Universitat de Barcelona

Jordi Vitrià

Deep Learning

Unsupervised Learning II: Diffusion Models

Diffussion models and generative modeling

Portrait of a Catalan professor teaching the stable diffusion method in the style of cubism

https://labs.openai.com/e/h8QkKviSQyjqH4KFAP5UZGUD

Diffusion models and generative modeling

portrait of a professor in the style of cubism

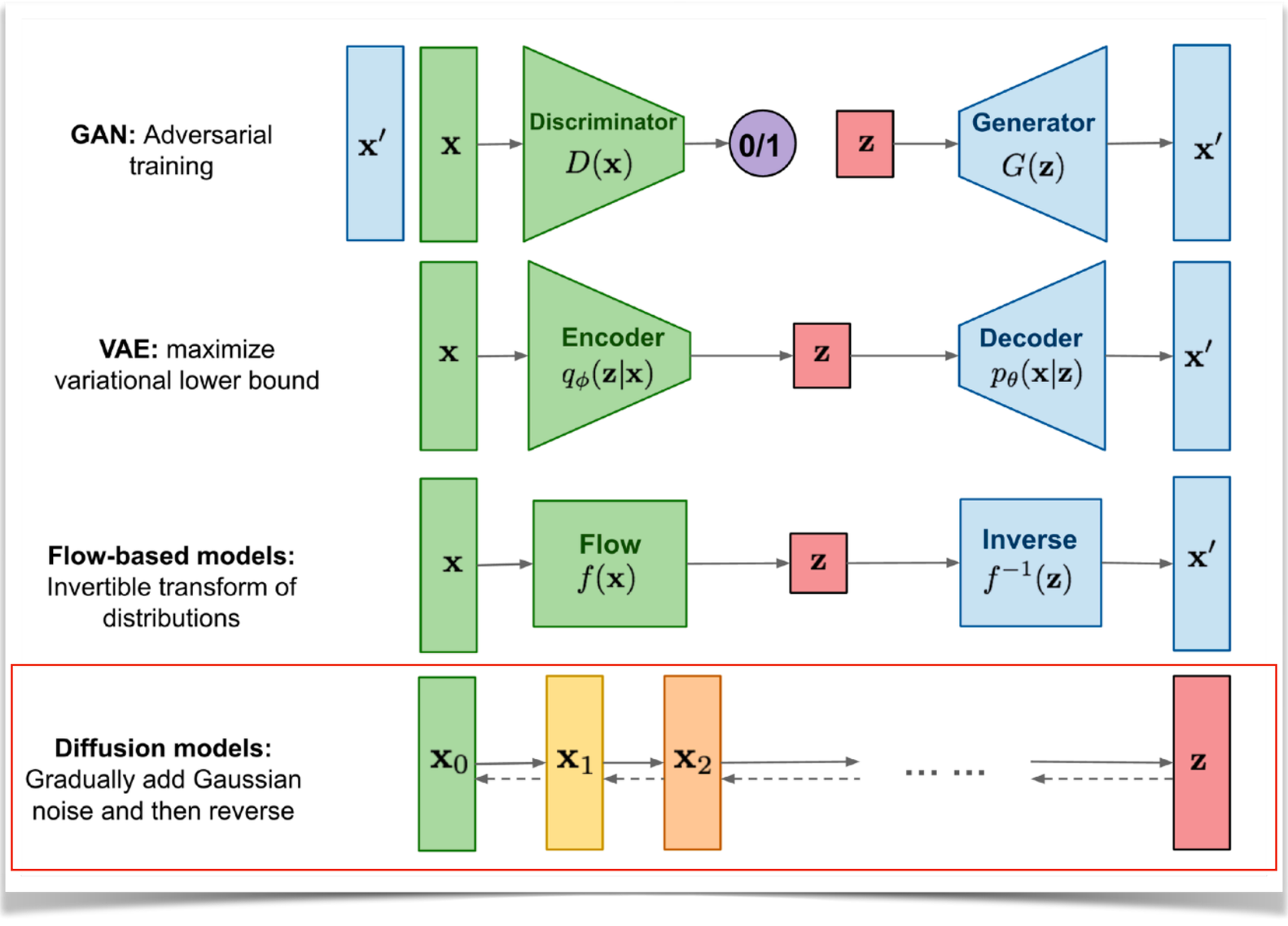
https://labs.openai.com/e/GeLq3027ehOvpTocnBYFkqkM

