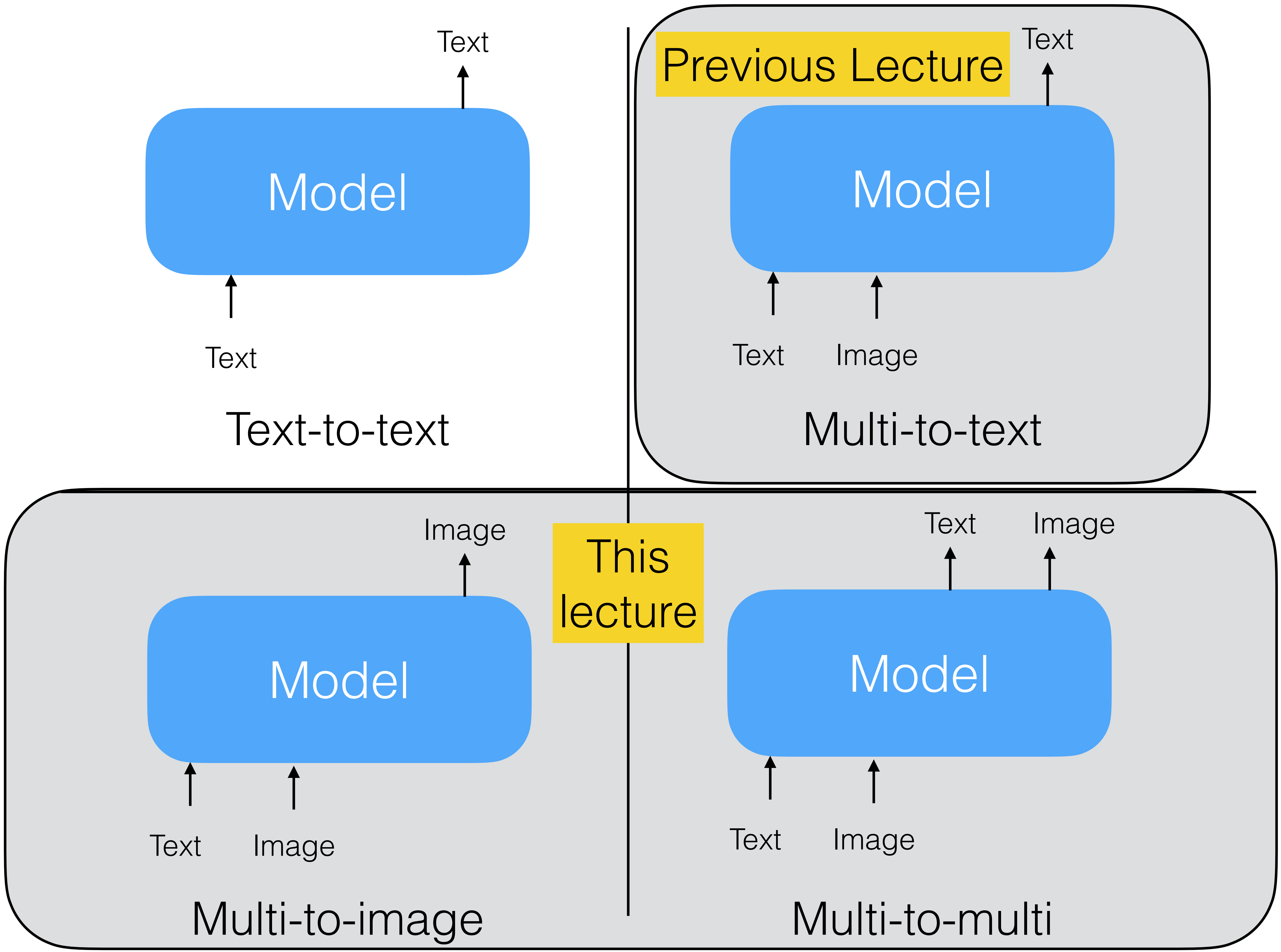
CS11-711 Advanced NLP

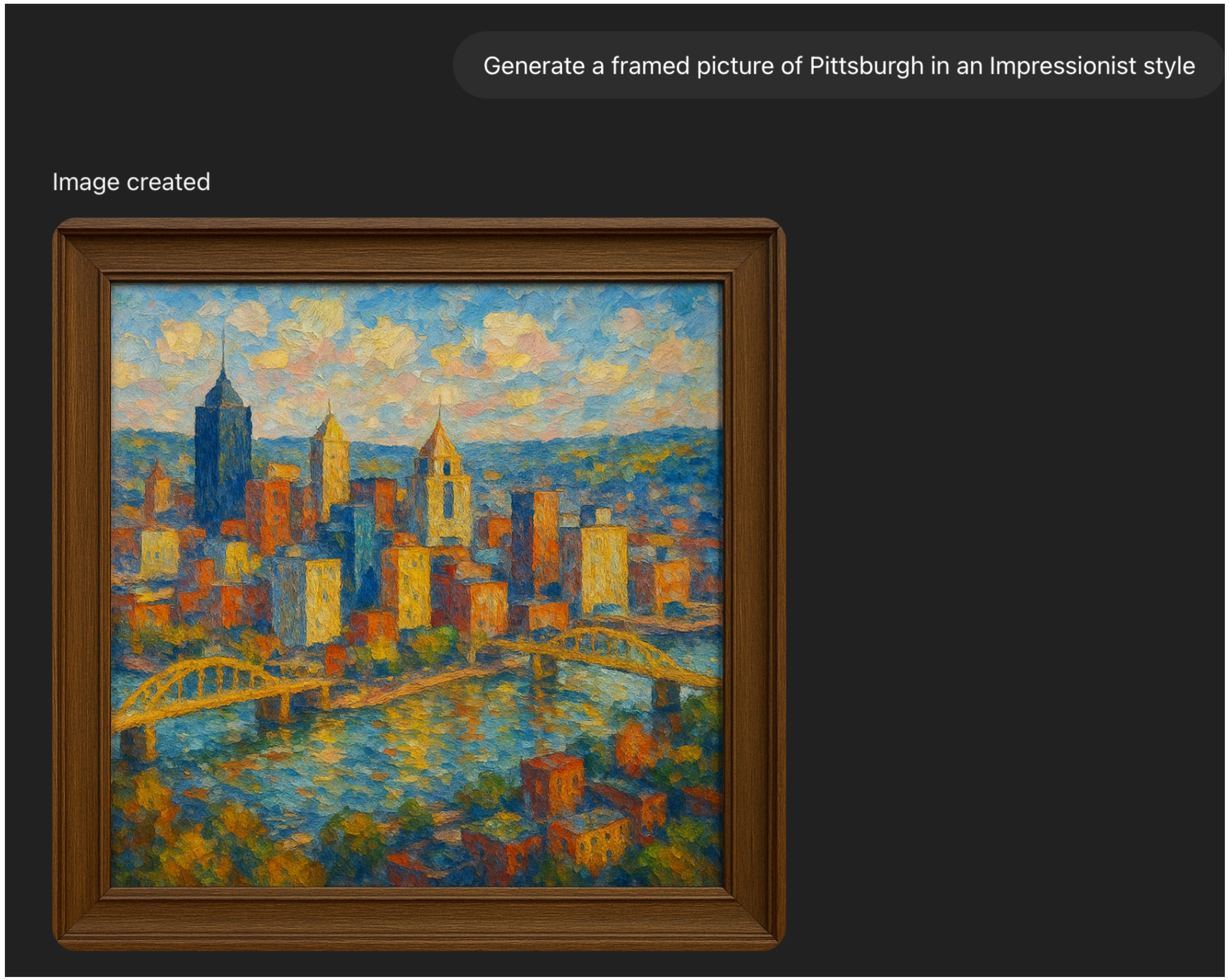
Multimodal Modeling II

Sean Welleck

https://cmu-l3.github.io/anlp-spring2026/

