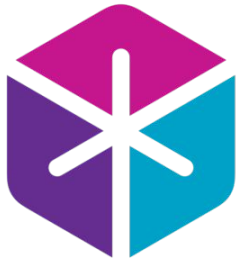
Lecture 04

Latent Spaces, Neural network architectures

MIT IAP 2026 | Jan 28, 2025

Peter Holderrieth and Ron Shprints

Sponsor: Tommi Jaakkola



Class Overview

- **Lecture 1 - Flow and Diffusion Models**
- **Lecture 2 - Flow Matching:** Training algorithm.
- **Lecture 3 - Score Matching, Guidance:** How to condition on a prompt.

- **Lecture 4 - Build Image Generators: Latent spaces + Network architectures**
- **Lecture 5 - Advanced Topics:** Discrete diffusion models + distilled models

Section 6:

Latent Spaces

Goal: Understand how data can be transformed into latents that can be more efficiently modelled

The Need for Latent Spaces

$$z \in \mathbb{R}^d$$

- High-resolution image: 3 color channels, height 600, width 1000 → the resulting dimension is: 3*600*1000 **= 1.8 million!**
- **This is a very high-dimensional space!!!!**

The Need for Latent Spaces

- High-resolution image: 3 color channels, height 600, width 1000 → the resulting dimension is: 3*600*1000 **= 1.8 million!**
- **This is a very high-dimensional space!!!!**

Several problems:

- GPU memory blows up
- Learning problem is very hard
- Redundancy: Closeby pixels highly correlated

The Need for Latent Spaces

- High-resolution image: 3 color channels, height 600, width 1000 → the resulting dimension is: 3*600*1000 **= 1.8 million!**
- **This is a very high-dimensional space!!!!**

Several problems:

- GPU memory blows up
- Learning problem is very hard
- Redundancy: Closeby pixels highly correlated

Why is this a problem for diffusion models and not for supervised learning?

The Need for Latent Spaces

- High-resolution image: 3 color channels, height 600, width 1000 → the resulting dimension is: 3*600*1000 **= 1.8 million!**
- **This is a very high-dimensional space!!!!**

Several problems:

- GPU memory blows up
- Learning problem is very hard
- Redundancy: Closeby pixels highly correlated

Why is this a problem for diffusion models and not for supervised learning?

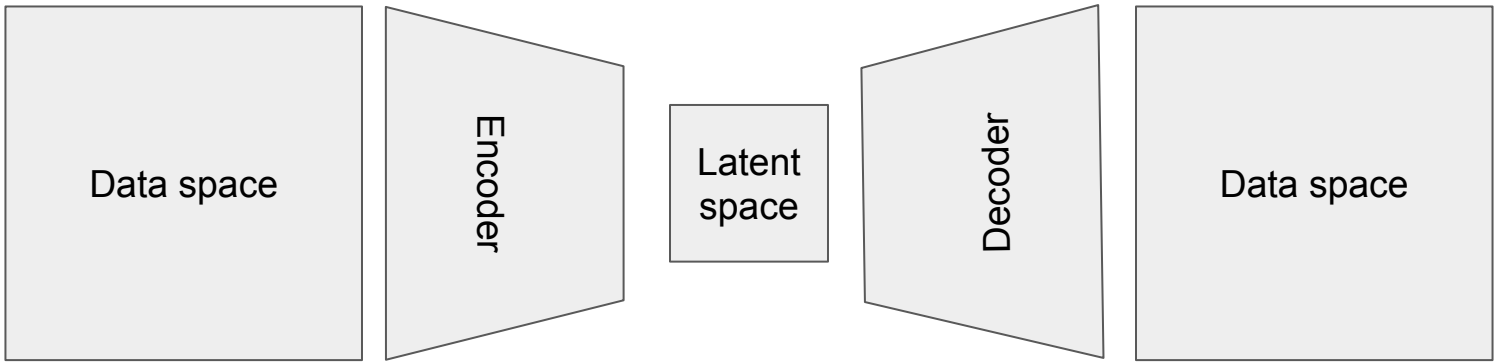
$$u_t^\theta : \mathbb{R}^d \to \mathbb{R}^d$$

We learn a vector field - output is very high-dimensional!

