake, Counterfactuals, Conversational agents, Music generation.

Applications of Deep Generative Models

Where Are They Used? Real-World Impact

Deep generative models are transforming multiple domains:

- **Image**: Super-resolution, inpainting, style transfer (e.g., DALL·E, Stable Diffusion)
- **Text**: Story generation, dialogue systems, code synthesis (e.g., GPT)
- **Audio**: Voice cloning, music generation (e.g., Jukebox)
- **Video**: Prediction, animation, deepfakes
- **Scientific Discovery**:
  - Generate novel drug-like molecules (graph VAEs, diffusion on graphs)
  - Simulate physical systems
- **Anomaly Detection**: Learn normal data, flag outliers (e.g., in medical imaging)

Applications of Deep Generative Models

Where Are They Used? Real-World Impact

Deep generative models are transforming multiple domains:

- **Image**: Super-resolution, inpainting, style transfer (e.g., DALL·E, Stable Diffusion)
- **Text**: Story generation, dialogue systems, code synthesis (e.g., GPT)
- **Audio**: Voice cloning, music generation (e.g., Jukebox)
- **Video**: Prediction, animation, deepfakes
- **Scientific Discovery**:
  - Generate novel drug-like molecules (graph VAEs, diffusion on graphs)
  - Simulate physical systems
- **Anomaly Detection**: Learn normal data, flag outliers (e.g., in medical imaging)

Generative AI is not just art — it's a tool for science, medicine, and engineering.

Autoregressive Models (ARMs)

Building Data Step by Step

- Core idea: Factorize joint distribution sequentially:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

- Predict each element conditioned on previous ones — strictly sequential.

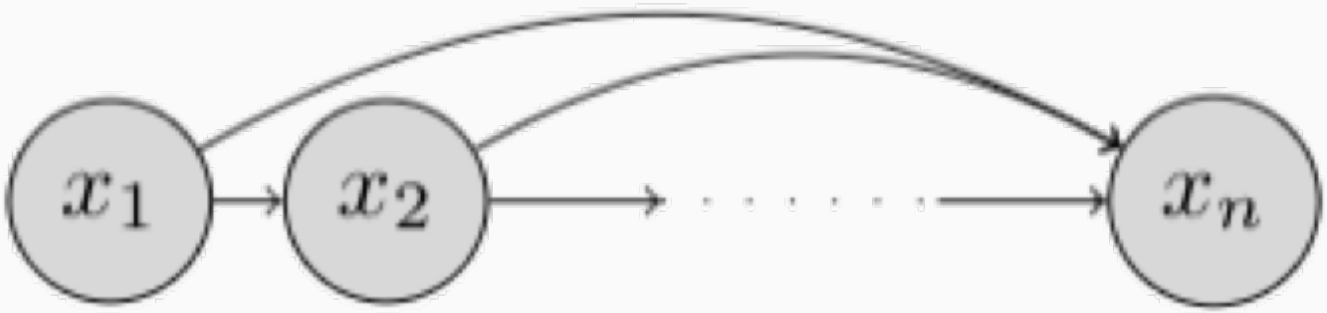


Figure 4: Chain rule factorization expressed graphically as a Bayesian network.

Autoregressive Models (ARMs)

Building Data Step by Step

- Core idea: Factorize joint distribution sequentially:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

- Predict each element conditioned on previous ones — strictly sequential.

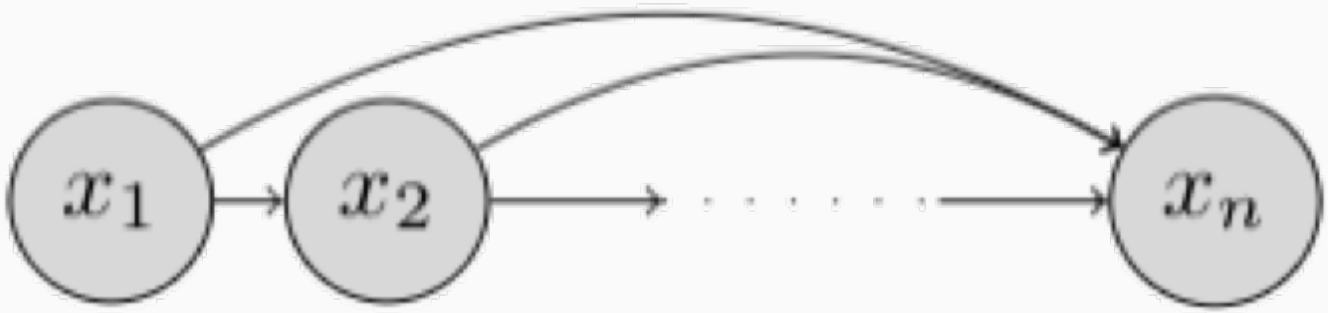


Figure 4: Chain rule factorization expressed graphically as a Bayesian network.

Basically, you use the past values to generate the future ones, and so on.

Autoregressive Models (ARMs)

Building Data Step by Step

- Core idea: Factorize joint distribution sequentially:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i \mid x_{<i})$$

- Predict each element conditioned on previous ones — strictly sequential.
- **Examples**:
  - Causal Convolutions on images (PixelCNN, PixelRNN) or audio (Wavenet)
  - Causal Transformers Attention (GPT)
  - More info here

Pros:

- Exact likelihood
- Lossless compression
- Stable training

Cons:

- Slow sampling (no parallelization)
- Quality depends on generation order

Latent Variable Models: VAE, Diffusion, GANs

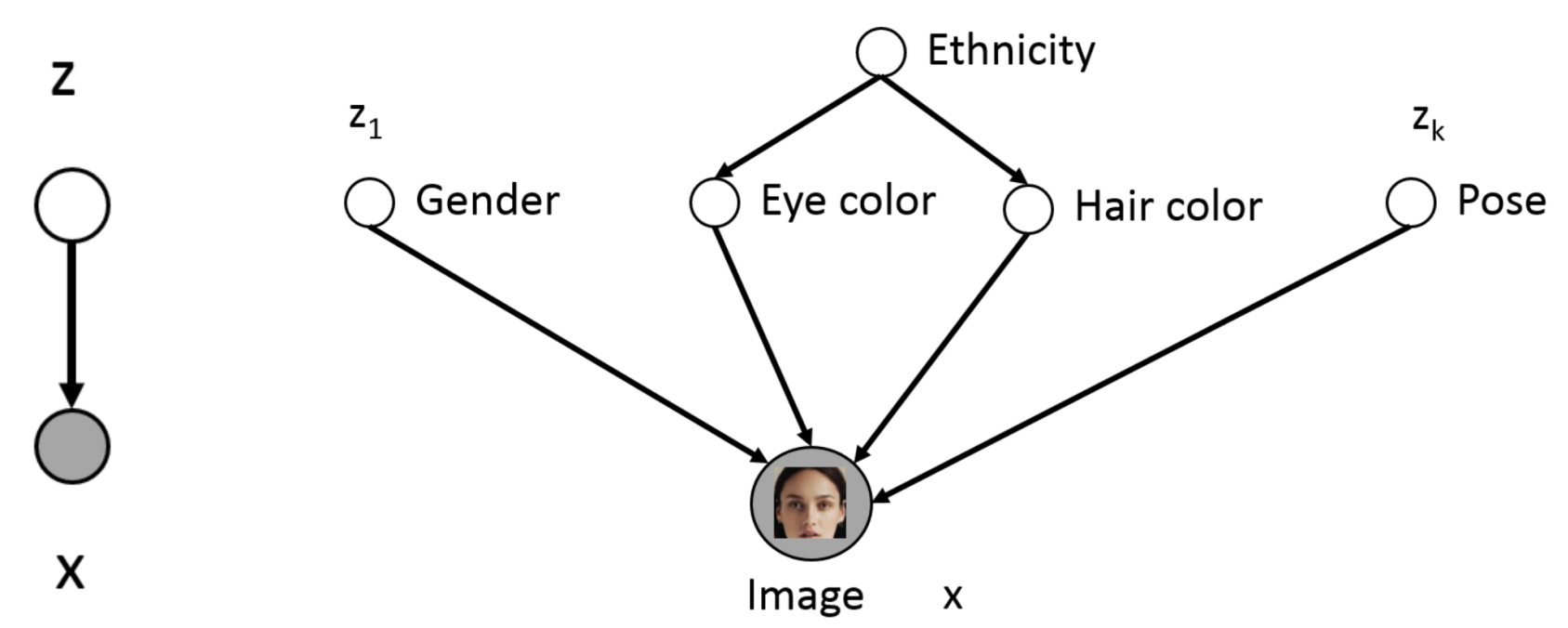
These models assume data x is generated from latent variables z:

$$x \sim p(x|z), \quad z \sim p(z)$$

Motivation

- Lots of variability in images x due to gender, eye color, hair color, pose, etc. However, unless images are annotated, these factors of variation are not explicitly available (latent).
- **Idea**: explicitly model these factors using latent variables z

Latent Variable Models: VAE, Diffusion, GANs



- Only shaded variables $\mathbf{x}$ are observed in the data (pixel values)
  Latent variables $\mathbf{z}$ correspond to high level features
  - If $\mathbf{z}$ chosen properly, $p(\mathbf{x}|\mathbf{z})$ could be much simpler than $p(\mathbf{x})$
  - If we had trained this model, then we could identify features via $p(\mathbf{z}|\mathbf{x})$, e.g., $p(EyeColor = Blue|\mathbf{x})$

Latent Variable Models: VAE, Diffusion, GANs

VAE

Variational Autoencoder: learns a probabilistic encoder and decoder. Maximizes ELBO. Good for structured latent space.

Diffusion

Gradually adds noise, then reverses the process. State-of-the-art quality via score matching.

GAN

Adversarial training: generator vs. discriminator. No explicit density. Fast sampling, sharp outputs.

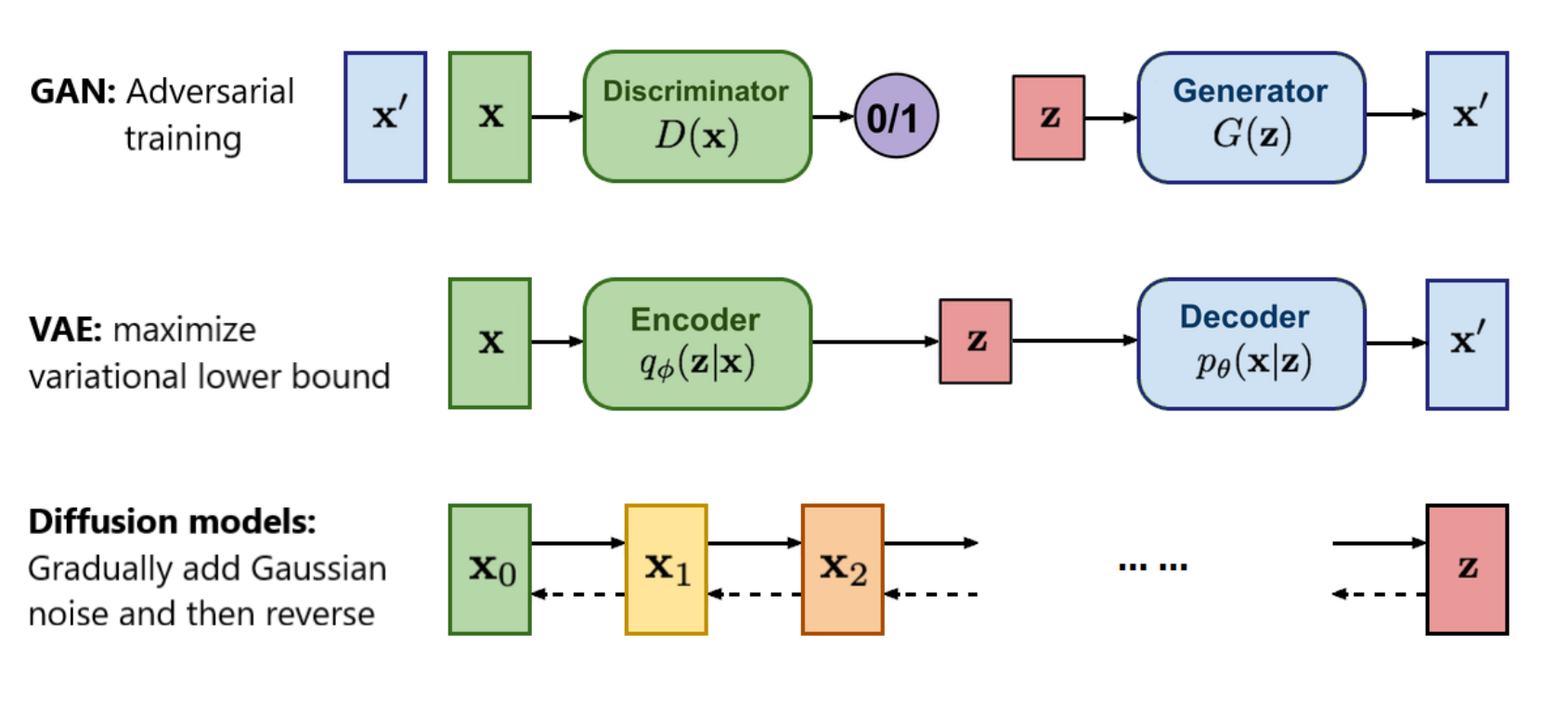


Figure 4: Probabilistic (VAE), Iterative (Diffusion), Adversarial (GAN).

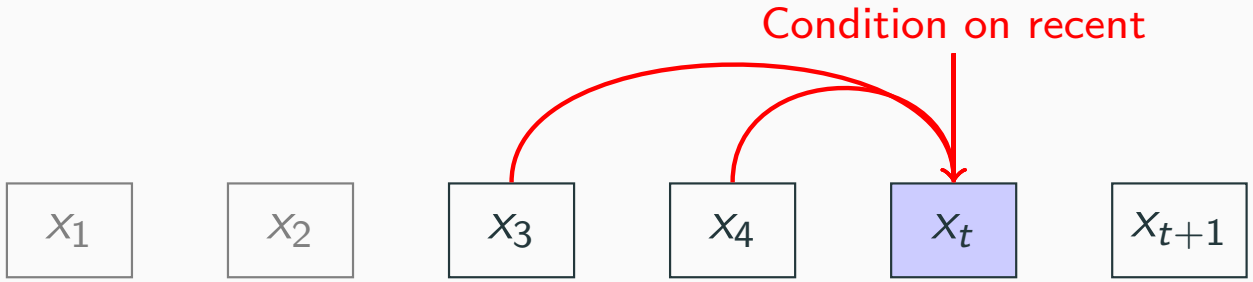
Outline : Autoregressive Models

Autoregressive Models: Finite Memory and Context

An autoregressive (AR) model generates data one step at a time:

$$p(x) = \prod_{t=1}^{T} p(x_t \mid x_{<t}), \quad x_{<t} = (x_1, \ldots, x_{t-1})$$

But in practice, models have **finite memory** — they can't condition on the entire past. Distant past is ignored with a context window.



Distant past ignored: $p(x_t \mid x_{<t}) \approx p(x_t \mid x_{t-k}, \ldots, x_{t-1})$

Context window: Only the last k tokens are used in memory:

$$p(x) = p(x_1)p(x_2 \mid x_1) \prod_{d=3}^{D} p(x_d \mid x_{d-1}, x_{d-2})$$

Challenge: How to model **long-range dependencies** efficiently?