Diffussion models and generative modeling

https://arxiv.org/pdf/2105.05233.pdf

Diffusion models and generative modeling

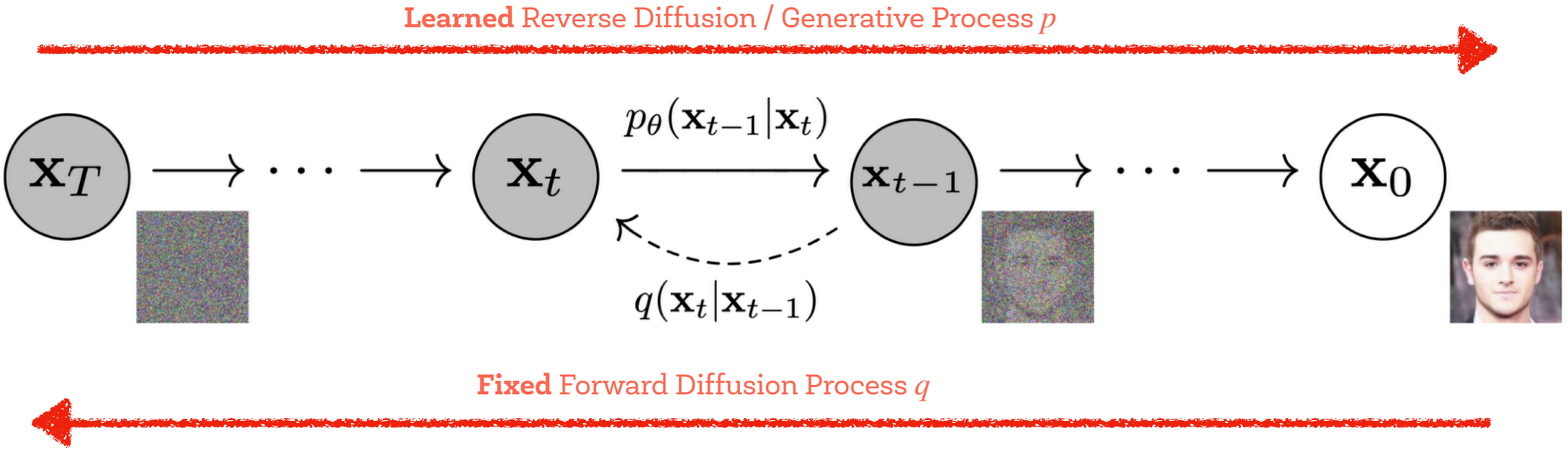
Diffusion Models, Normalizing Flows, GANs or VAEs: they all convert noise from some simple distribution ($\mathbf{z}$) to a data sample ($\mathbf{x}$).



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

Diffusion models and generative modeling

Denoising Diffusion Probabilistic Models are a relatively recent (2020's) addition to generative models.



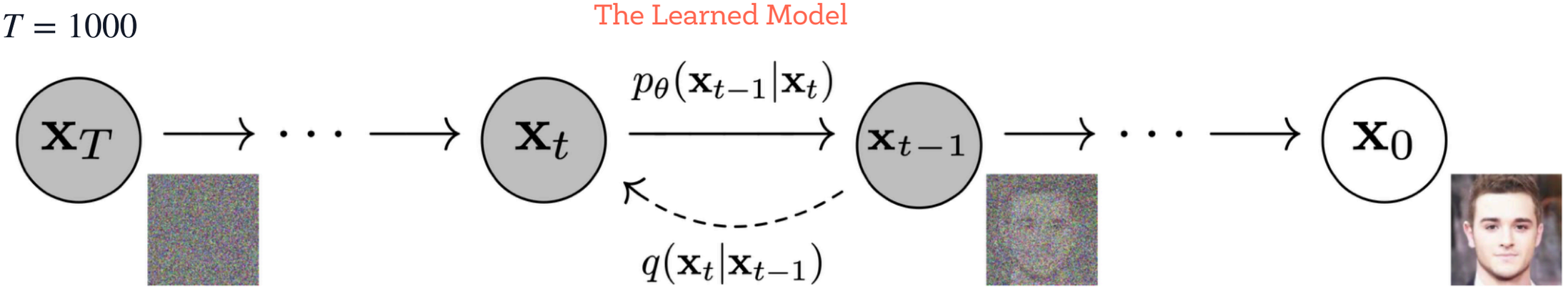
The secret to diffusion models' success is the finite, **iterative** nature of the "diffusion process".
Generation begins with random noise, but this is gradually refined over a number of steps (T) until an output image emerges.