We apply vector field many times (simulation of ODE)!

Latent Spaces are constructed via autoencoders

Autoencoder = Encoder + Decoder



Latent Spaces are constructed via autoencoders

$\mathbb{R}^d$

Autoencoder = Encoder + Decoder

$\mathbb{R}^d$

Data space

Encoder

$\mathbb{R}^k$

Latent space

Decoder

Data space

Compression

$k < n$

The Problem of Standard Autoencoders

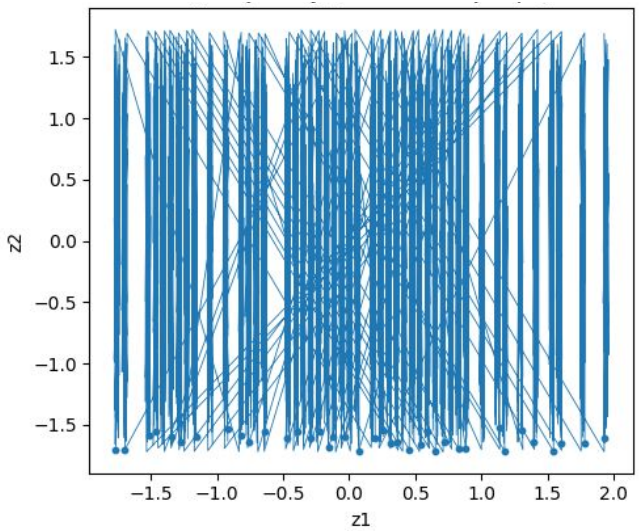
Our goal is to transform data into latents and then learn the distribution of latents.

Question: What happens to the data distribution when we transform it into latent space?

Answer: We don't know. The reconstruction loss does not say anything about the distribution.

Problem: We might make the learning problem (i.e. training) much harder. Therefore, we might be able to compress but not able to learn how to sample the distribution anymore.

A "bad" latent space



Goal: Create autoencoder that creates "nice" latent distribution

KL-divergence

For two probability densities q, p, the **Kullback-Leibler divergence** (KL-divergence) is defined as

$$D_{\mathrm{KL}}(q(x) \parallel p(x)) = \int q(x) \log \frac{q(x)}{p(x)} = \mathbb{E}_{X \sim q} \left[\log \frac{q(X)}{p(X)} \right].$$

The KL divergences measures how different two distributions are. It fulfills the natural properties:

$$D_{\mathrm{KL}}(q(x) \parallel p(x)) \geq 0,$$

$$D_{\mathrm{KL}}(q(x) \parallel p(x)) = 0 \quad \Leftrightarrow \quad q = p.$$

KL-divergence of two normal distributions

Two normal distributions are given:

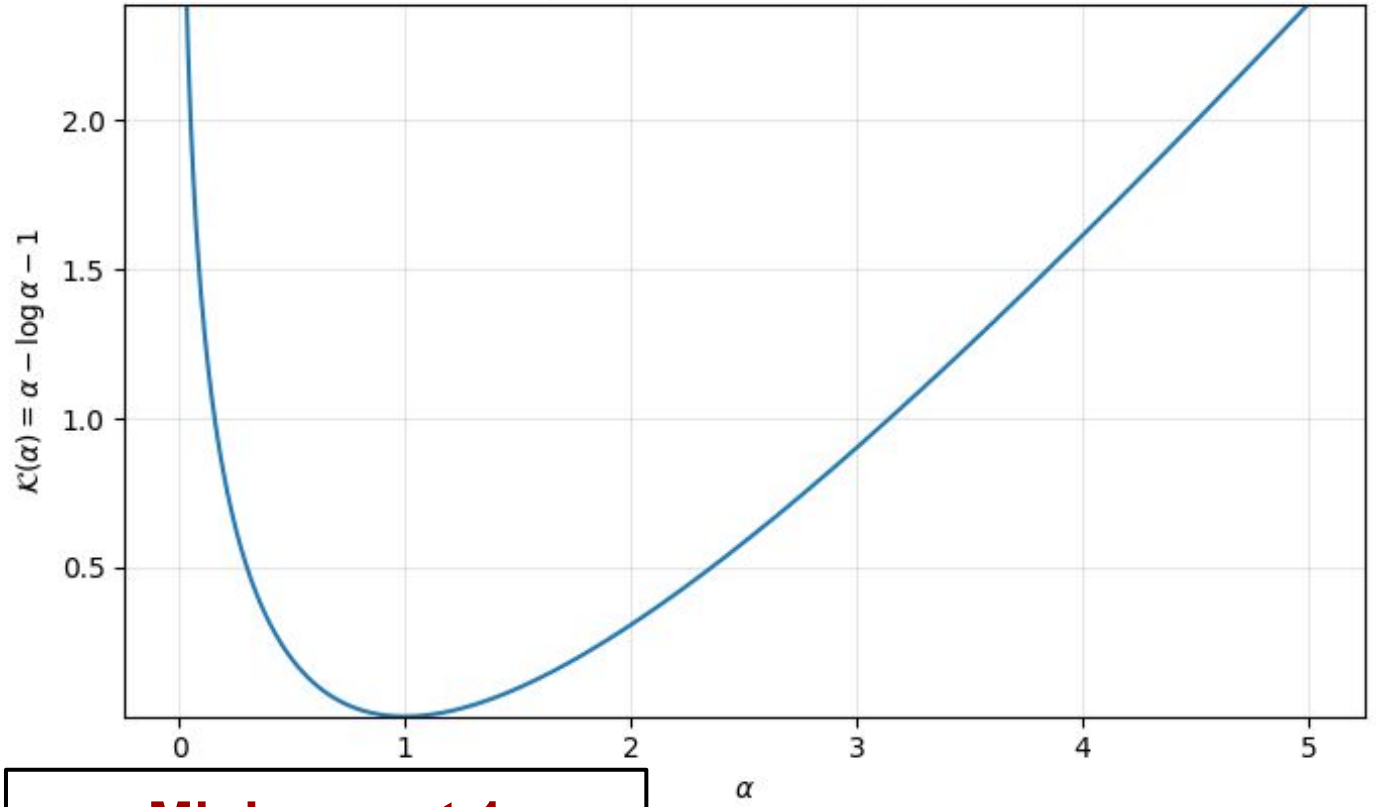
$$q(x) = \mathcal{N}(x; \mu_q, \sigma_q^2 I_d) \qquad p(x) = \mathcal{N}(x; \mu_p, \sigma_p^2 I_d)$$

The formula for the KL-divergence is given in this case by:

$$D_{\mathrm{KL}}(q \parallel p) = \frac{1}{2}\left(\mathcal{K}\left(\frac{\sigma_q^2}{\sigma_p^2}\right) + \frac{\|\mu_q - \mu_p\|^2}{\sigma_p^2}\right), \quad \text{where } \mathcal{K}(\alpha) = \sum_i \alpha_i - \log \alpha_i - 1.$$

Dist. of mean

Plotting $\mathcal{K}(\alpha) = \alpha - \log \alpha - 1$



Minimum at 1

KL-divergence of two normal distributions

Let two normal distributions be given:

$$q(x) = \mathcal{N}(x; \mu_q, \sigma_q^2 I_d) \qquad p(x) = \mathcal{N}(x; \mu_p, \sigma_p^2 I_d)$$

The formula for the KL-divergence is given in this case by:

$$D_{\mathrm{KL}}(q \parallel p) = \frac{1}{2}\left(\mathcal{K}\left(\frac{\sigma_q^2}{\sigma_p^2}\right) + \frac{\|\mu_q - \mu_p\|^2}{\sigma_p^2}\right), \quad \text{where } \mathcal{K}(\alpha) = \sum_i \alpha_i - \log \alpha_i - 1.$$