https://github.com/cmu-l3/anlp-spring2026-code

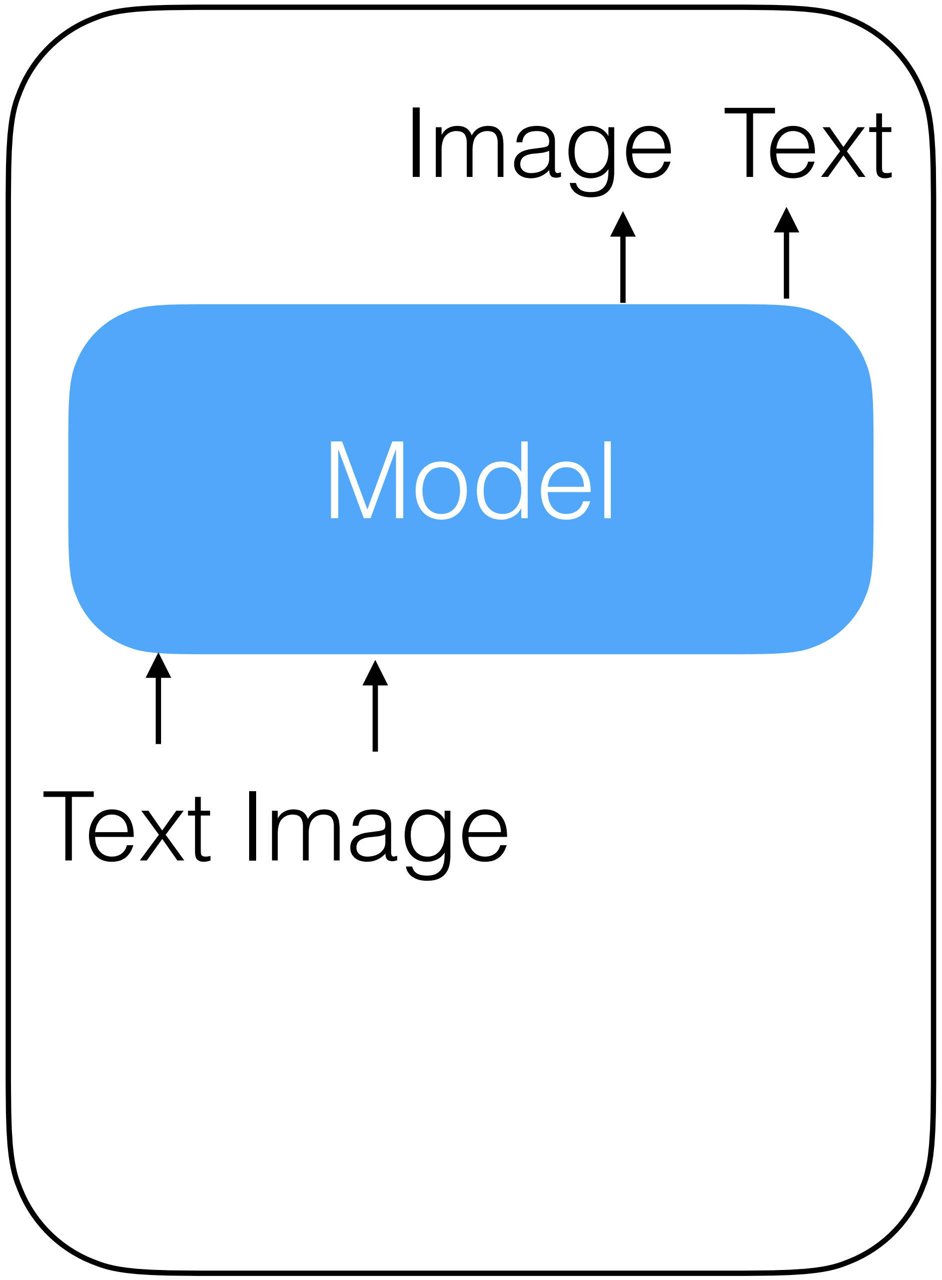

Text
Model
Text
Text-to-text
Previous Lecture
Text
Model
Text
Image
Multi-to-text
This lecture
Image
Model
Text
Image
Multi-to-image
Text
Image
Model
Text
Image
Multi-to-multi



Example: ChatGPT 4o

Today's lecture

- Basic generative modeling paradigms
- Autoregressive modeling of pixels
- Autoregressive modeling of "image tokens"
- Other paradigms