Long-Range Memory Through RNNs

$$p(x_d \mid x_{<d}) = p(x_d \mid h_d) = p\big(x_d \mid \mathrm{RNN}(x_{d-1}, h_{d-1})\big)$$

RNNs maintain a **hidden state** h_t that summarizes the past:

$$h_t = f_\theta(h_{t-1}, x_{t-1}), \quad \hat{x}_t = g_\theta(h_t)$$

h_d is a hidden context that acts as a memory that allows learning long-range dependencies

Pros:

- Naturally autoregressive and causal
- Can, in theory, remember entire history

Cons:

- Vanishing gradients $\rightarrow$ hard to learn long-term dependencies
- Sequential: cannot parallelize training

Comparison Finite Memory vs RNN

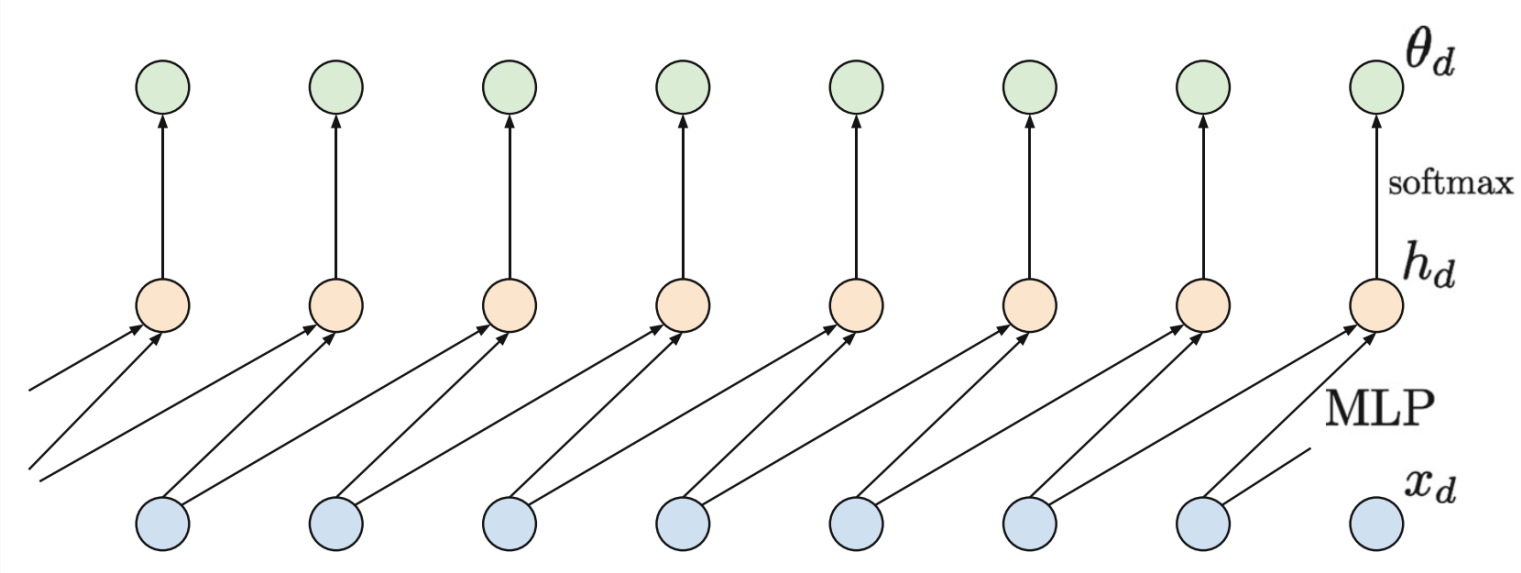


Figure 5: MLP depending on 2 last inputs. Proba θ_d is not dependent on x_d

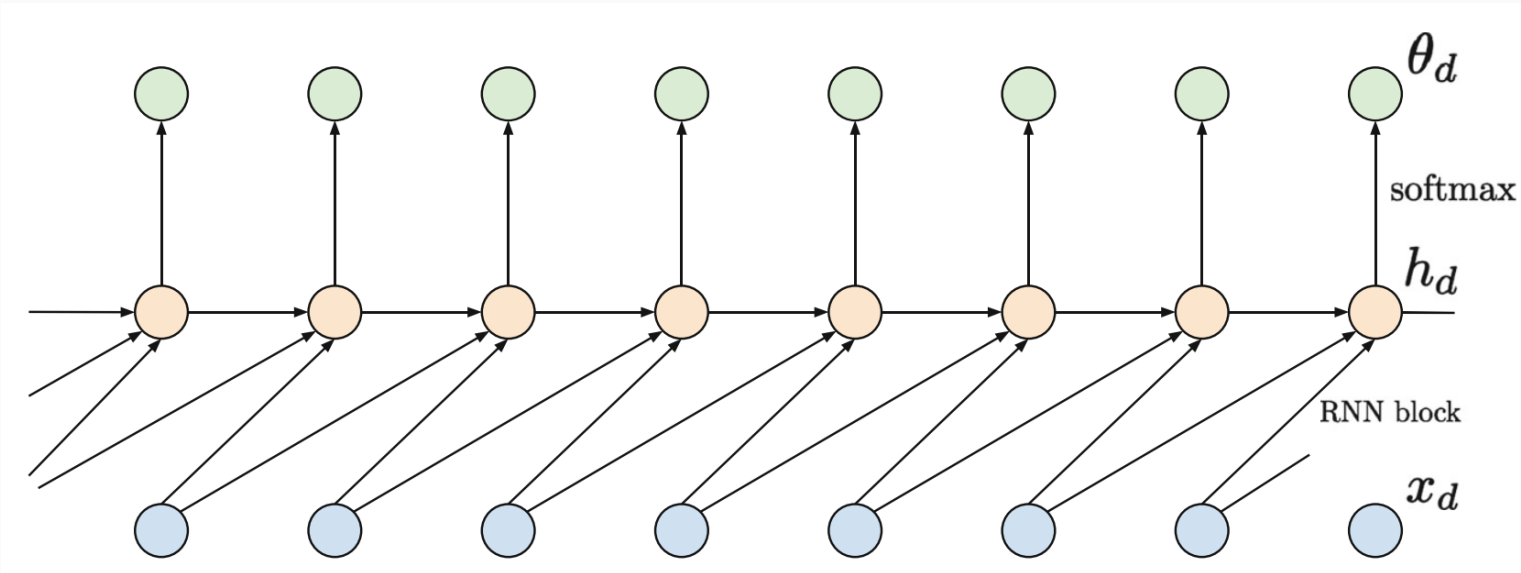


Figure 6: Same with RNN, adding additional dependency between h_d

Causal Convolutional Networks: Parallel Autoregression

Causal CNNs apply convolutions **only on past inputs**, preserving autoregression: $h_t = \sigma\left(\sum_{k=1}^{K} W_k x_{t-k} + b\right)$

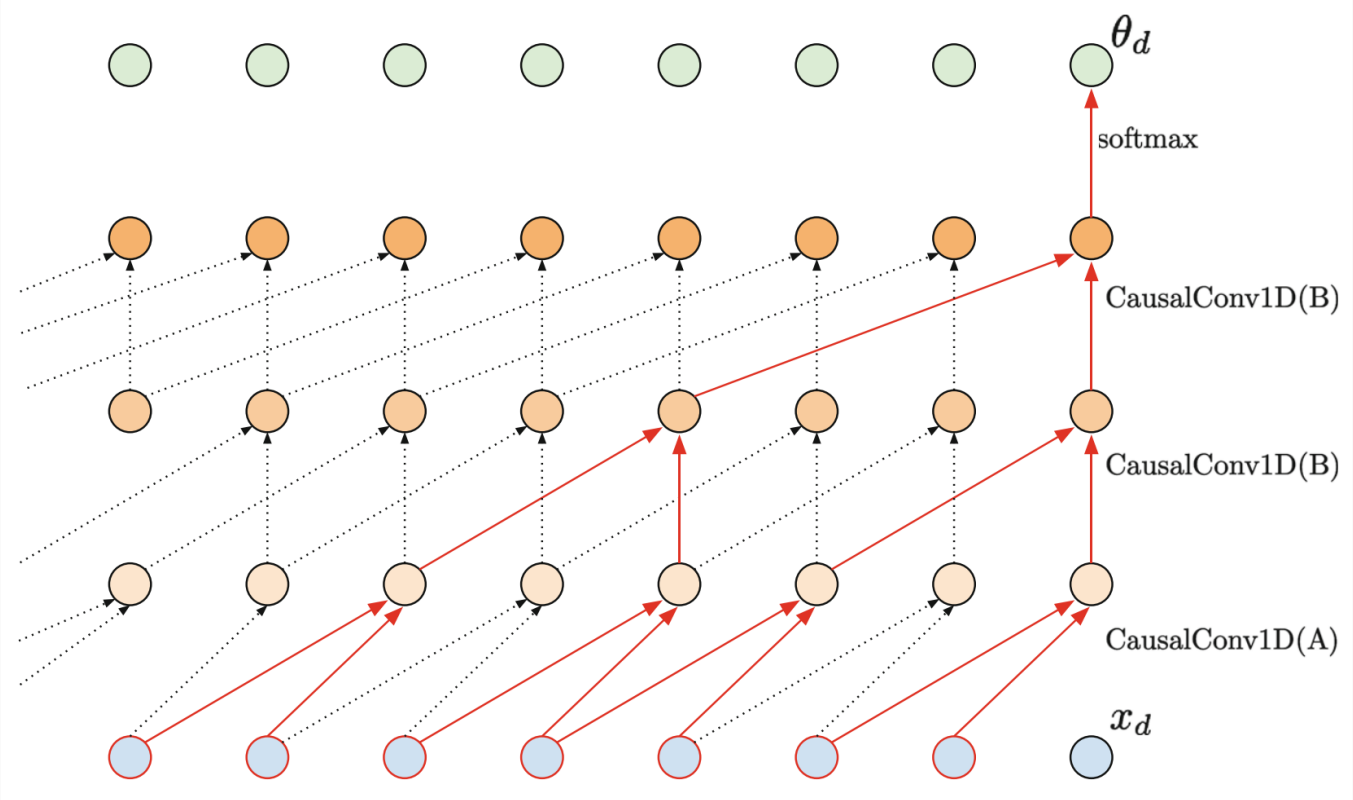
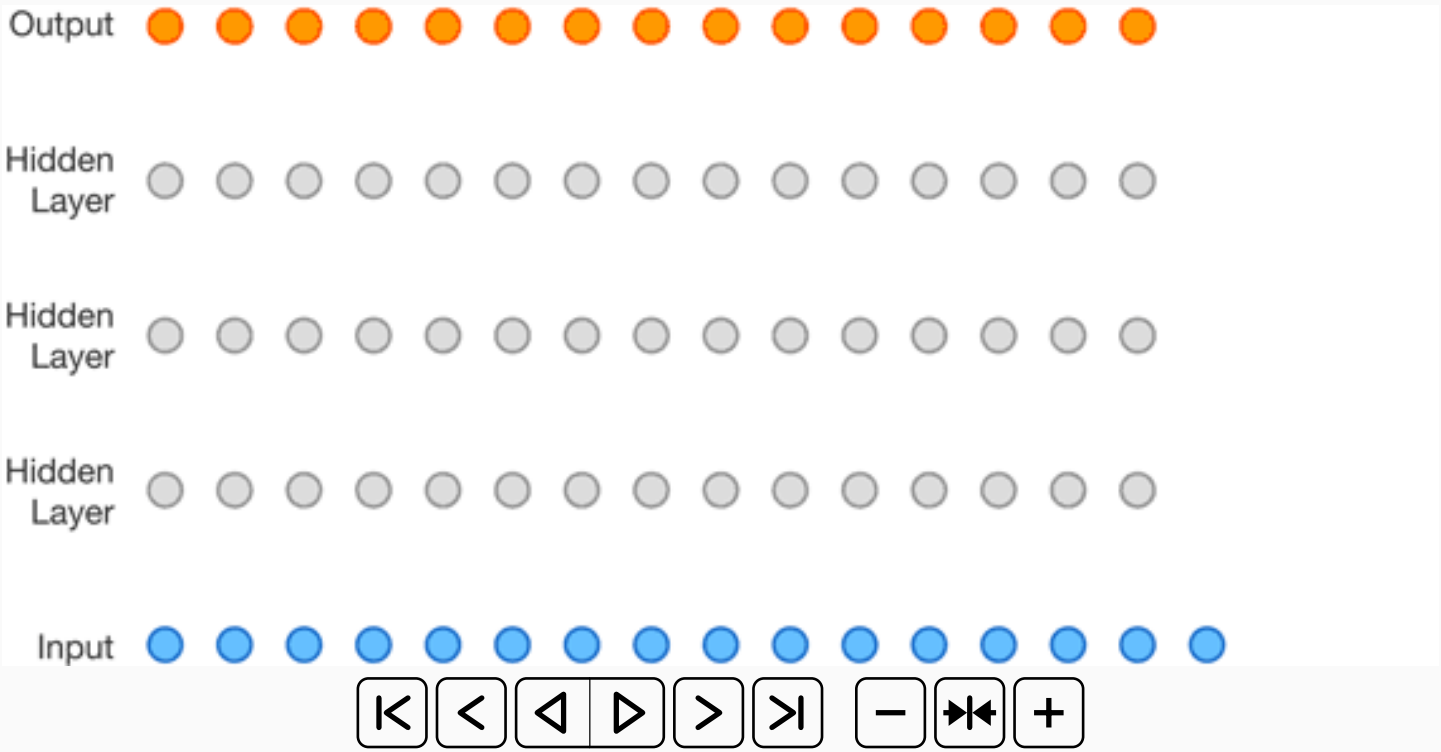


Figure 7: Causal convolutions with kernel size is 2, but by applying dilation in higher layers, a much larger input could be processed (red edges); thus, a larger memory is utilized.

Causal Convolutional Networks: Parallel Autoregression

Advantages of Stacked causal convolutions

- **Dilated convolutions** $\rightarrow$ exponentially growing receptive field
- **Parallelizable** over time $\rightarrow$ fast training
- **Causal** $\rightarrow$ no future leakage



Gif del [Wavenet]

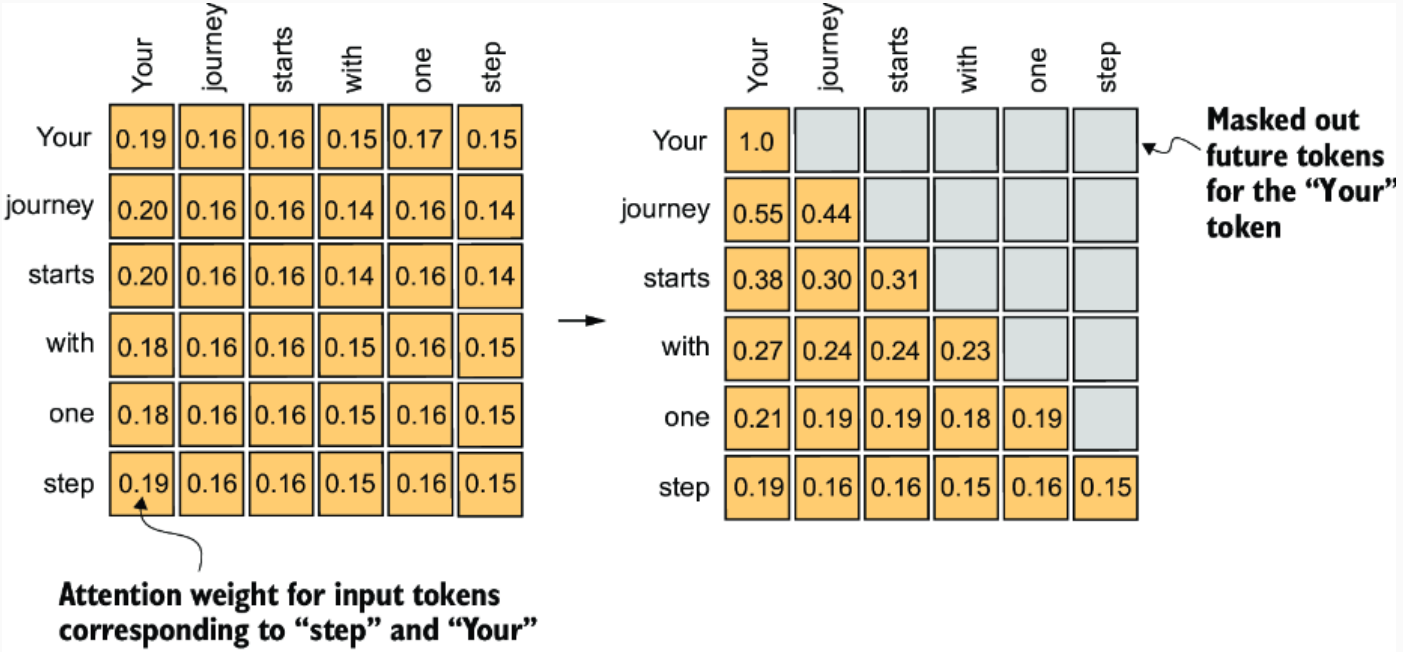
Causal Attention: Transformers for Autoregressive Modeling