Dist. of variances

Dist. of mean

VAE - Training Algorithm

Algorithm 7 β-VAE Training Procedure

Require: Dataset of samples $x \sim p_{\text{data}}$, encoder networks $(\mu_\phi(x), \log \sigma^2_\phi(x))$, decoder network $\mu_\theta(z)$, latent dim k, constants $\beta \geq 0$, $\sigma^2 > 0$

1: **for** each mini-batch $\{x_i\}_{i=1}^B$ **do**

2: Encode each x_i: $\mu_i \leftarrow \mu_\phi(x_i), \quad \log \sigma_i^2 \leftarrow \log \sigma^2_\phi(x_i)$

3: Sample noise $\epsilon_i \sim \mathcal{N}(0, I_k)$

4: Reparameterize: $z_i \leftarrow \mu_i + \sigma_i \odot \epsilon_i$

5: Decode mean: $\hat{x}_i \leftarrow \mu_\theta(z_i)$

6: Reconstruction loss:

$$\mathcal{L}_{\text{recon}} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \frac{1}{2\sigma^2} \|x_i - \hat{x}_i\|^2$$

7: KL loss to the prior $p_{\text{prior}}(z) = \mathcal{N}(0, I_k)$:

$$\mathcal{L}_{\text{KL}} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \frac{1}{2} \sum_{j=1}^{k} \left(\mu_{i,j}^2 + \sigma_{i,j}^2 - \log \sigma_{i,j}^2 - 1\right)$$

8: Total loss: $\mathcal{L} \leftarrow \mathcal{L}_{\text{recon}} + \beta \, \mathcal{L}_{\text{KL}}$

9: Update $(\phi, \theta) \leftarrow \text{grad\_update}(\mathcal{L})$

10: **end for**

Latent Diffusion Models (LDMs) - Recipe

1. **Data**: Take all training data x1,....,xN (e.g. all images on the internet)
2. **Encoding**: Convert all images into latents (for VAE simply take the mean prediction)
3. **Latent data**: This gives us a dataset of latents z1,...,zN of significantly smaller size than high resolution images
4. **Latent diffusion model:** Build a diffusion model for the dataset of latents, i.e. the diffusion model now generates latent vectors!
5. **Decoding**: After sampling from the diffusion model, map generated latent back to data space

Return decoded image as sample!

Note: Exact same recipe as before. Just with a transformed dataset!

Virtually all AI-generated Images or Videos that you see are generated in latent spaces!

- **Latent space is key:** Memory would blow up too much otherwise. We need to allow the model to focus on key parts!

Example - Stable Diffusion: Images of shape [3,256,256] -> [4,32,32]