Figure from DDPM paper (https://arxiv.org/abs/2006.11239).

Diffusion models and generative modeling

Data Generation:

The DDPM authors use $T = 1000$



At each step, the model p_θ estimates how we could go from the current input to a completely denoised version.

However, since we only make a **small change at every step**, any errors in this estimate at early stages (where predicting the final output is extremely difficult) can be corrected in later updates.

Figure from DDPM paper (https://arxiv.org/abs/2006.11239).

Diffusion models and generative modeling

Training the model (a neural network p) is relatively straightforward compared to some other types of generative models.

We repeatedly

1. Load in some images from the training data.
2. Add noise, in different amounts.

   Remember, we want the model to do a good job estimating how to 'fix' (denoise) both extremely noisy images and images that are close to perfect.
3. Feed the noisy versions of the inputs into the model.
4. Evaluate how well the model does at denoising these inputs.
5. Use this information to update the model weights.

Diffussion models and generative modeling

To **generate** new images with a trained model, **we begin with a completely random input and repeatedly feed it through the model**, updating it each time by a small amount based on the model prediction.

There are a number of **sampling methods** that try to streamline this process so that we can generate good images with as few steps as possible.

Mathematical basis

A (denoising) diffusion model isn't that complex if you compare it to other generative models such as Normalizing Flows or GANs.

The **sampling** chain transitions in the **forward process** can be set to **conditional Gaussians** when the noise level is sufficiently low.

Combining this fact with the Markov assumption leads to a simple parameterization of the forward process:

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

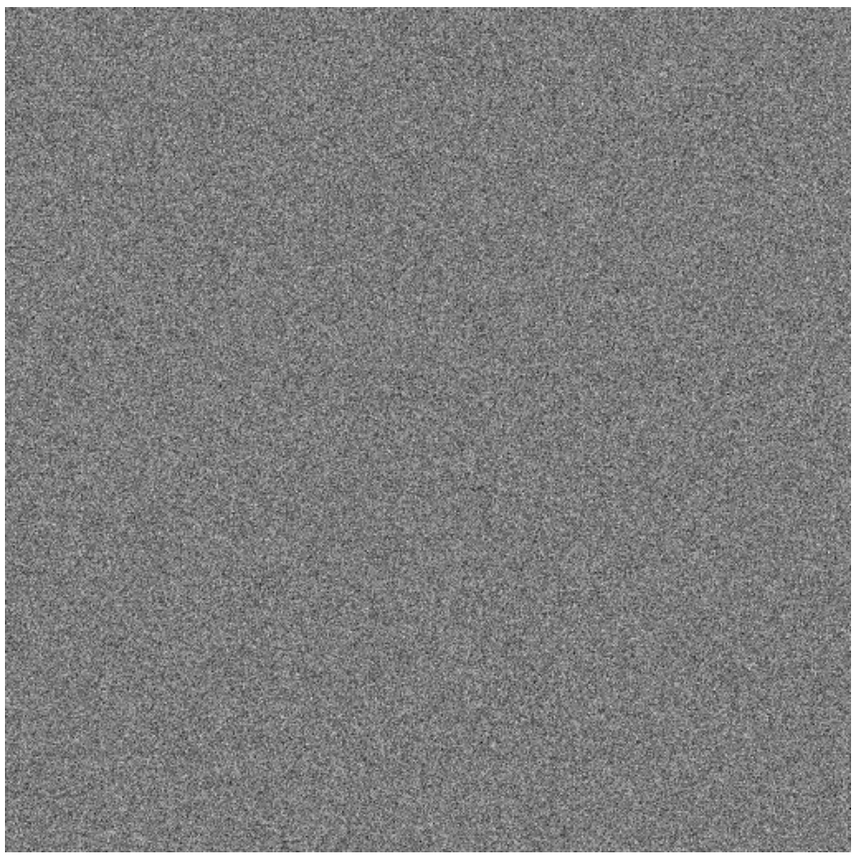
At each step in the chain we are simply sampling from a Gaussian distribution whose mean is the previous value (i.e. image) in the chain.