Transformers use **self-attention**:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

But to be autoregressive, we apply a **causal mask**:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \underbrace{M}_{\text{mask}}\right), \quad M_{ij} = \begin{cases} 0 & i \geq j \\ -\infty & i < j \end{cases}$$



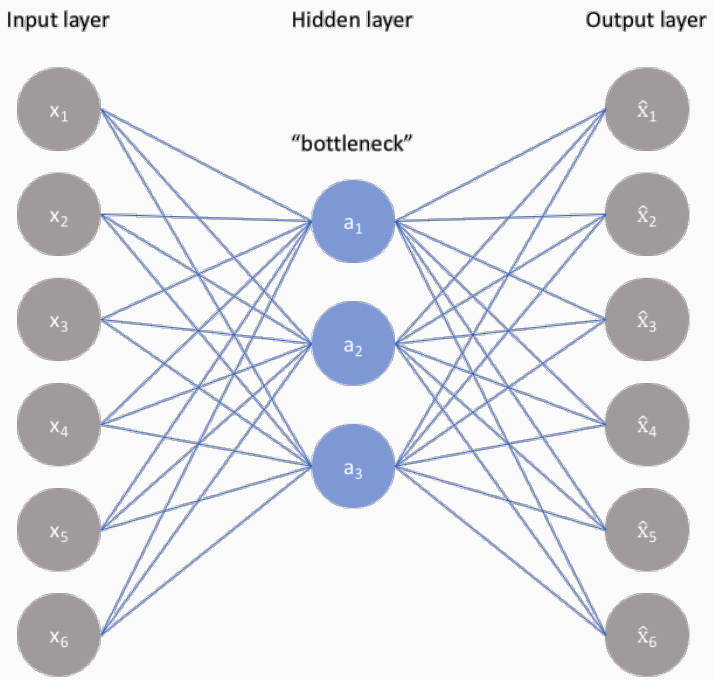
- **Global context**: attends to all previous tokens
- **Parallel training**: unlike RNNs
- **Flexible weighting**: learns what to remember

Outline : Variational Auto-Encoders

Autoencoder: Principle

Principle

The principle of an autoencoder is to find a lower-dimensional distributed representation **a** that allows reconstruction of the original input **X**, thus containing as much information.



Autoencoder: Training

Unsupervised Training

The autoencoder is trained in an unsupervised manner using a loss function based on the reconstruction $\hat{\mathbf{X}}$ of the original input $\mathbf{X}$: $\ell(\mathbf{X}) = ||\mathbf{X} - \hat{\mathbf{X}}||_2$.

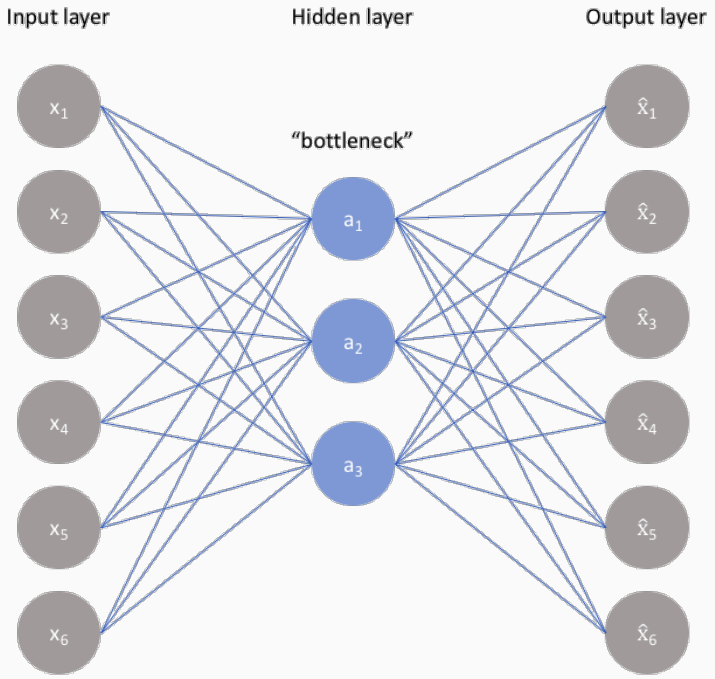
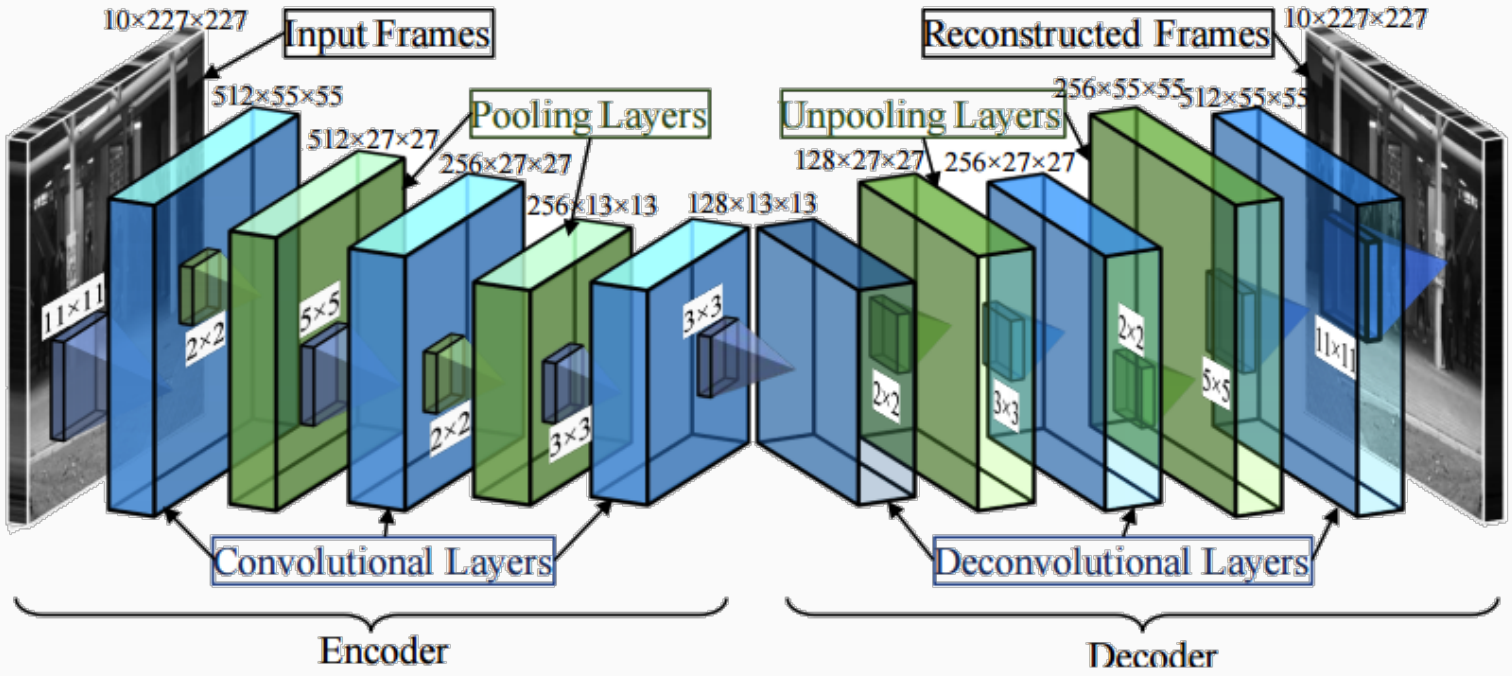


Figure 8: Example of a single-layer AE; any number of layers can be used

Autoencoder: Types of Encoders

Any Type of Encoder

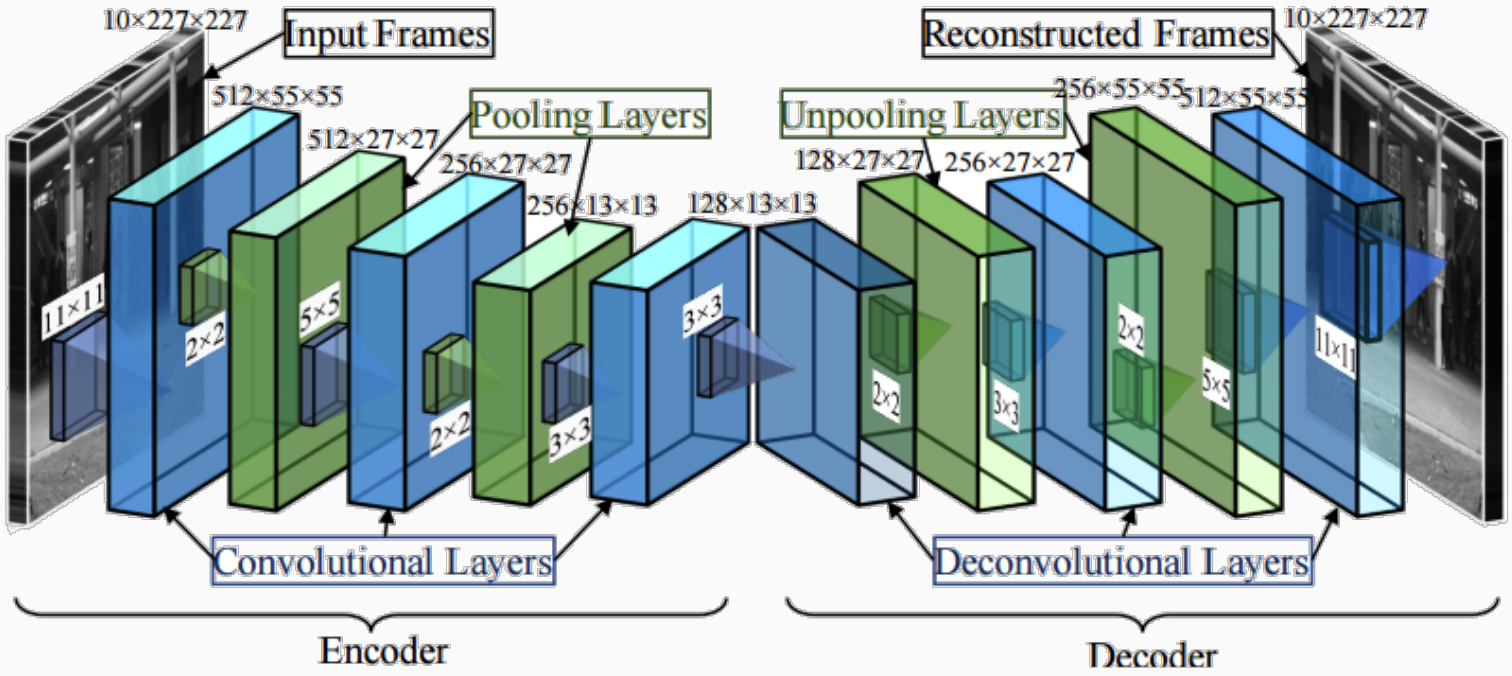
The encoder and decoder can be more than just multilayer perceptrons—they can also be CNNs or RNNs.



Autoencoder: Types of Encoders

Any Type of Encoder

The encoder and decoder can be more than just multilayer perceptrons—they can also be CNNs or RNNs.

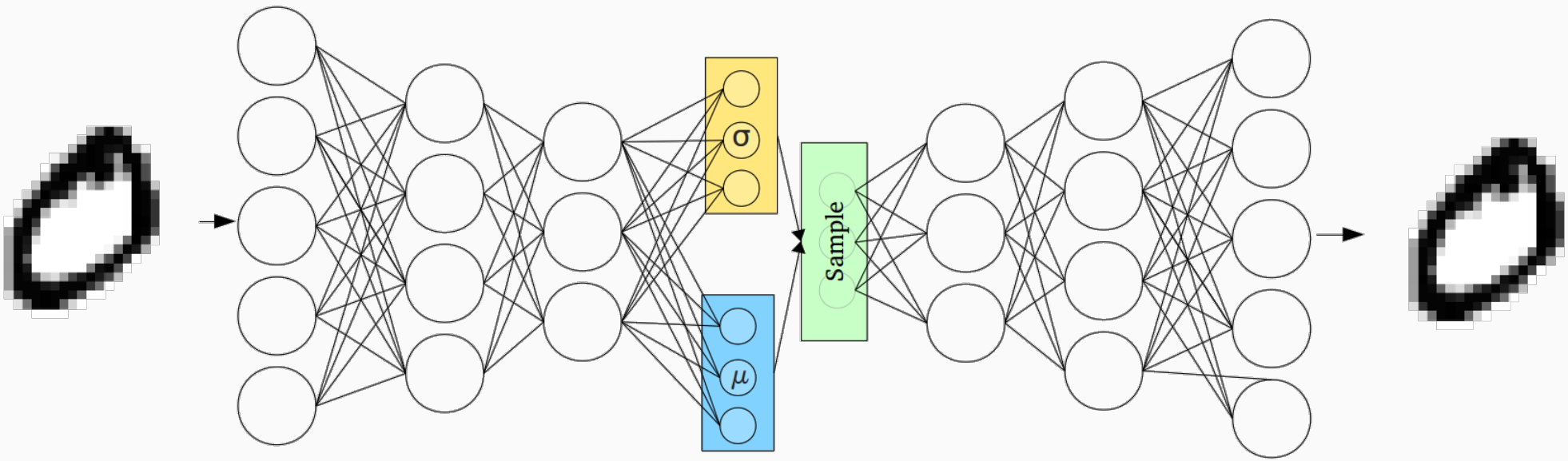


Problem: AE are very good at compression but bad a generation

Variational Autoencoder

Principle

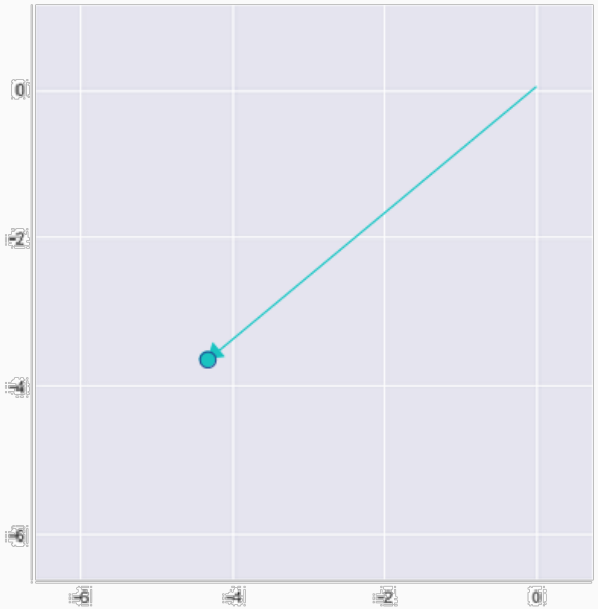
The VAE no longer encodes data as a single point in space, but as a Gaussian distribution with a mean μ and standard deviation σ.