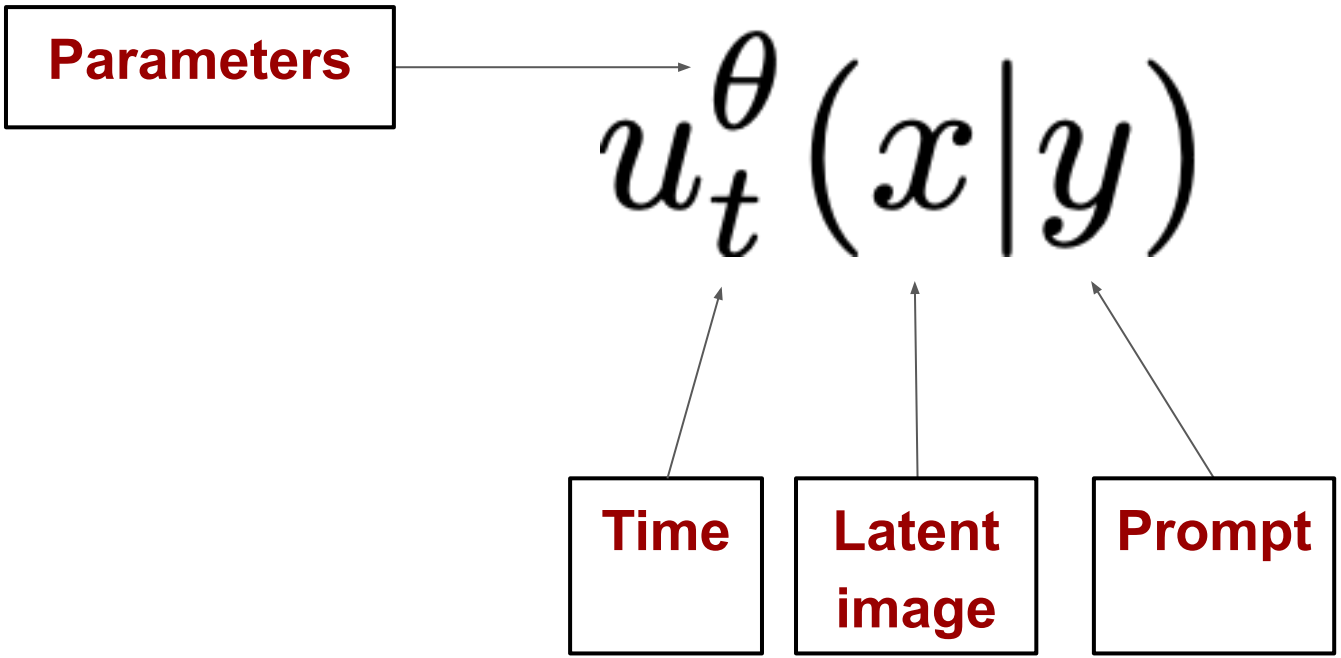
Example - FLUX 2.0: Images of shape [3,1024,1024] ->[32,64,64]

Section 7:

Neural network architectures

Goal: Understand how to build neural networks for diffusion models

Parameterizing a vector field in a neural network



Encoding Time

Time is only one-dimensional, while all other variables are high-dimensional.

To make it “count” more, we embed it via the following sinusoidal embedding into high-dimensional vector:

$$\text{TimeEmb}(t) = \frac{1}{\sqrt{d}} \left[\cos(2\pi w_1 t) \quad \cdots \quad \cos(2\pi w_{d/2} t) \quad \sin(2\pi w_1 t) \quad \cdots \quad \sin(2\pi w_{d/2} t) \right]^T$$

Frequencies are chosen as follows:

$$w_i \;=\; w_{\min} \left(\frac{w_{\max}}{w_{\min}} \right)^{\frac{i-1}{d/2-1}}, \qquad i = 1, \ldots, d/2.$$

Note: Specific format is less important than the time embedding is a d-dimensional normed vector:

$$\|\text{TimeEmb}(t)\| \;=\; 1$$

Encoding Language Prompt

“A dog running on grass in a park at sunshine in an Italian city.””

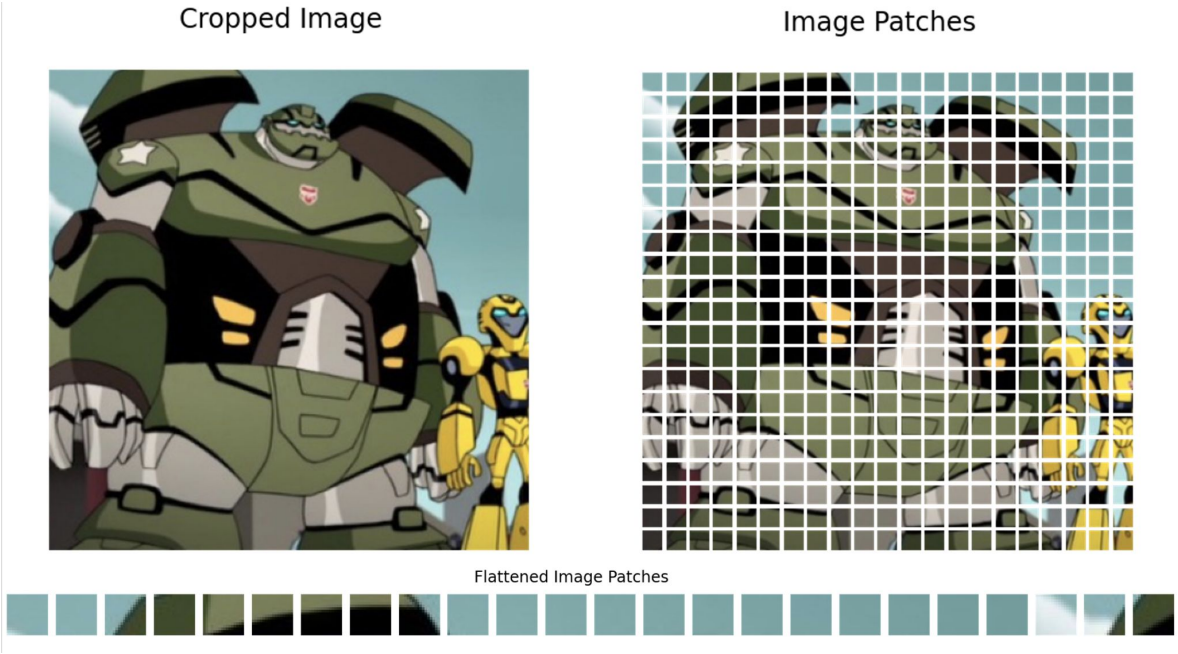
To embed text, most models rely on pre-trained language embeddings:

- Use CLIP embeddings (Contrastive Language-Image Pre-training)
- T5 embeddings, etc. (other pre-trained models)
- Can also use LLM embeddings

Prompt embedding: The result of these embeddings is that the prompt a sequence of vectors of length S:

$$\text{PromptEmbed}(y_{\text{raw}}) \in \mathbb{R}^{S \times k}$$

Patchify: Turn an Image into a Sequence of Vectors



$$x \in \mathbb{R}^{3 \times H \times W}$$

$$\tilde{x} = \text{Patchify}(x)$$

$$\tilde{x} \in \mathbb{R}^{L \times k}$$

L is the sequence length

Source:
https://gowrishankar.info/blog/transformers-everywhere-patch-encoding-technique-for-vision-transformersvit-explained/

Diffusion Transformer (DiT)

Transformer model for diffusion models.

Adapts transformer model for specific form that is required for diffusion models.

Peebles, W., & Xie, S. (2023). Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 4195-4205).

Diffusion Transformer Overview

Inputs

$$\tilde{t} = \text{TimeEmb}(t) \in \mathbb{R}^k \qquad \tilde{x}_0 = \text{PatchEmb}(x) \in \mathbb{R}^{N \times k}$$

$$\tilde{y} = \text{PromptEmb}(y) \in \mathbb{R}^{S \times k}$$

Attention loop

$$\tilde{x}_{i+1} = \text{DiTBlock}(\tilde{x}_i, \tilde{t}, \tilde{y}) \in \mathbb{R}^{N \times k} \quad (i = 0, \ldots, N-1)$$

Unpatchify

$$u = \text{Unpatchify}(\tilde{z}_N \tilde{W}) \in \mathbb{R}^{C \times H \times W}$$

Background - Scaled dot-product attention

Scaled dot-product attention. Given queries $Q \in \mathbb{R}^{N \times d_h}$, keys $K \in \mathbb{R}^{M \times d_h}$, and values $V \in \mathbb{R}^{M \times d_h}$,

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right) V \in \mathbb{R}^{N \times d_h},$$

where the softmax is applied row-wise.

Attention is an operation that maps sequences of vectors to sequences of vectors via simple matrix multiplication + softmax operation.

3 inputs:

1. Keys
2. Values
3. Queries

DiTBlock Overview

- **Image via Self-attention**: Process image itself

  Queries = image, Keys = image, values = image

- **Text via Cross-attention**: Attend image to text

  Queries = image, Keys = text embeds, values = text embeds

- **Time via Adaptative Layer Normalization**:

  Scaling and offset of normalization is determined by time variable

All inputs are combined by summing them up.

Best way to understand a transform is to implement it! See Lab 03! Further, lecture notes contain detailed information.