- β_t aren't constant at each time step t in fact one defines a so-called "**variance schedule**"(a bit like a learning rate schedule).
- We can implement it by sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then setting $\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon$

Mathematical basis

$\beta_1, \ldots, \beta_T$ is a variance schedule which, if well-behaved, ensures that $\mathbf{x}_T$ is **nearly an isotropic Gaussian** for sufficiently large T.

(the DDPM authors use T=1000)



$\mathbf{x}_T$

Mathematical basis

Now, if we knew the conditional distribution $p(x_t - 1 \mid x_t)$, then we could run the process in reverse: by sampling some random Gaussian noise $\mathbf{x}_T$, and then gradually "denoise" it so that we end up with a sample from the real distribution $\mathbf{x}_0$. We're going to leverage a neural network to approximate this conditional probability distribution

During training, the model p_θ learns to reverse this diffusion process in order to generate new data.

Starting with the pure Gaussian noise $p(\mathbf{x}_T) := \mathcal{N}(\mathbf{x}_T, 0, \mathrm{I})$, the model learns the joint distribution $p_\theta(\mathbf{x}_{0:T})$ as

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

The neural network learns the mean and the variance of this backwards process.

where the time-dependent parameters of the Gaussian transitions are learned.

Mathematical basis

A Diffusion Model is **trained** by finding the reverse Markov transitions that maximize the **likelihood of the training data**.

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

Model choices

For the **forward process**, the only choice required is defining the variance schedule, the values of which are generally increasing during the forward process.

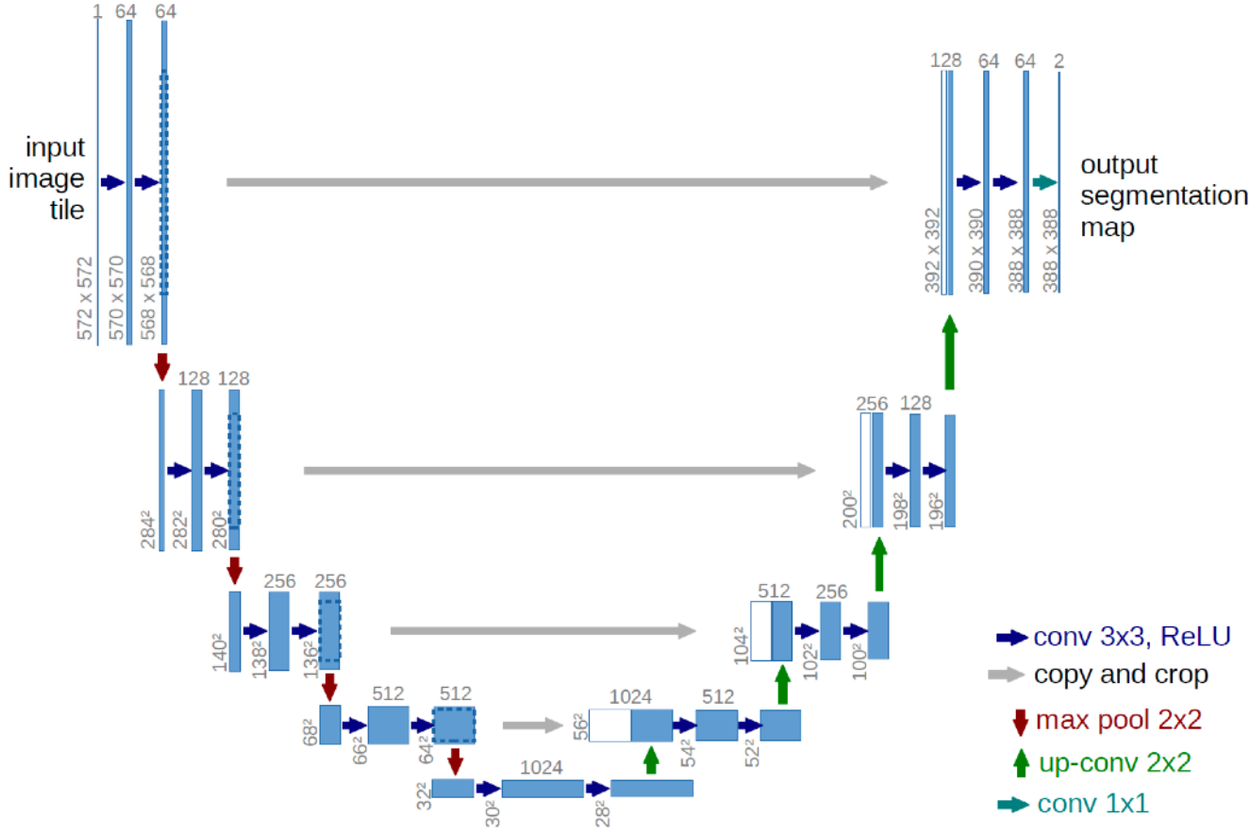
- Regarding the forward process, we can set the variance schedule to be time-dependent constants.
- For example, a **linear** schedule from $\beta_1 = 10^{-4}$ to $\beta_T = 0.2$ might be used.