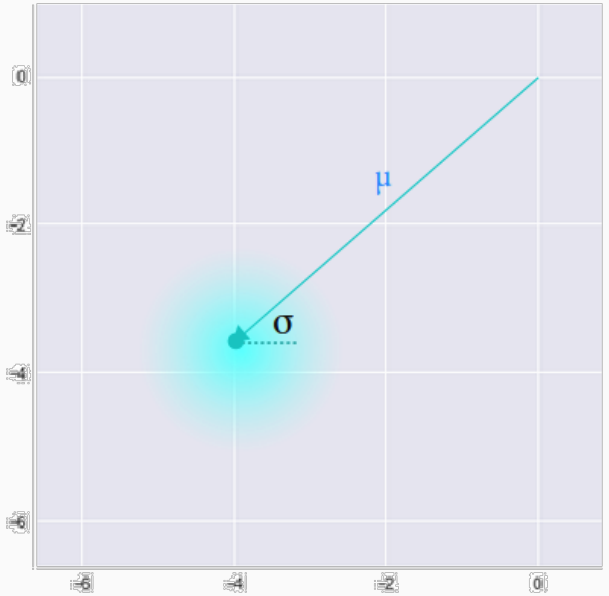
Variational Autoencoder

Advantage

This allows the latent space to be "smoothed" between different points, enabling meaningful transitions from one point to another and generating coherent new data.



Standard Autoencoder
(direct encoding coordinates)

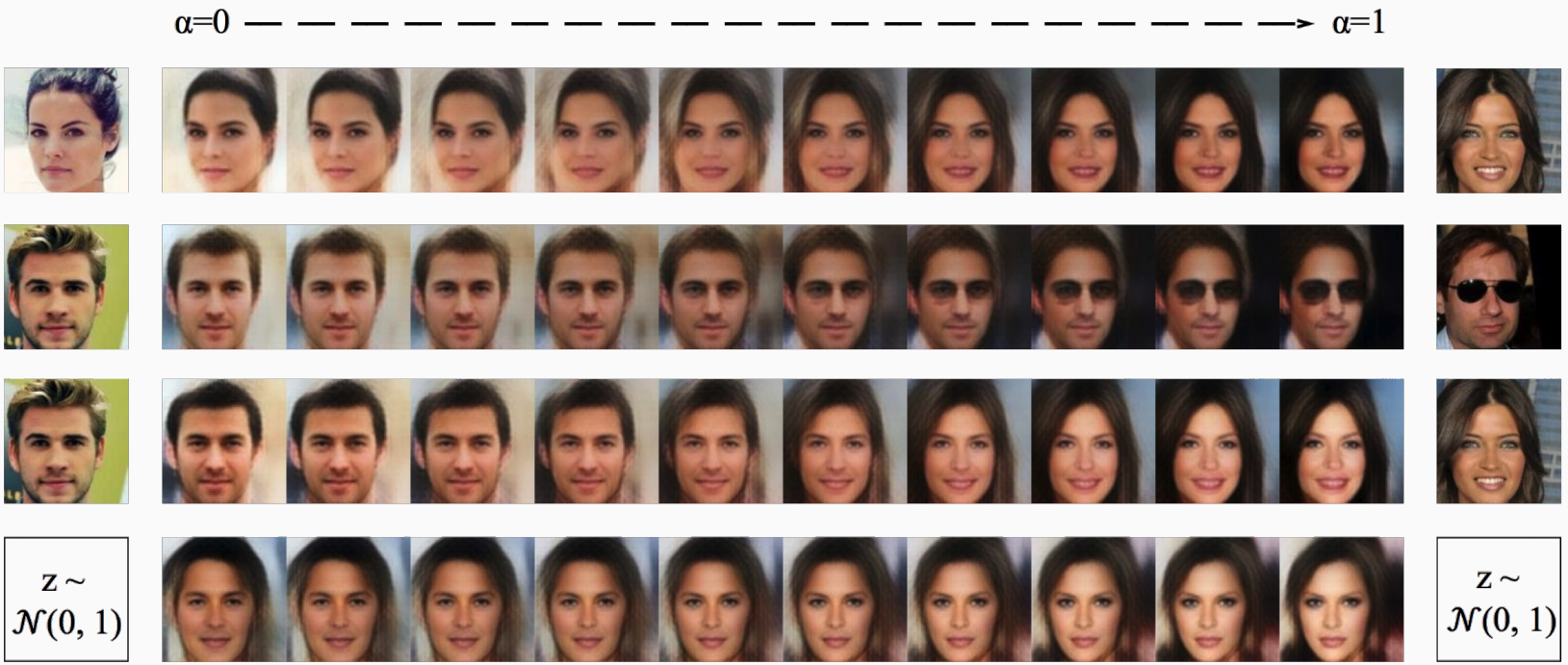


Variational Autoencoder
(μ and σ initialize a probability distribution)

Variational Autoencoder: Example Application

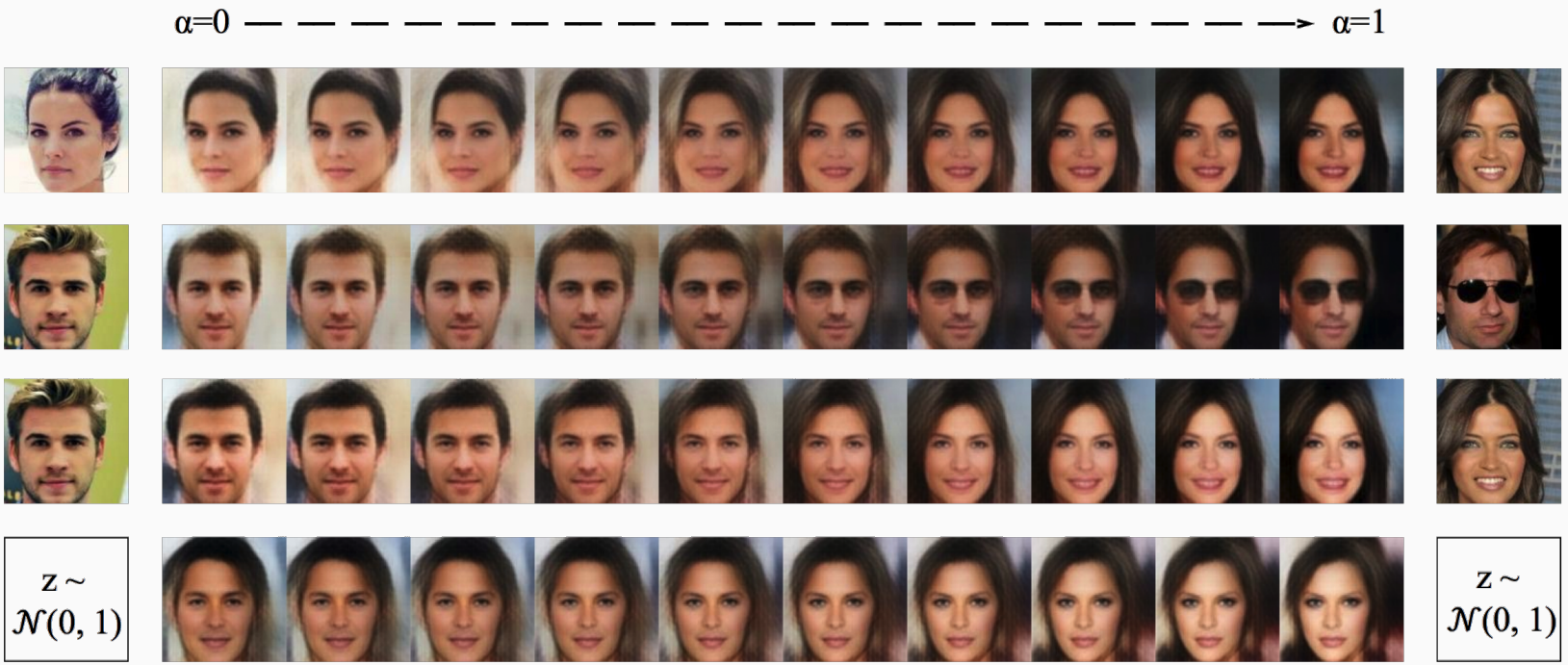
Variational Autoencoder: Example Application

Smooth transition from one image to another:



Variational Autoencoder: Example Application

Smooth transition from one image to another:



Great for tasks where a well-structured latent space is important such as: data exploration, anomaly detection, or generating diverse data samples.

The Problem: Intractable Posterior

We want to model data x using latent variables z:

$$p_\theta(x) = \int p_\theta(x|z)p(z)\,dz$$

But to learn good representations, we also need the posterior:

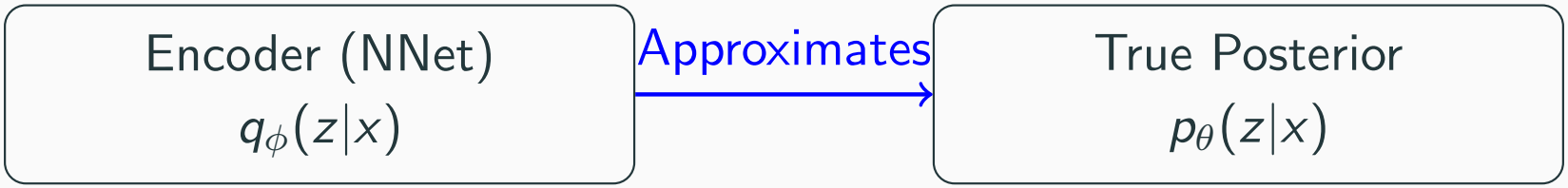
$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)}$$

Problem: $p_\theta(x)$ (the marginal likelihood) is intractable

- It requires integrating over all possible z
- For high-dimensional z and complex decoders, this is impossible.
- We need an approximation

The Solution: Variational Inference with $q_\phi(z|x)$

We introduce a **variational distribution** $q_\phi(z|x)$ — an approximation to the true posterior.



$$\text{Outputs } \mu, \sigma \rightarrow z \sim \mathcal{N}(\mu, \sigma^2)$$

Our goal: make $q_\phi(z|x)$ close to $p_\theta(z|x)$ by minimizing:

$$\text{KL}\left(q_\phi(z|x) \,\middle\|\, p_\theta(z|x)\right)$$

But this KL still depends on $p_\theta(x)$... So we rewrite it:

$$\log p_\theta(x) =$$

$$\underbrace{\mathbb{E}_q\left[\log p_\theta(x|z)\right] - \text{KL}\left(q_\phi(z|x) \,\|\, p(z)\right)}_{\text{ELBO: what we maximize}} + \underbrace{\text{KL}\left(q_\phi(z|x) \,\|\, p_\theta(z|x)\right)}_{\text{always}>0}$$

Knowing $p_\theta(x) < 1 \Rightarrow \log p_\theta(x) < 0$, ELBO is the objective.

The ELBO: A Computable Objective

The **Evidence Lower BOund (ELBO)** is:

$$\mathcal{L}(\theta, \phi; x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{Reconstruction}} - \underbrace{\mathrm{KL}\left(q_\phi(z|x) \,\middle\|\, p(z)\right)}_{\text{Latent Regularization}}$$

Reconstruction Term

Maximizing this ensures the decoder can recreate x from z.

KL Divergence Term

Forces $q_\phi(z|x)$ to match the prior $p(z) = \mathcal{N}(0, 1)$.

Key insight: Maximizing ELBO:

- Improves data fit
- Makes $q_\phi(z|x)$ close to $p_\theta(z|x)$
- Regularizes the latent space for meaningful sampling

Why Regularize $q_\phi(z|x)$ to $\mathcal{N}(0,1)$?

One would say: "The encoder must output $\mu = 0$, $\sigma = 1$"
Nope: It can output any μ, σ — but pays a **KL cost** if they deviate.

Why do this?

- Ensures the **decoder** learns to interpret $z \sim \mathcal{N}(0,1)$
- Enables **sampling**: generate new data by $z \sim \mathcal{N}(0,1)$, then $x = \text{decode}(z)$
- Prevents **posterior collapse**: encoder doesn't ignore the prior
- Creates a **smooth, structured latent space** for interpolation

Why Regularize $q_\phi(z|x)$ to $\mathcal{N}(0,1)$?

One would say: "The encoder must output $\mu = 0$, $\sigma = 1$"
Nope: It can output any μ, σ — but pays a **KL cost** if they deviate.

Why do this?

- Ensures the **decoder** learns to interpret $z \sim \mathcal{N}(0,1)$
- Enables **sampling**: generate new data by $z \sim \mathcal{N}(0,1)$, then $x = \text{decode}(z)$
- Prevents **posterior collapse**: encoder doesn't ignore the prior
- Creates a **smooth, structured latent space** for interpolation

Trade-off: KL loss vs. reconstruction — the ELBO balances them.

VQ-VAE [9]

Blabla

Outline : Diffusion Models

Diffusion: New advances in text-to-image I

Diffusion models allow to generate images conditioned on text

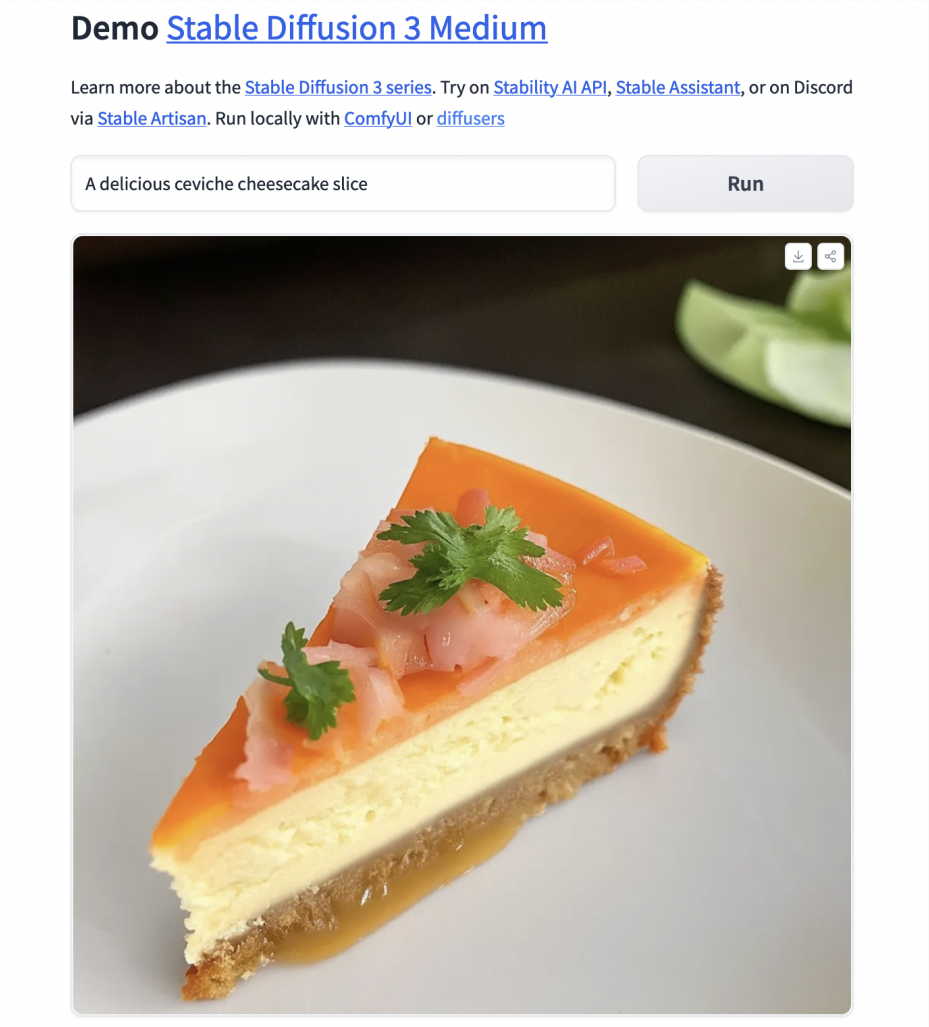


Figure 9: Examples of funny images obtained with diffusion models

Diffusion: New advances in text-to-image II

Definition

Probabilistic models that learn the data distribution by modeling the reverse of a diffusion process (adding noise step-by-step) to generate new data points.