Bonus:

Large-scale diffusion models

Goal: Understand how diffusion models are trained at scale

Case Study: Stable Diffusion 3

Image source: Scaling Rectified Flow Transformers for High-Resolution Image Synthesis [1]

Overview:

- Flow Matching model with “straight line” schedulers (CondOT path)
- Classifier-free guidance with weight 2.0 - 5.0
- Flow Matching in latent space (use pre-trained VAE)
- Number of parameters of model: 8 billion
- Number of simulation/sampling steps: 50
- Dataset: LAION



Neural network architecture for Stable Diffusion 3

- Conditions on **CLIP** (coarse-grained) and **T5-XXL** (sequence-level) text embeddings via cross-attention.
- **MM-DiT architecture**: Extends DiT from class-conditioning to text-conditioning and processes text and images through the entire network via cross-attention

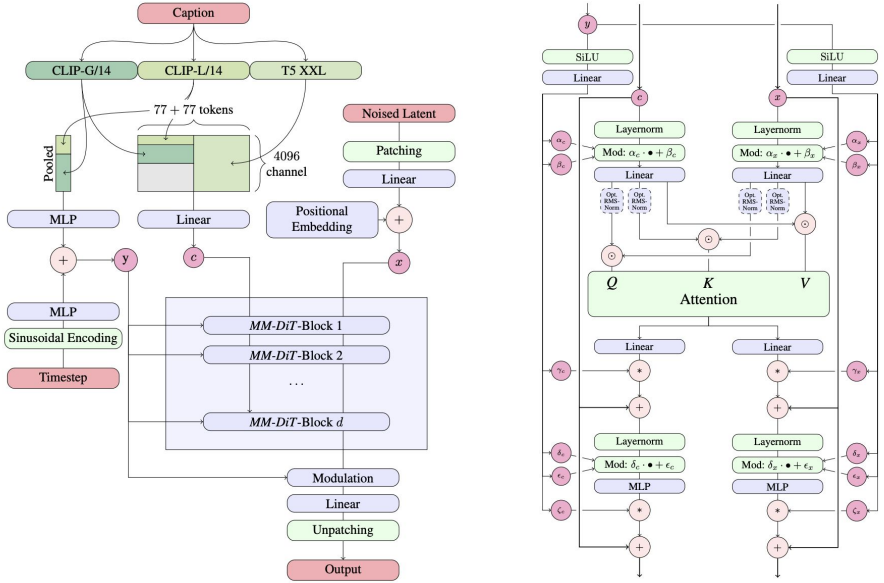


Image source: Scaling Rectified Flow Transformers for High-Resolution Image Synthesis [1]

Case Study - Meta MovieGen

Case Study: Meta MovieGen

Image source: Scaling Rectified Flow Transformers for High-Resolution Image Synthesis [1]

- Flow Matching model with “straight line” schedulers (CondOT path)
- Classifier-free guidance
- Flow Matching in latent space (use pre-trained VAE) - Really crucial for videos because of added time dimension
- Neural network architecture: DiT adapted to videos (what changs?!)
- Number of parameters of model: 30 billion
- **6,144 H100 GPUs!**

Bonus:

A guide to the diffusion literature

Goal: Understand different interpretations for diffusion models

Time conventions:

“Flow time convention”:

- Data: t = 1
- Noise: t = 0

Flow matching, rectified flows, stochastic interpolants

“Diffusion time convention”:

- Data t = 0
- Noise: t → infinity

Score-based diffusion models with SDEs

“Discrete time”:

- Use discrete time steps instead of continuous time steps
- No ODE or SDE but Markov chain
- DDIM ~= Probability flow ODE

DDPM
DDIM

Noising Procedure - how to corrupt data?

Probability path (here):

$$p_t(x|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$$

Flow matching, rectified flows

Interpolant function:

$$I_t(\epsilon, z) = \alpha_t \epsilon + \beta_t z$$

Stochastic interpolants

“Forward” diffusion process:

$$\mathrm{d}X_t = a_t(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t$$

Denoising diffusion models

Note: For Gaussian probability paths, the above procedures are equivalent.