For the **reverse process**, we must now define the functional forms of μ_θ and $\mathbf{\Sigma}_\theta$ and the network architecture.

Model choices

Regarding the architecture of the model, note that
the only requirement for the model is that its input and output dimensionality are identical.

Given this restriction, it is perhaps unsurprising that image Diffusion Models are commonly implemented with **U-Net-like architectures**.



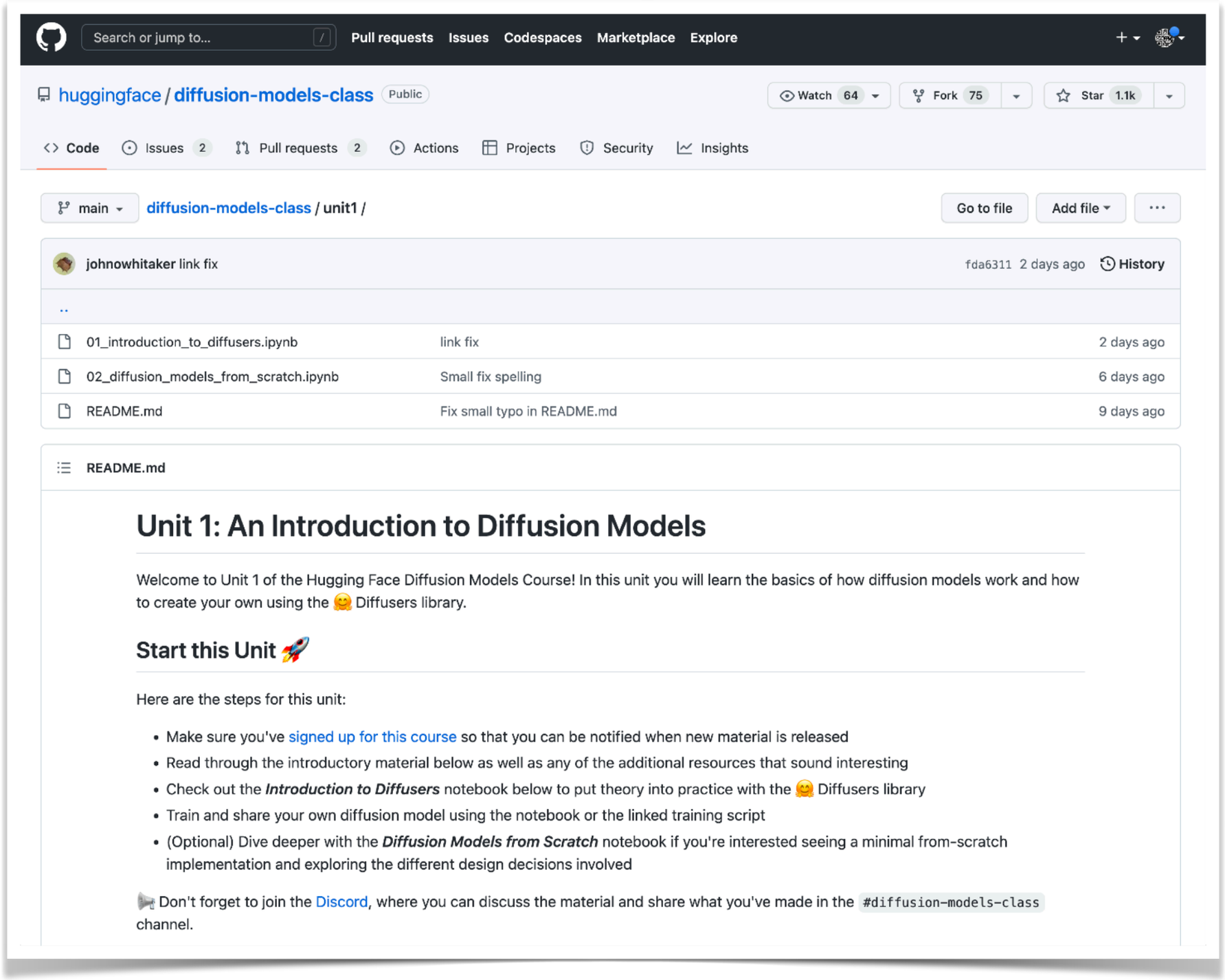
Model choices