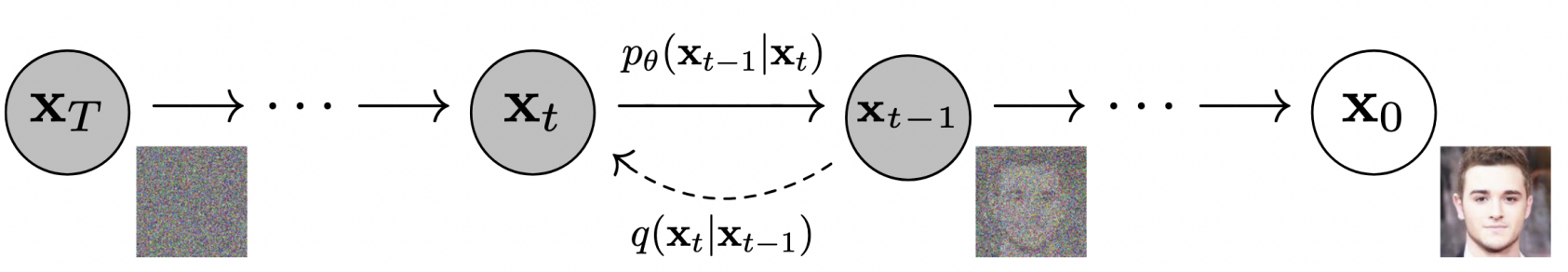


Figure 10: Denoising an image [2]

Process

- **Forward**: Add small amounts of noise to the data, which gradually becomes noisier over time steps, until resembling random noise
- **Reverse**: Learn to gradually remove noise to recover the original data, starting from pure noise and generating realistic data.

More here

Code example of a Diffusion Model generation

```python
import torch
from diffusers import DDPMPipeline

# Set the device to use our GPU or CPU
device = get_device()

# Load the pipeline
image_pipe = DDPMPipeline.from_pretrained("google/ddpm-celebahq-256")
image_pipe.to(device)

# Sample an image
image_pipe().images[0]
```

An example of code in a colab notebook: available here.

Note that it is way faster if generating using quantization.

U-net Encoder-Decoder

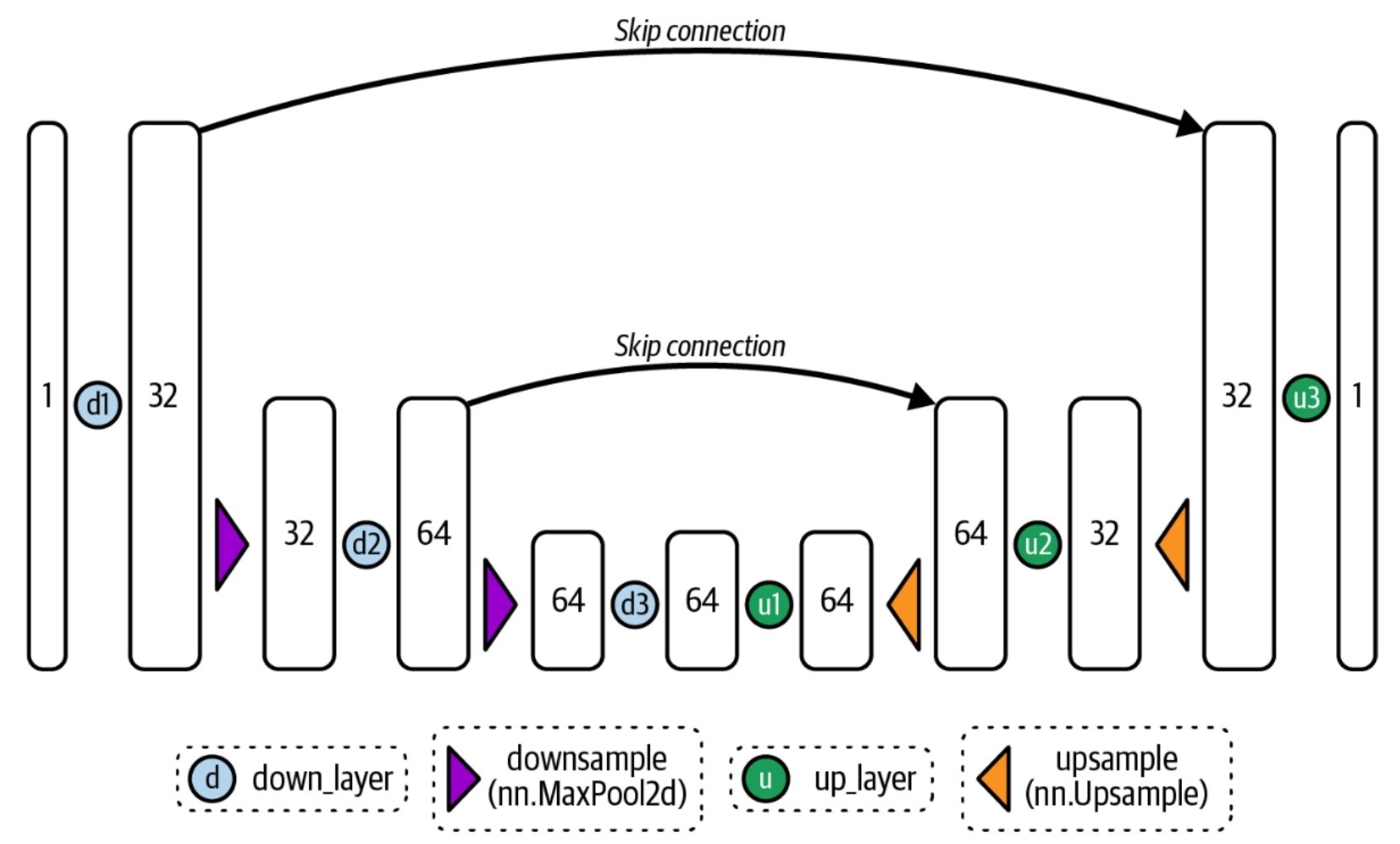


Figure 11: Unet architecture [6]

Diffusion: Stable Diffusion

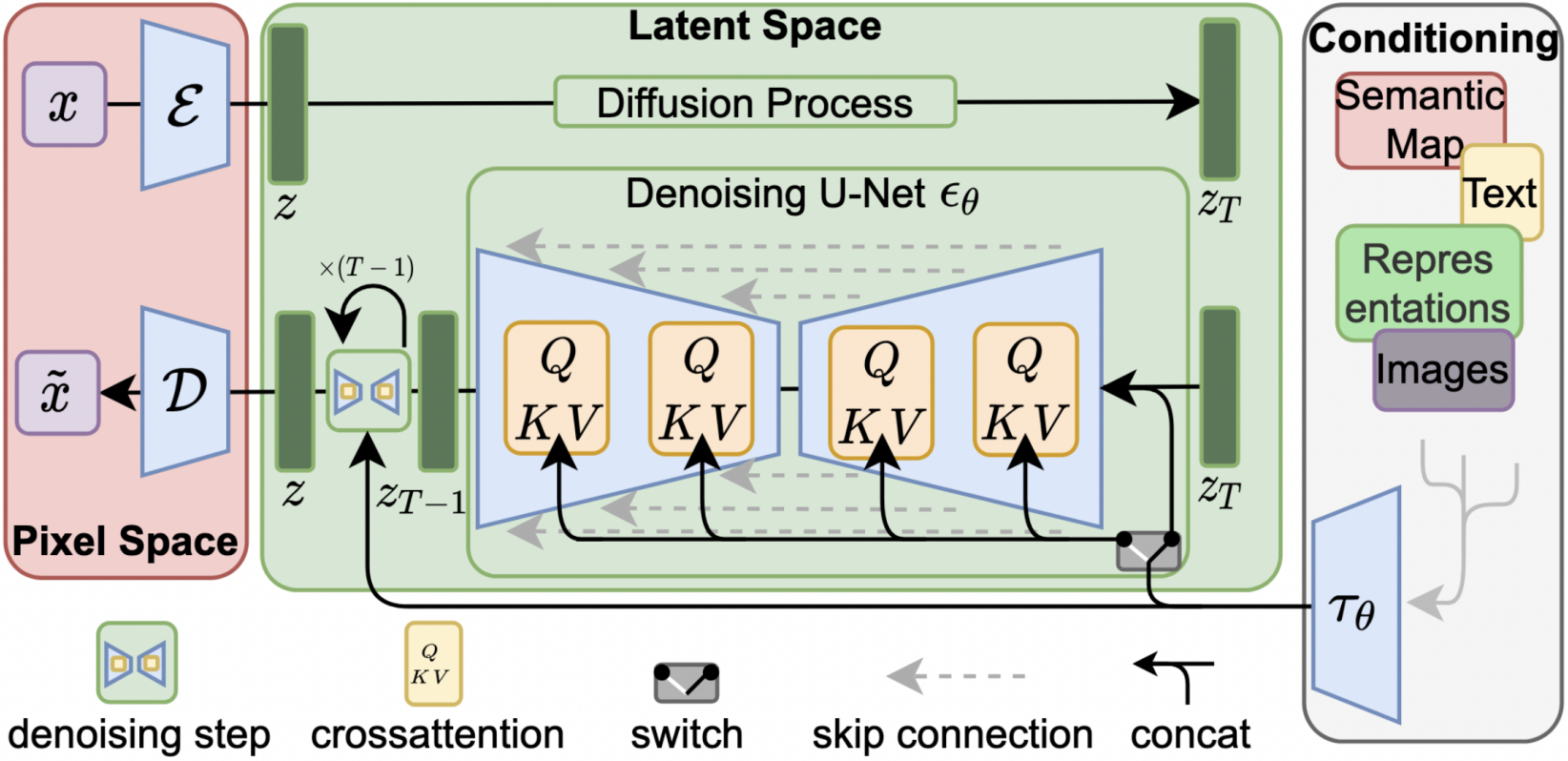


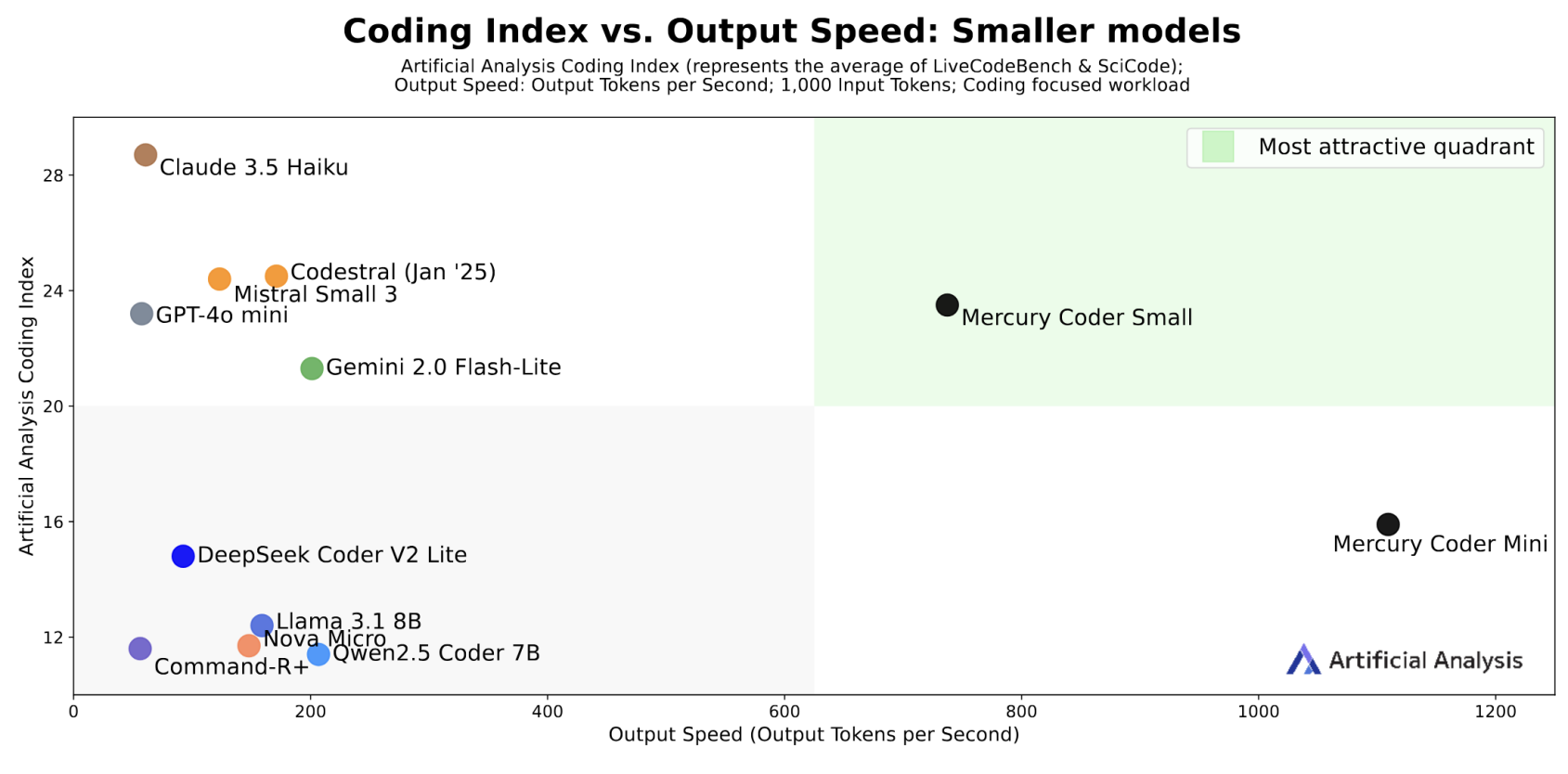
Figure 12: Stable Diffusion architecture [5]

- Use a Denoising Network [2] (e.g., U-Net) to reduce the noise at each step.
- Conditioning (e.g., text embedding) helps guide the model to generate a specific output using x-attention mechanism.

Textual Diffusion Models

It also exists for text. They have many qualities [3]:

- Faster (5-10x)
- Computationally Cheaper (5-10x)
- Similar Quality



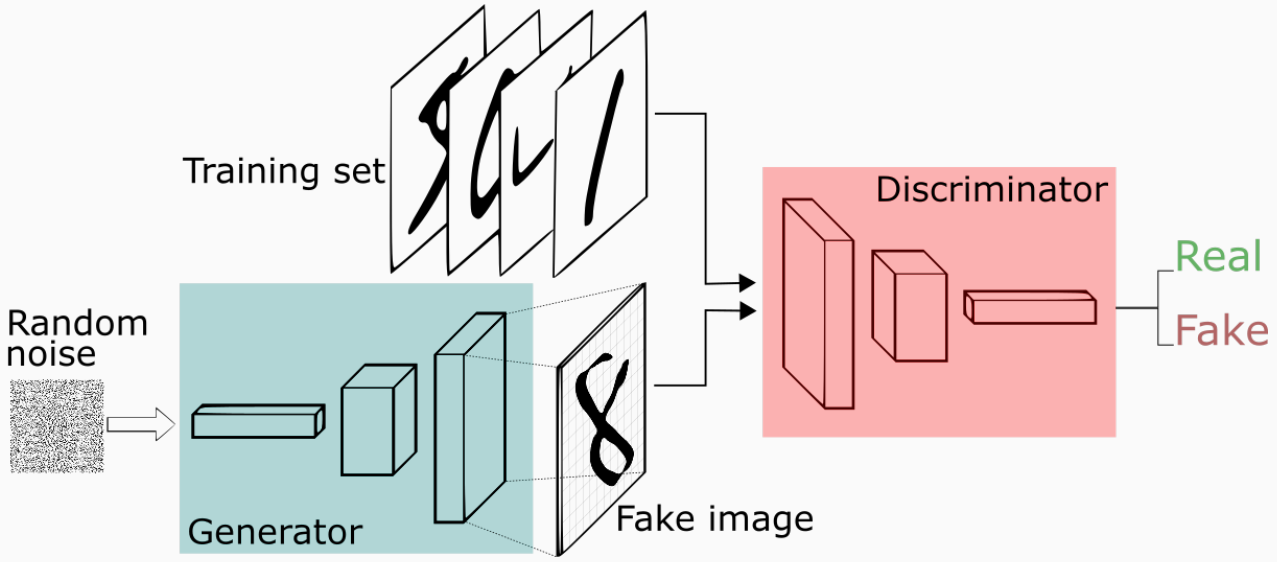
Outline : Generative Adversarial Networks

Generative Adversarial Network (GAN)

Principle

- Trains a model (**Generator**) to generate data that another model (**Discriminator**) should not be able to recognize as artificial.
- The discriminator is slow to learn

In the end, we obtain generated data (e.g., images) that resemble those from our dataset.