Constructing noising procedures via forward noising processes

Forward SDE (data → noise)

$\mathbf{x}(0)$ —— $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$ ——→ $\mathbf{x}(T)$

score function

$\mathbf{x}(0)$ ←—— $\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$ —— $\mathbf{x}(T)$

Reverse SDE (noise → data)

The time-reversed SDE is a specific solution to the SDE extension trick that we discussed for a specific noise level. Empirically, this is often not the best solution in practice.

Here: Flow Matching

- Arguably most simple flow and diffusion algorithms
- Allows you to restrict yourself to flows
- Allows you go from arbitrary p_{init} to arbitrary p_{data}

Note: The method presented here allows to convert arbitrary distributions into arbitrary distributions!

Bridging arbitrary distributions - Example

Videos without audio → videos with audio

Low resolution images → high resolution images

Unperturbed cells → perturbed cells

etc.

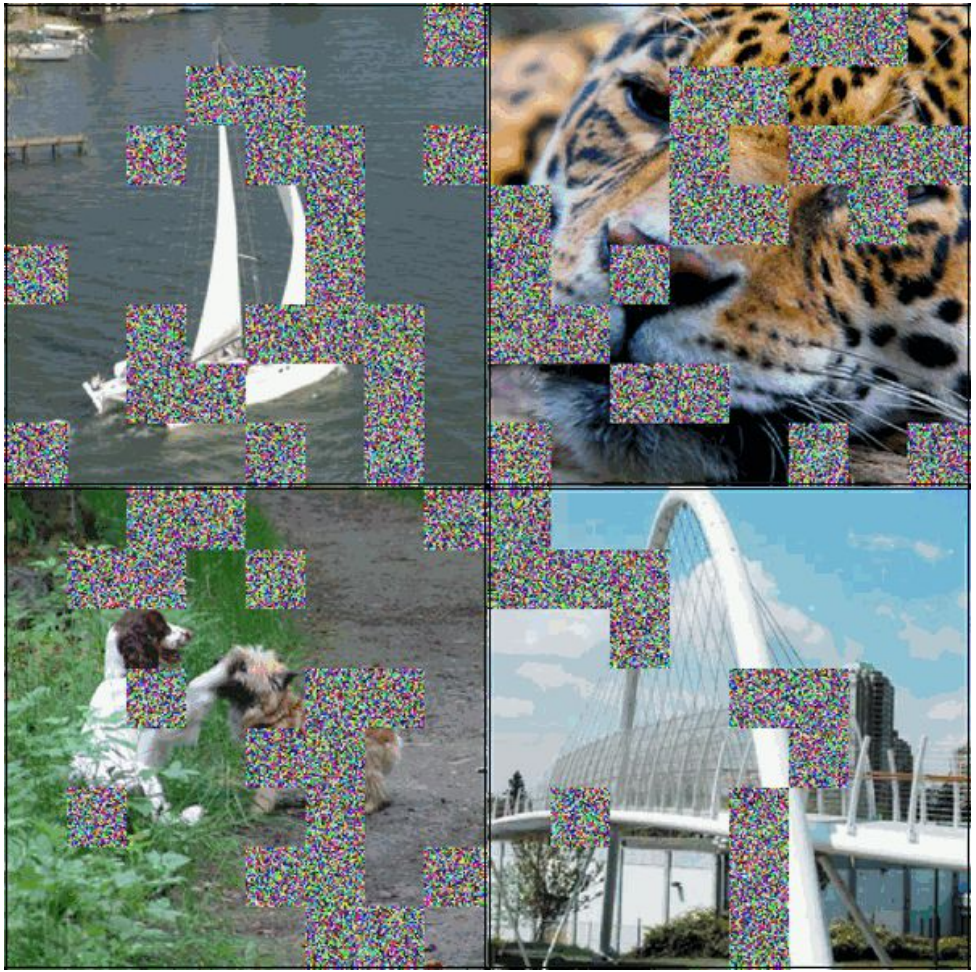


Figure credit: Michael Albergo

Next class:

Friday, 12:30pm-2pm

Discrete diffusion models

E25-111 (same room)