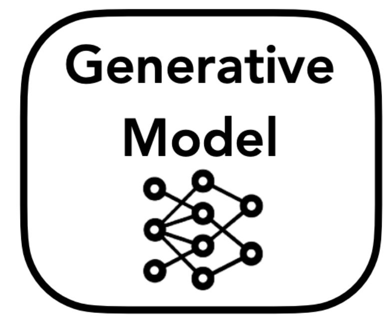


Generative modeling paradigms

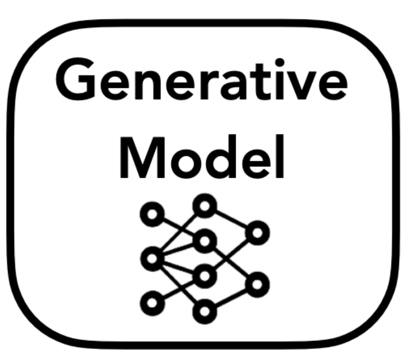
$p_*(x)$

Train

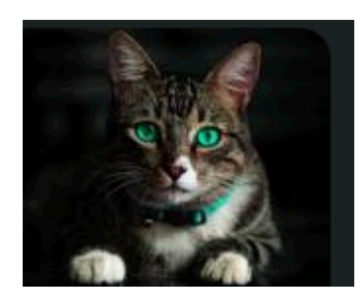
$p_\theta(x)$



$p_\theta(x)$

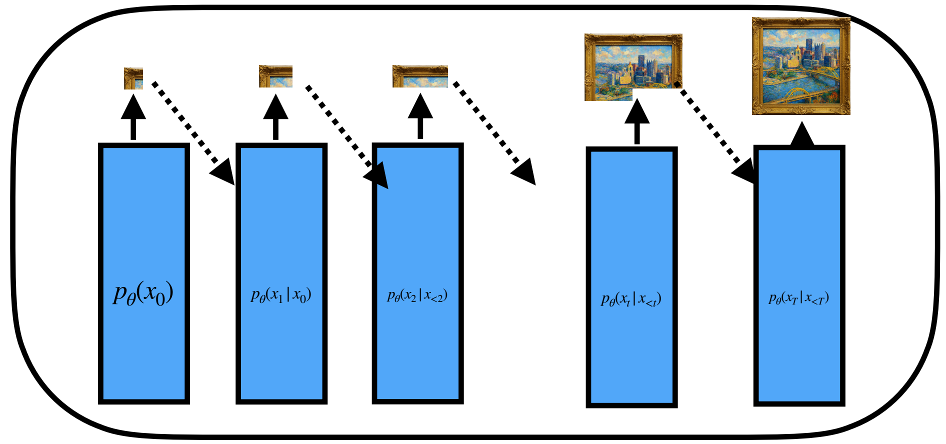


Sample