Generate High Quality Images

Two-Sample Test: Problem Setup

$S_1 = \{\mathbf{x} \sim P\}$ $S_2 = \{\mathbf{x} \sim Q\}$

Given a finite set of samples from two distributions: $S_1 = \{x \sim P\}$ and $S_2 = \{x \sim Q\}$, can we determine whether the samples are from the same distribution (i.e., $P = Q$)?

Test Statistic

This is the **two-sample test** problem:

- **Null hypothesis** H_0: $P = Q$
- **Alternative hypothesis** H_1: $P \neq Q$

A test statistic $T(S_1, S_2)$ quantifies the difference between S_1 and S_2.

Example: Difference in means

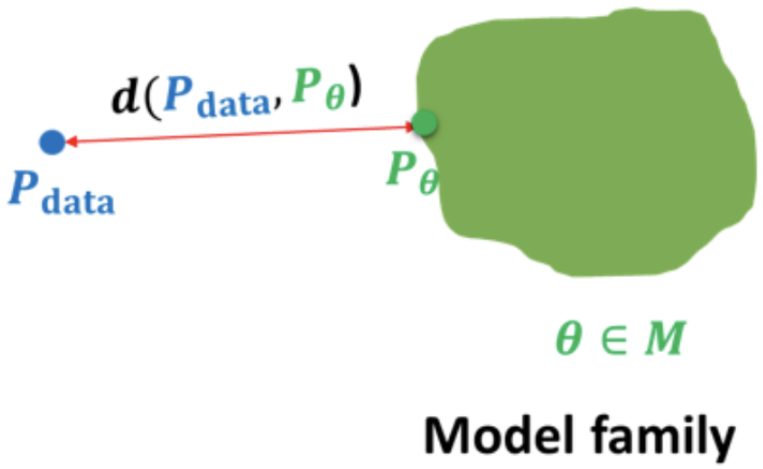
$$T(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} x - \frac{1}{|S_2|} \sum_{x \in S_2} x$$

- If $T > \alpha$ (threshold), **reject** H_0
- Otherwise, H_0 is consistent with the data

Key observation: T is **likelihood-free** — it uses only samples, not densities P or Q.

Connection to Generative Modeling

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



- We have real data samples: $S_1 = \mathcal{D} = \{x \sim p_{\text{data}}\}$
- A model with distribution p_θ, from which we can sample: $S_2 = \{x \sim p_\theta\}$

Idea: Two-sample test to measure discrepancy between p_{data} and p_θ.

Alternative notion of training: Minimize a two-sample test objective:

$$\min_\theta \mathcal{L}(\theta) = T(S_1, S_2)$$

Goal: Make p_θ indistinguishable from p_{data}.

A model can learn this statistic T

Finding a good test statistic T in high dimensions is difficult. Moreover, we know $S_1 \sim p_{\text{data}}$, $S_2 \sim p_\theta$, and initially $p_{\text{data}} \neq p_\theta$.

Key idea: *Learn* the test statistic to automatically detect differences between S_1 and S_2, by training a **classifier** (called a *discriminator*) to distinguish them.

A model can learn this statistic T

Finding a good test statistic T in high dimensions is difficult. Moreover, we know $S_1 \sim p_{\text{data}}$, $S_2 \sim p_\theta$, and initially $p_{\text{data}} \neq p_\theta$.

Key idea: *Learn* the test statistic to automatically detect differences between S_1 and S_2, by training a **classifier** (called a *discriminator*) to distinguish them.

Discriminator learns a powerful statistic for the two-sample test

Binary classifier D_ϕ (e.g., neural network) to distinguish:

- Real samples $x \sim p_{\text{data}}$ (label $y = 1$)
- Fake samples $x \sim p_\theta$ (label $y = 0$)
- **Test statistic**: Negative classification loss.
- **Low loss** $\Rightarrow$ samples are easy to distinguish ($p_{\text{data}} \neq p_\theta$)
- **High loss** $\Rightarrow$ samples are hard to distinguish ($p_{\text{data}} \approx p_\theta$)
- **Goal**: Maximize the test statistic (support $H_1 : p_{\text{data}} \neq p_\theta$), i.e., minimize classification loss.

Discriminator Training Objective

Maximize the following w.r.t. D_ϕ:

$$\max_{D_\phi} V(p_\theta, D_\phi) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{x\sim p_\theta}[\log(1 - D_\phi(x))]$$

Approximated from samples:

$$V \approx \sum_{x\in S_1} \log D_\phi(x) + \sum_{x\in S_2} \log(1 - D_\phi(x))$$

This is binary cross-entropy:

- Predict $D_\phi(x) \approx 1$ for $x \in S_1$ (real)
- Predict $D_\phi(x) \approx 0$ for $x \in S_2$ (fake)

Optimal discriminator is $D^*_\theta(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_\theta(x)}$. Which makes impossible to discriminate when $p_\theta = p_{\text{data}}$: $D^*_\theta(x) = \frac{1}{2}$

GAN Objective: Minimax Loss

The GAN training objective is:

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

GAN Objective: Minimax Loss

The GAN training objective is:

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

Discriminator D_ϕ tries to:

- Maximize the probability of assigning correct labels:
- $\log D_\phi(x)$ for real data
- $\log(1 - D_\phi(G_\theta(z)))$ for fake data

Generator G_θ tries to:

- Minimize $\log(1 - D_\phi(G_\theta(z)))$, i.e., *fool* the discriminator
- Make $p_\theta(x)$ indistinguishable from p_{data}

GAN Objective: Minimax Loss

The GAN training objective is:

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

Discriminator D_ϕ tries to:

- Maximize the probability of assigning correct labels:
- $\log D_\phi(x)$ for real data
- $\log(1 - D_\phi(G_\theta(z)))$ for fake data

Generator G_θ tries to:

- Minimize $\log(1 - D_\phi(G_\theta(z)))$, i.e., *fool* the discriminator
- Make $p_\theta(x)$ indistinguishable from p_{data}

However, it is a Nash equilibrium in a two-player non-cooperative game. **Updating the gradient of both models concurrently (updating independently of the other player) cannot guarantee a**

GAN Objective: Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

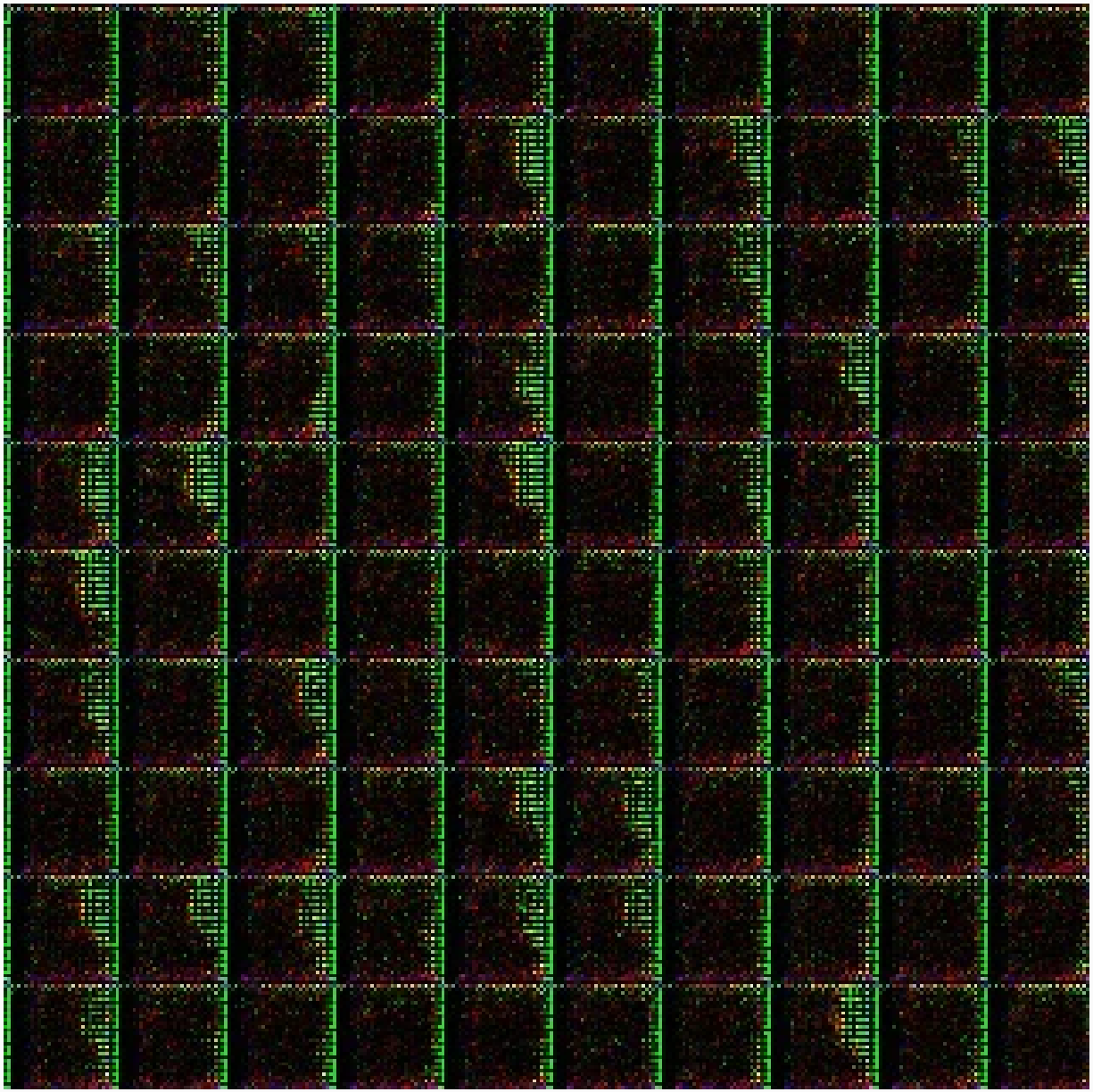
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Generative Adversarial Networks

In-depth GAN explanations here or there ; Code Tutorial ; GAN Applications ;

ImageNet GAN generation visualization



GANs: Creative Adversarial Network (CAN)

Same principle as GAN, but with two discrimination stages

- The first stage discriminates whether an image is real or fake: this encourages the generator to create images that resemble the data distribution of the dataset (i.e., look realistic).
- The second stage classifies the artistic style of the image: this encourages the generator to create images not associated with any known class. The result is an image that looks plausible and realistic, yet does not belong to any known artistic movement.

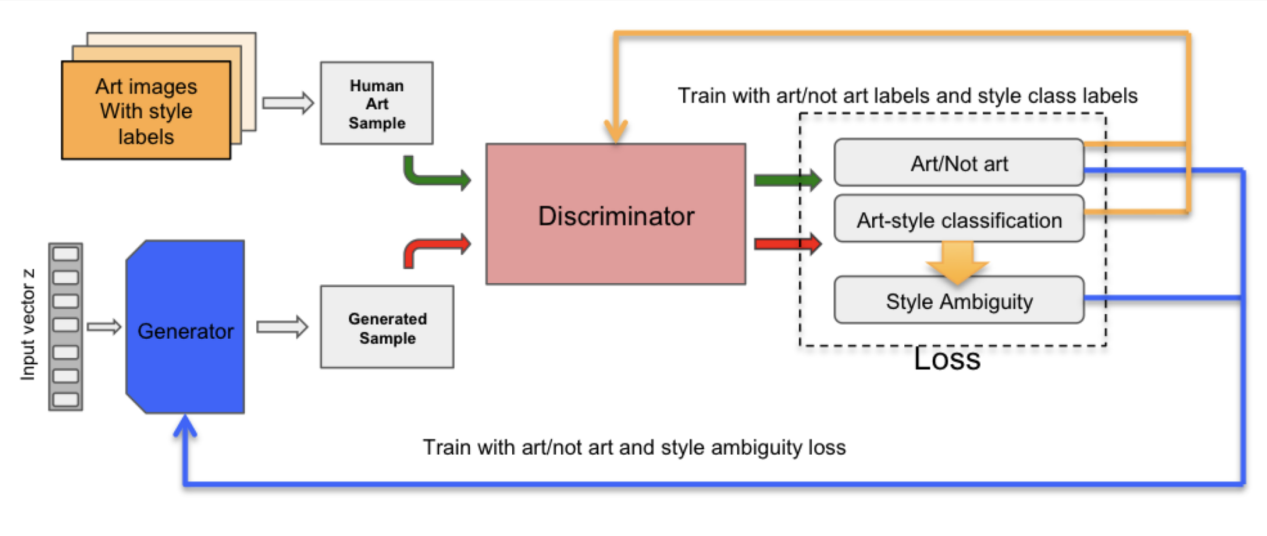


Figure 13: Discriminator has two heads: one for realism, one for artistic style.

GANs: Creative Adversarial Network (CAN)

TensorFlow Implementation ; Paper

Example of Image Edition

Simple way to edit a real picture:

1. Find the latent vector that will generate an image representing this face
2. Move this vector in the "direction of age/smile/gender" in the latent

Image Inversion in GANs

Figure 14: All the faces have their latent z that can generate it.