Regarding the functional forms of μ_θ and $\mathbf{\Sigma}_\theta$, we simply set:

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

We set these variances to be equivalent to our forward process variance schedule.

Hands-on

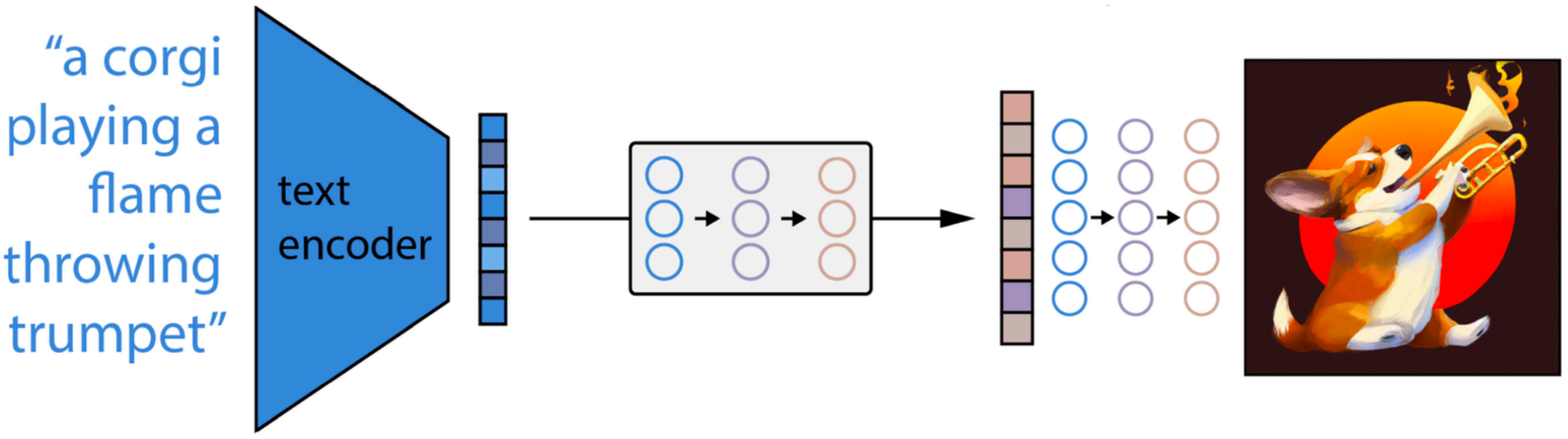


https://github.com/huggingface/diffusion-models-class

DALL-E 2: Creating realistic images and art from a description in natural language.

After inputting **"a teddy bear riding a skateboard in Times Square"**, DALL-E 2 outputs the following image:

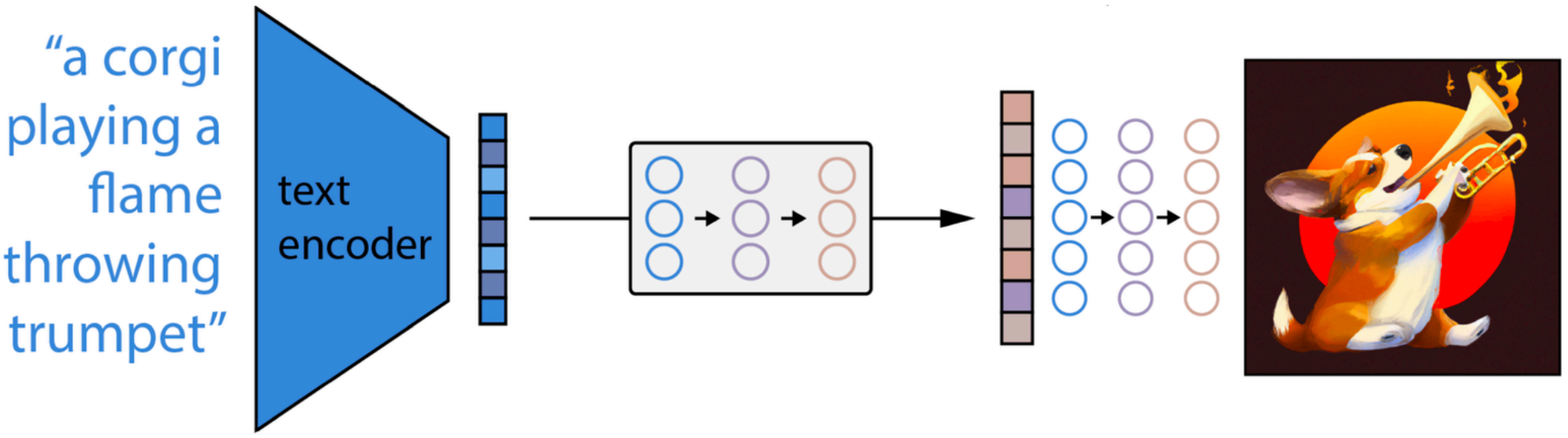
How DALL-E 2 Works



At the highest level, DALL-E 2's works very simply:

- First, a text prompt is input into a text encoder that is trained to map the prompt to a representation space.
- Next, a model called the prior maps the text encoding to a corresponding image encoding that captures the semantic information of the prompt contained in the text encoding.
- Finally, an image decoder stochastically generates an image which is a visual manifestation of this semantic information.