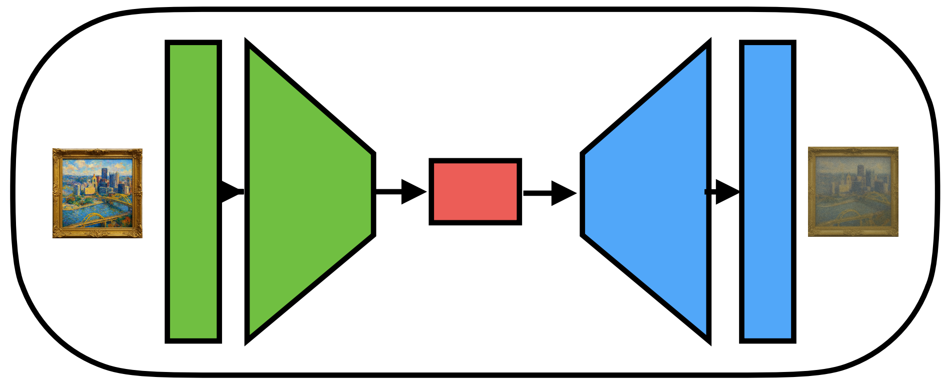


Generative modeling paradigms

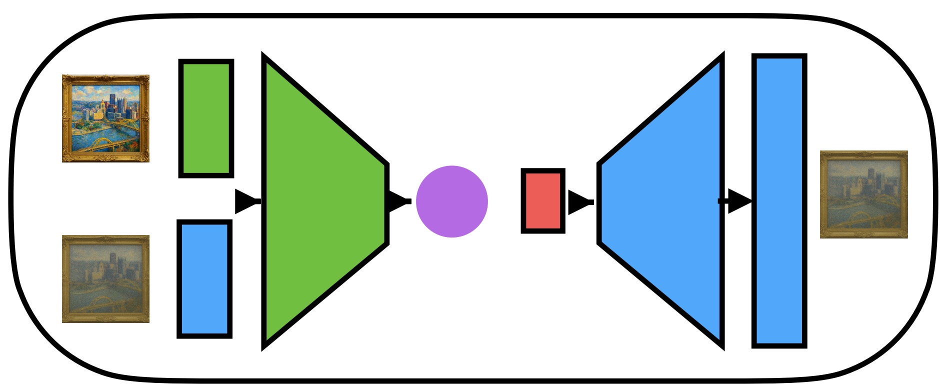
Autoregressive



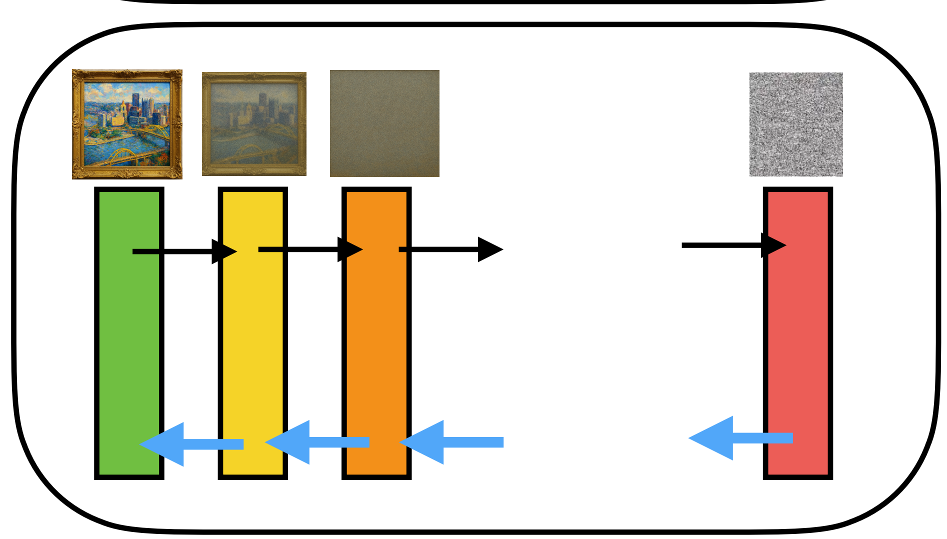
Variational Autoencoder (VAE)



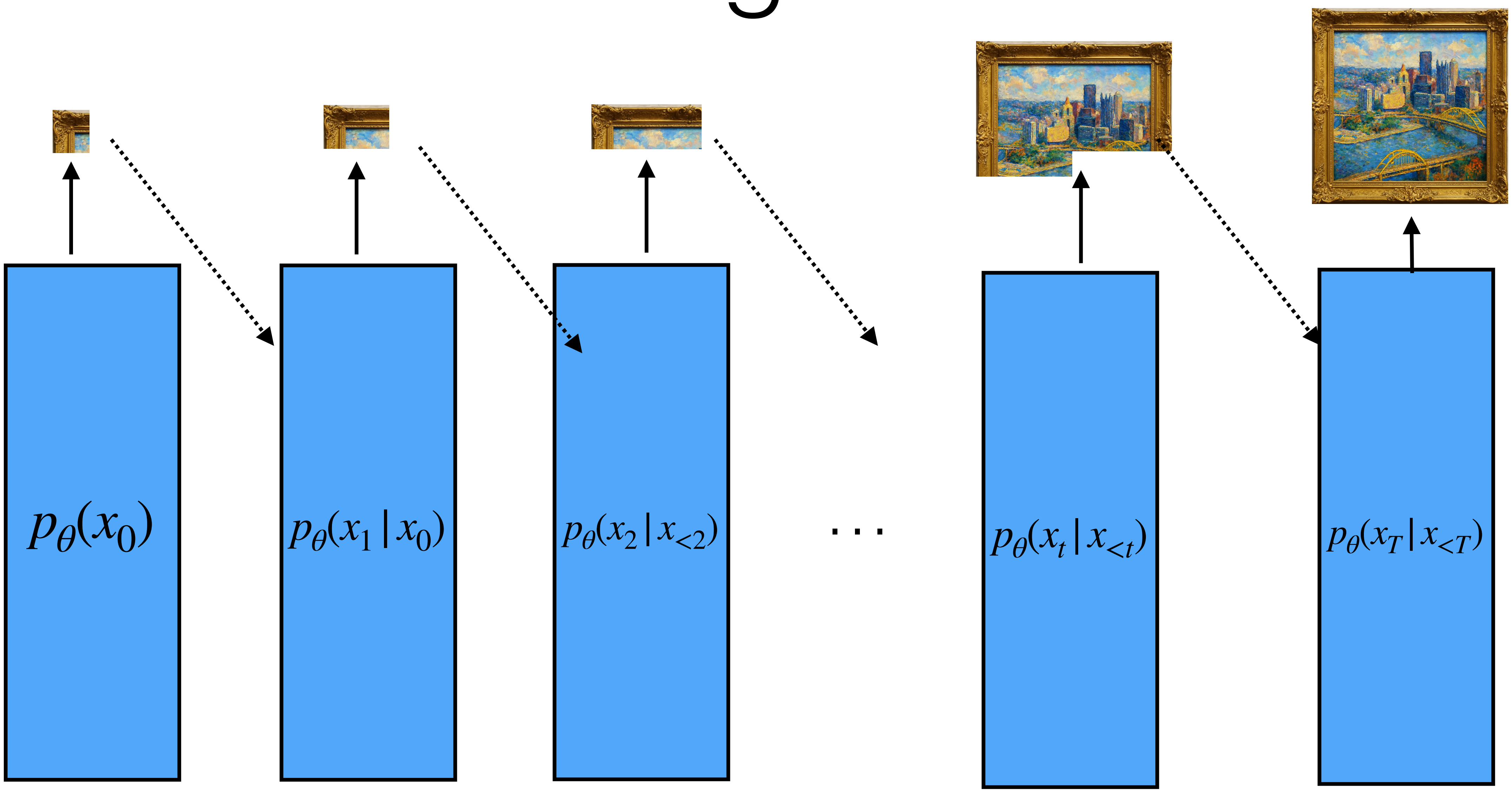
Generative Adversarial Networks (GAN)