Image Inversion in GANs

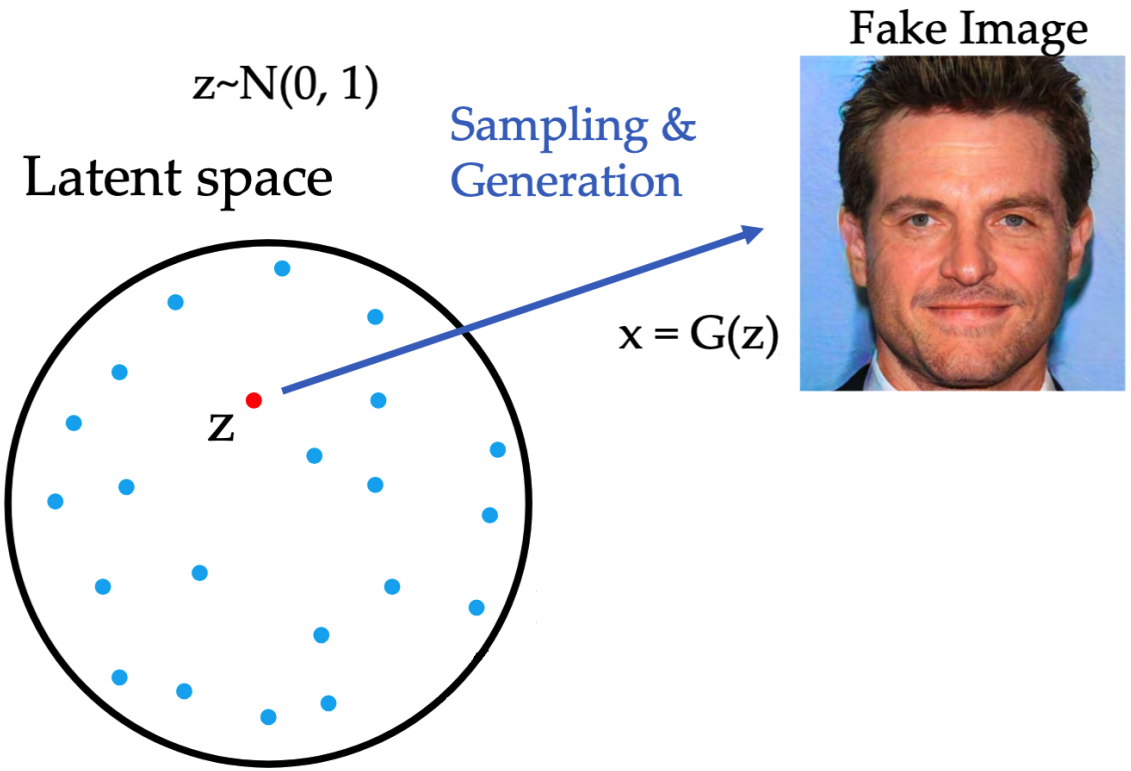


Figure 14: It is about to find the z^* which generate the picture

Image Inversion in GANs

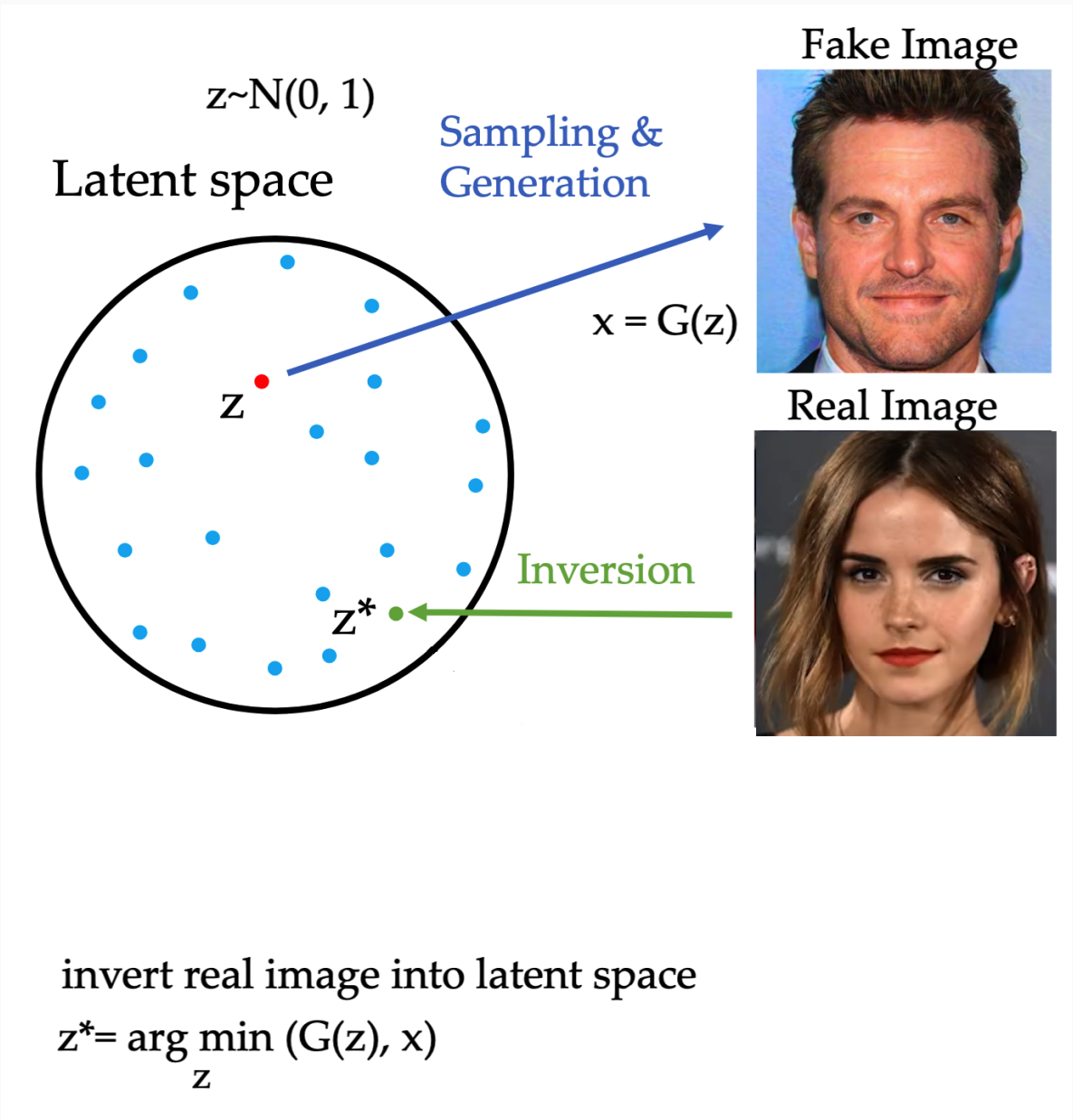


Figure 14: It is about to find the z^* which generate the picture

Image Inversion in GANs

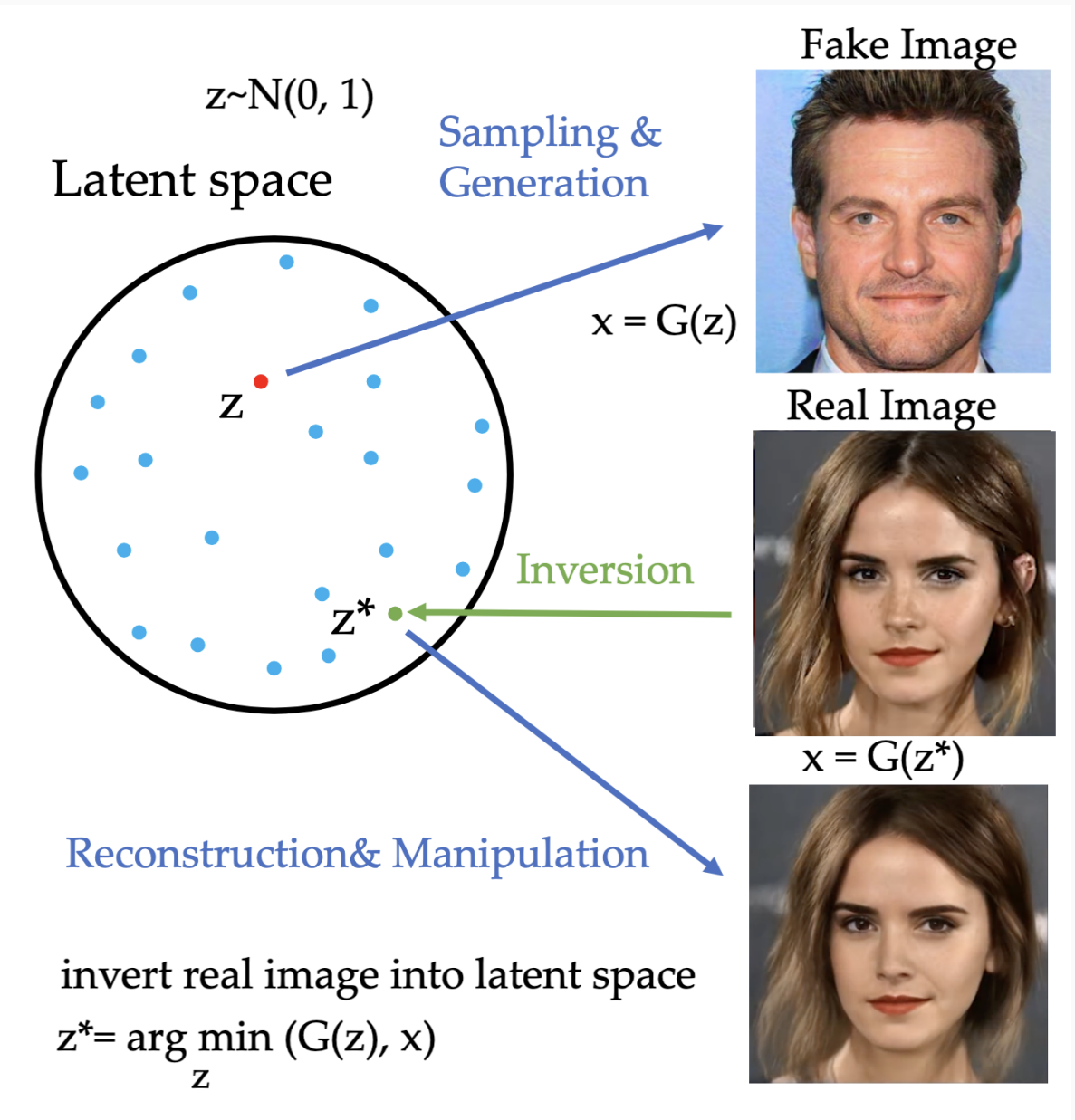


Figure 14: It is about to find the z^* which generate the picture

Image Edition with GANs

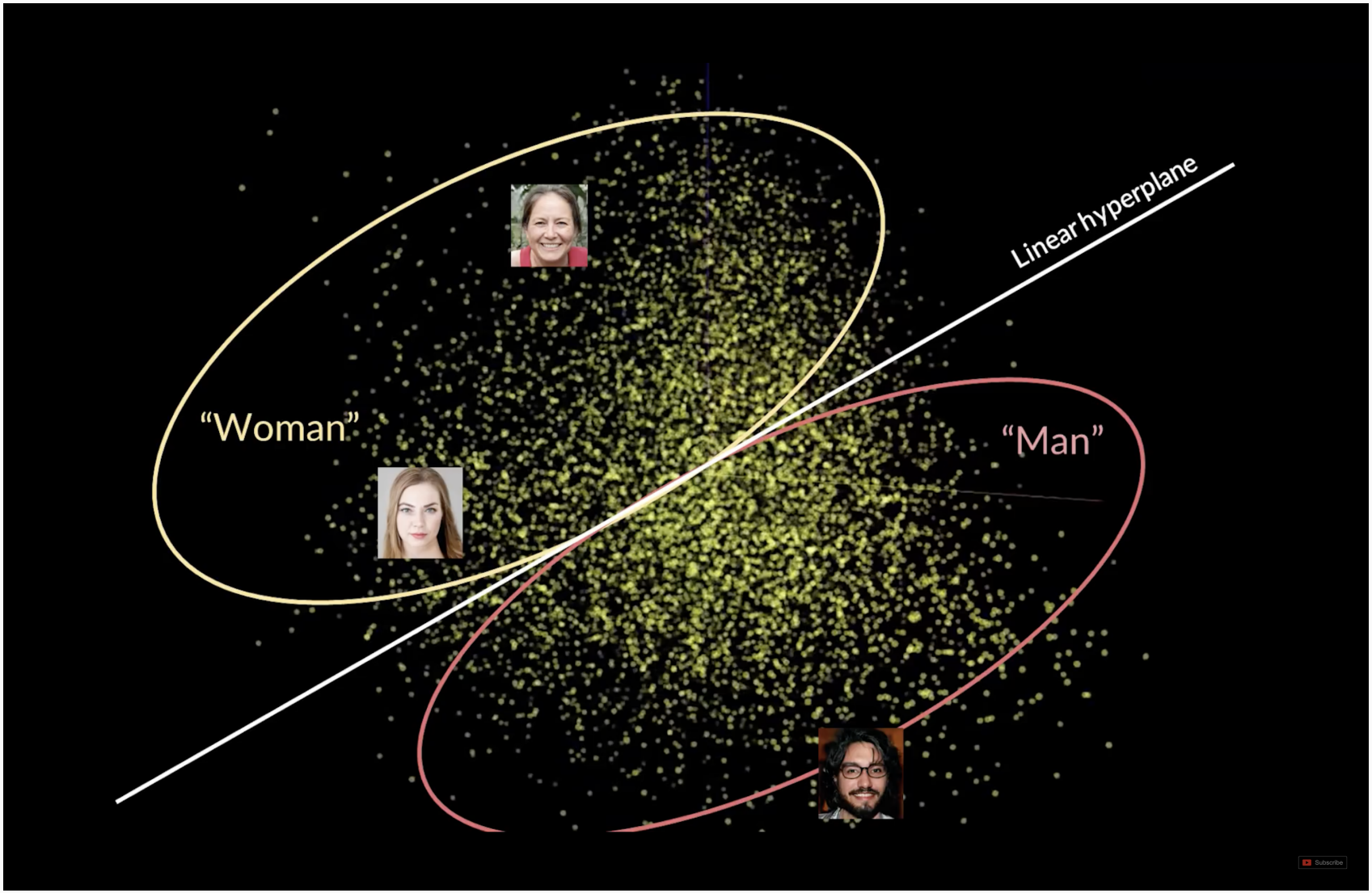


Figure 15: Classify z regarding gender

Image Edition with GANs

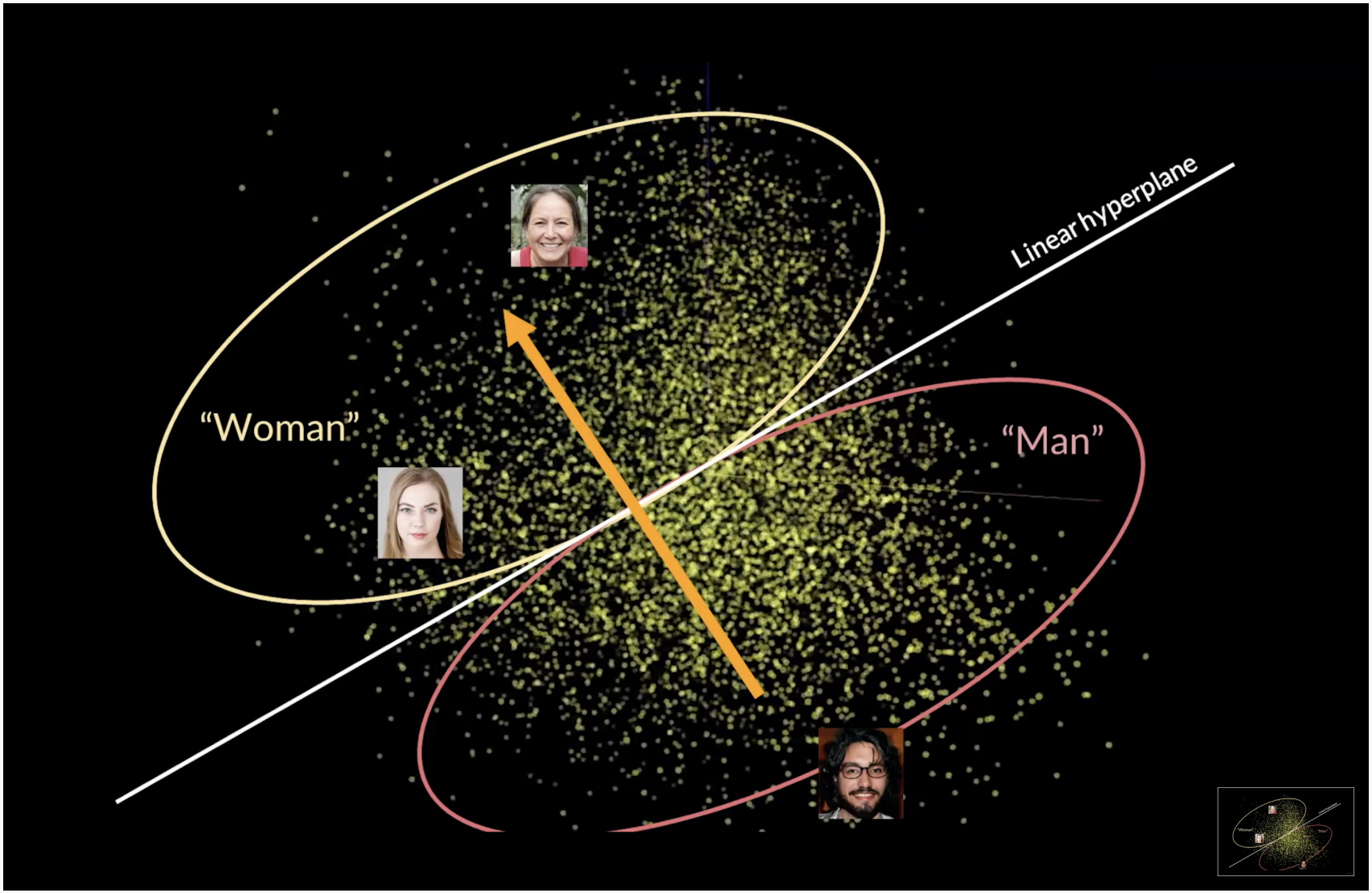


Figure 15: Take the normal of the hyperplan

Image Edition with GANs

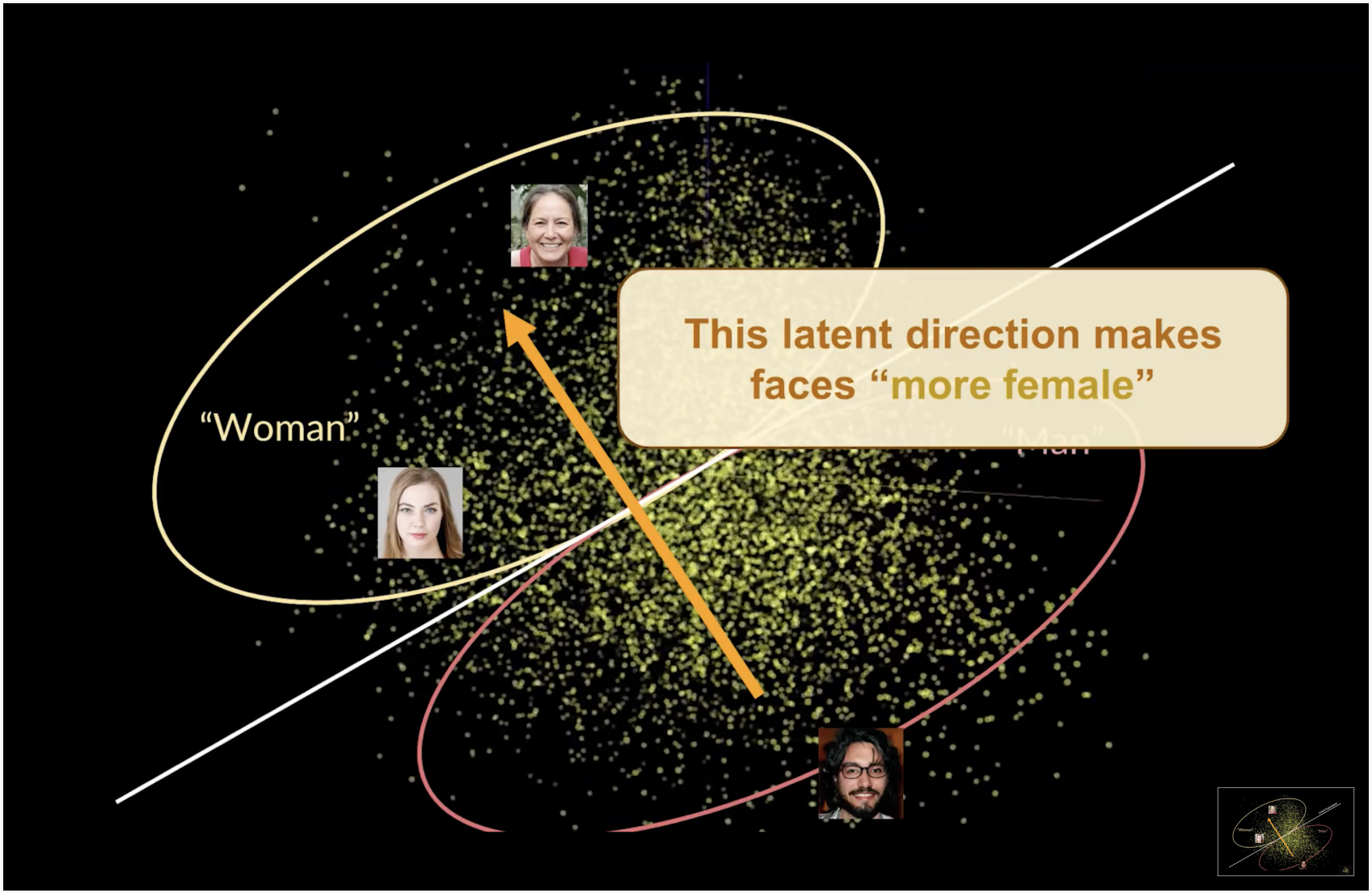


Figure 15: Edit images with this direction

Outline : Other

Neural Style Transfer

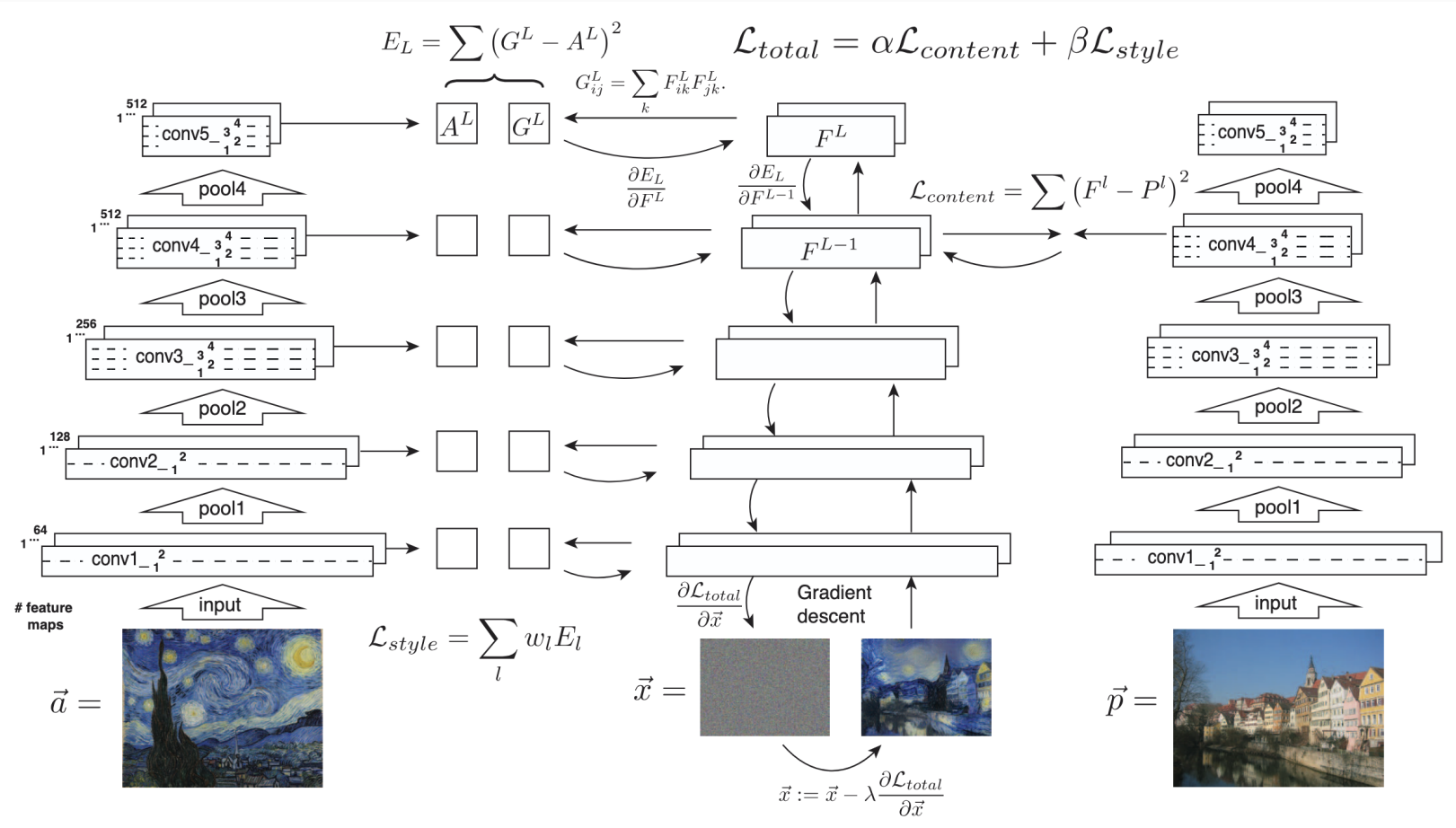
Principle

Applies the style of one image to another by separating the content (high-level representation) and the style (low-level representation).

PyTorch Tutorial; Paper

Neural Style Transfer

- **Optimization:** A white noise image **x** is iteratively optimized. At each step, its features F_l (content) and G_l (style) are computed.
- **Loss computation:** Content loss $\mathcal{L}_{\text{content}} = \|F_l - P_l\|^2$ and style loss $\mathcal{L}_{\text{style}} = \sum_l \|G_l - A_l\|^2$ minimized via backpropagation.
- **Output:** Optimized **x** matches the content of **p** and the style of **a**.



Neural Style Transfer: result

Neural Style Transfer: result

Image Edition with GANs: transferring style

Figure 16: Image-to-image translation based on StyleGan [4] allows for toonification using this model

Image Edition with GANs: transferring style

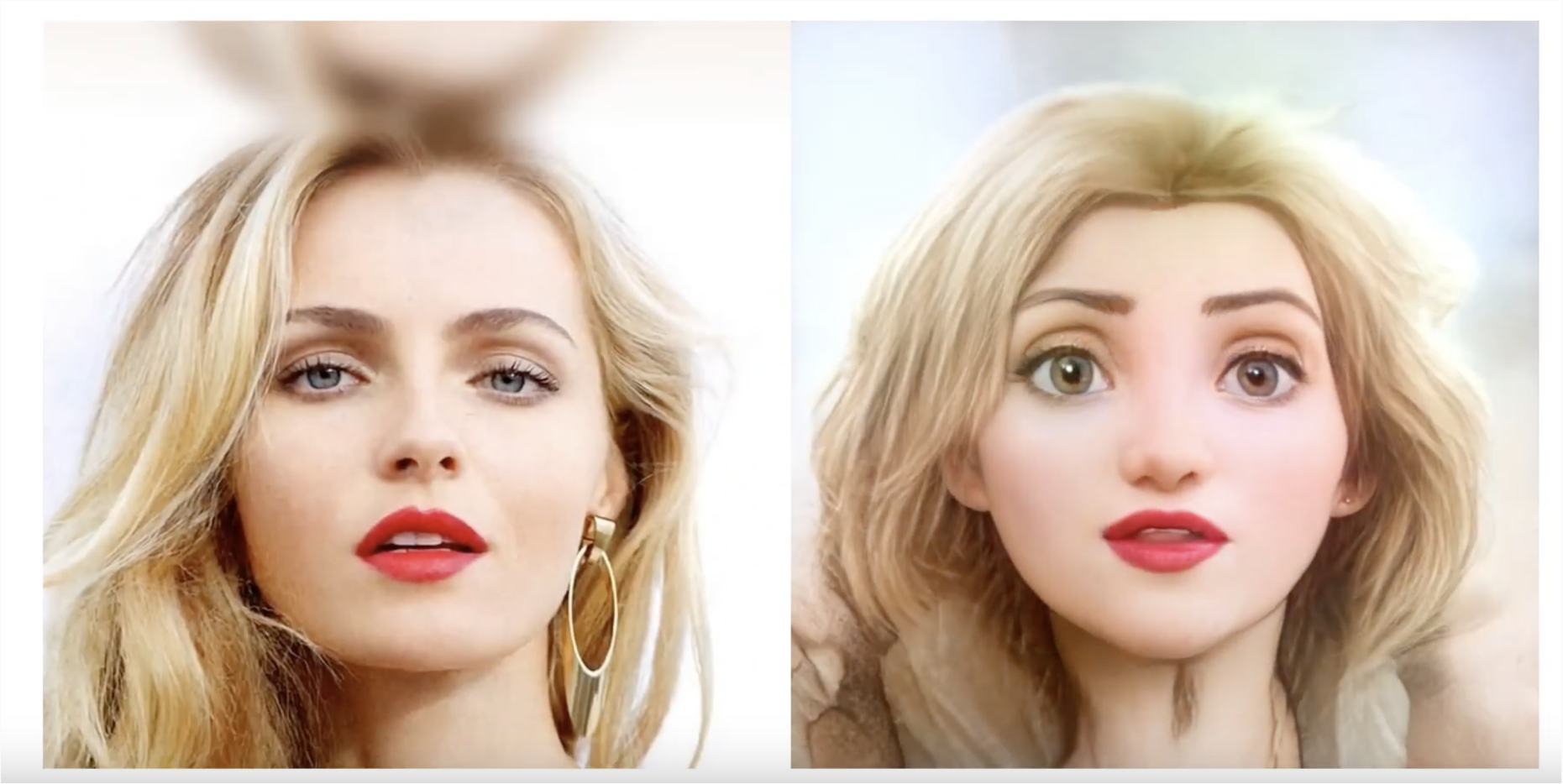


Figure 16: Image-to-image translation based on StyleGan [4] allows for toonification using this model

Outline : Conclusion

Summary Table: Generative Model Comparison

Feature	Autoregressive	Implicit (GAN)	Prescribed (VAE, Diffusion)
Training	Stable	Unstable	Stable
Likelihood	Exact	No	Approximate
Sampling	Slow	Fast	Fast
Compression	Lossless	No	Lossy
Representation	No	No	Yes

Green = Advantage, Red = Limitation, Yellow = Trade-off.

Table Explanation

- **Training**: GANs suffer from instability and mode collapse; AR, VAE and Diffusion are more stable.
- **Likelihood**: AR models compute exact probabilities (useful for anomaly detection). GANs have no likelihood. VAE uses ELBO; Diffusion uses score matching.