How DALL-E 2 Works

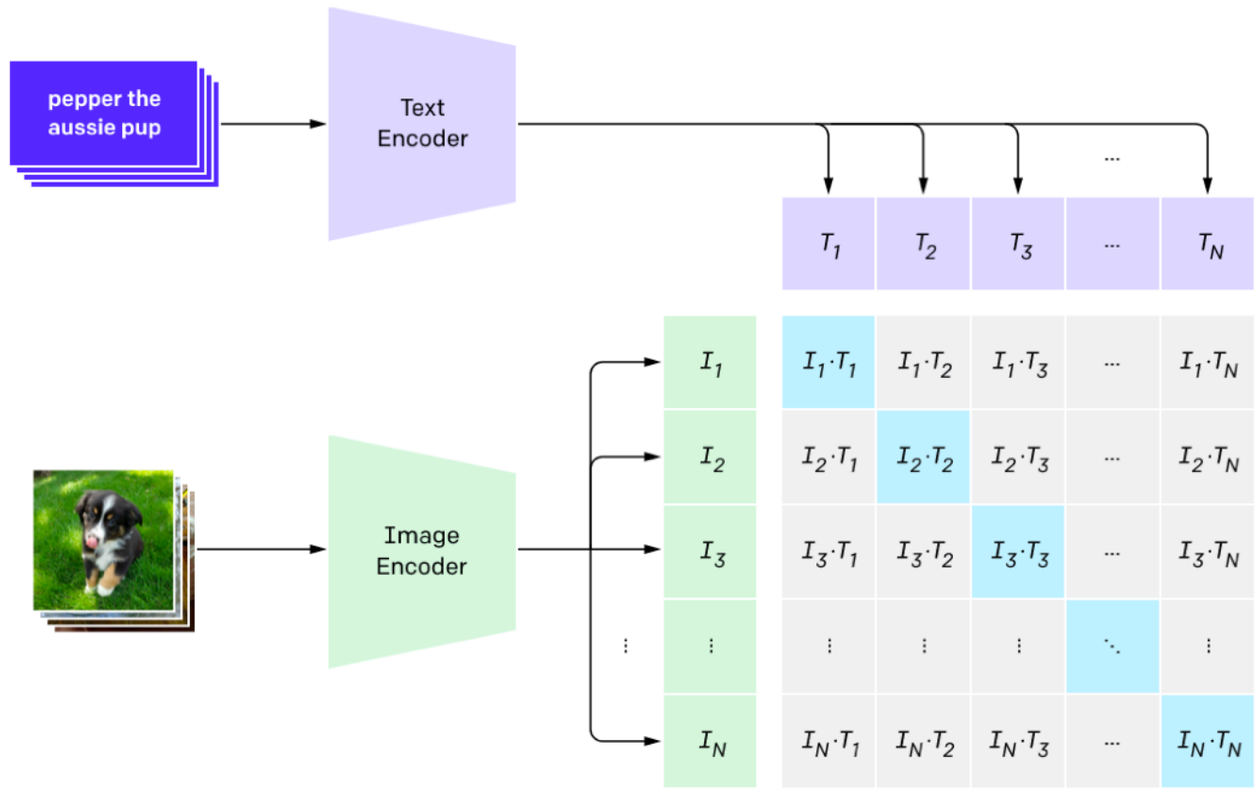


The link between textual semantics and their visual representations in DALL-E is learned by another model called CLIP (Contrastive Language-Image Pre-training).

CLIP is trained on hundreds of millions of images and their associated captions, learning how much a given text snippet relates to an image.

How DALL-E 2 Works

The fundamental principles of training CLIP are quite simple:
First, all images and their associated captions are passed through their respective encoders, mapping all objects into an m-dimensional space.

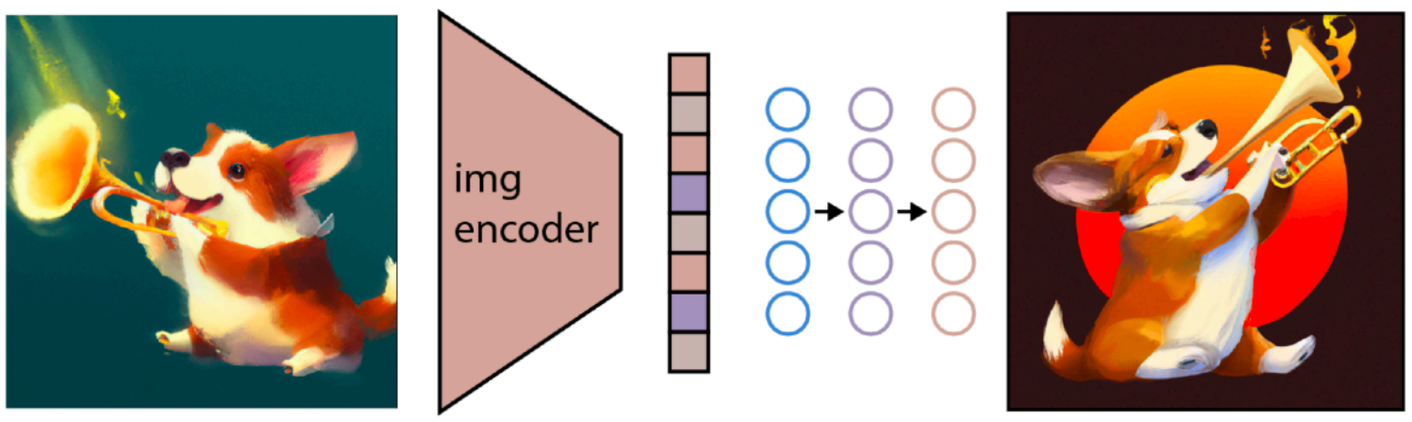


Then, the cosine similarity of each (image, text) pair is computed.
The training objective is to simultaneously maximize the cosine similarity between N correct encoded image/caption pairs and minimize the cosine similarity between $N^2 - N$ incorrect encoded image/caption pairs.

How DALL-E 2 Works

After training, the CLIP model is **frozen** and DALL-E 2 moves onto its next task - learning to reverse the image encoding mapping that CLIP just learned.

In particular, OpenAI employs a modified version of another one of its previous models, GLIDE (a Diffusion Model), to perform this image generation. The GLIDE model learns to invert the image encoding process in order to stochastically decode CLIP image embeddings.



The goal is not to build an autoencoder and exactly reconstruct an image given its embedding, but to instead generate an image which maintains the salient features of the original image given its embedding.

How DALL-E 2 Works

GLIDE extends the core concept of Diffusion Models by augmenting the training process with additional textual information, ultimately resulting in **text-conditional image generation**.