- **Sampling**: AR models are slow (sequential). GANs sample in one pass. Diffusion is fast per step but requires many steps.
- **Compression**: Only AR models allow lossless compression (e.g., Bits-Back). Others are inherently lossy.
- **Representation Learning**: VAEs and Diffusion offer meaningful latent spaces (e.g., for editing). GANs require extra work to invert.

Takeaways

There is **no single best model** — only the **best model for the task**:

- Need quality? $\rightarrow$ GAN or Diffusion
- Need latent space? $\rightarrow$ VAE or Diffusion
- Need likelihood? $\rightarrow$ AR or VAE
- Need speed? $\rightarrow$ GAN
- Need compression? $\rightarrow$ Autoregressive

Useful resources

- Ermon, S. (2023). CS236: Deep Generative Models [1]
- Sanseviero, O. et al. (2025). Hands-On Generative AI with Transformers and Diffusion Models. [7]
- Tomczak, J. M. (2022). Deep Generative Modeling. [8]

Questions?

References i

S. Ermon.
CS236: Deep Generative Models, 2023.

J. Ho, A. Jain, and P. Abbeel.
Denoising diffusion probabilistic models.
In Advances in Neural Information Processing Systems, volume 2020-Decem, pages 1–25, 2020.

I. Labs, S. Khanna, S. Kharbanda, S. Li, H. Varma, E. Wang, S. Birnbaum, Z. Luo, Y. Miraoui, A. Palrecha, S. Ermon, A. Grover, and V. Kuleshov.
Mercury: Ultra-Fast Language Models Based on Diffusion.
pages 1–15, 2025.

References ii

E. Richardson, Y. Alaluf, O. Patashnik, Y. Nitzan, Y. Azar, S. Shapiro, and D. Cohen-Or.
Encoding in Style: A StyleGAN Encoder for Image-to-Image Translation.
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 2287–2296, 2021.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer.
High-Resolution Image Synthesis with Latent Diffusion Models.
CVPR, 2022.

References iii

O. Ronneberger, P. Fischer, and T. Brox.
U-net: Convolutional networks for biomedical image segmentation.
In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer, 2015.

O. Sanseviero, P. Cuenca, A. Passos, and J. Whitaker.
Hands-On Generative AI with Transformers and Diffusion Models.

2025.

J. M. Tomczak.
Deep Generative Modeling.
2022.

References iv


A. Van Den Oord, O. Vinyals, and K. Kavukcuoglu.
Neural discrete representation learning.
Advances in Neural Information Processing Systems, 2017-Decem(Nips):6307–6316, 2017.