Diffusion

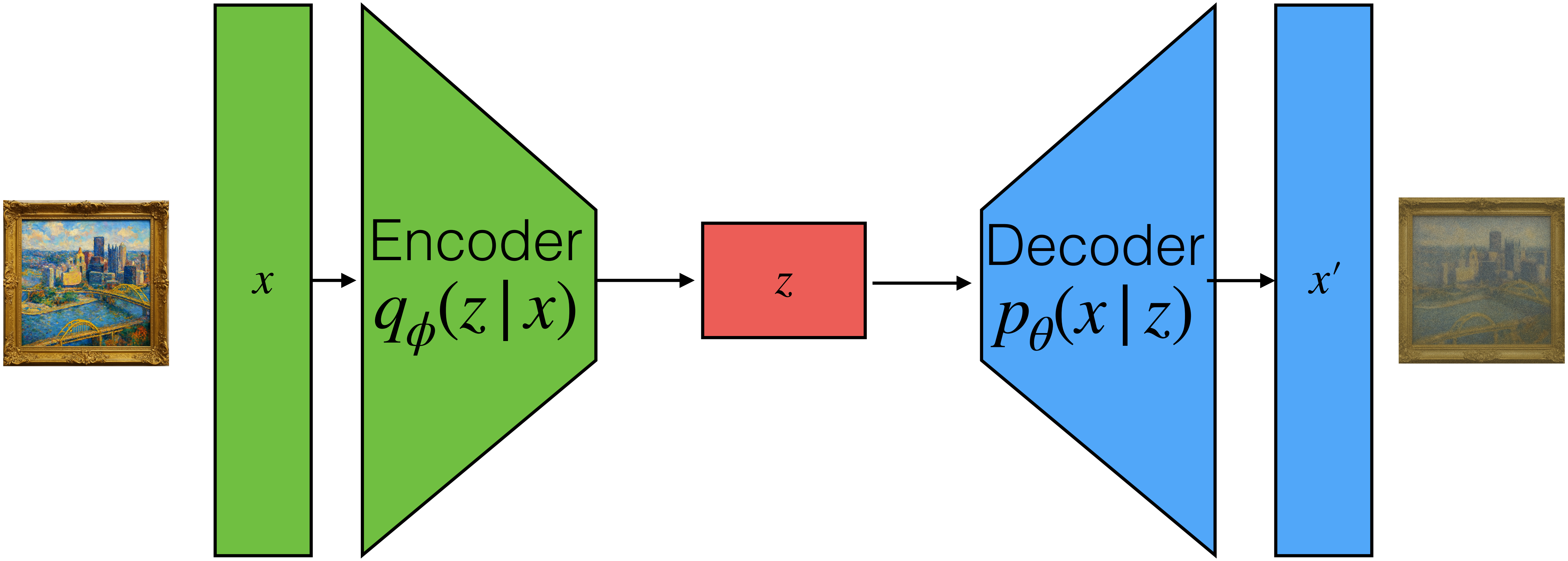


Autoregressive



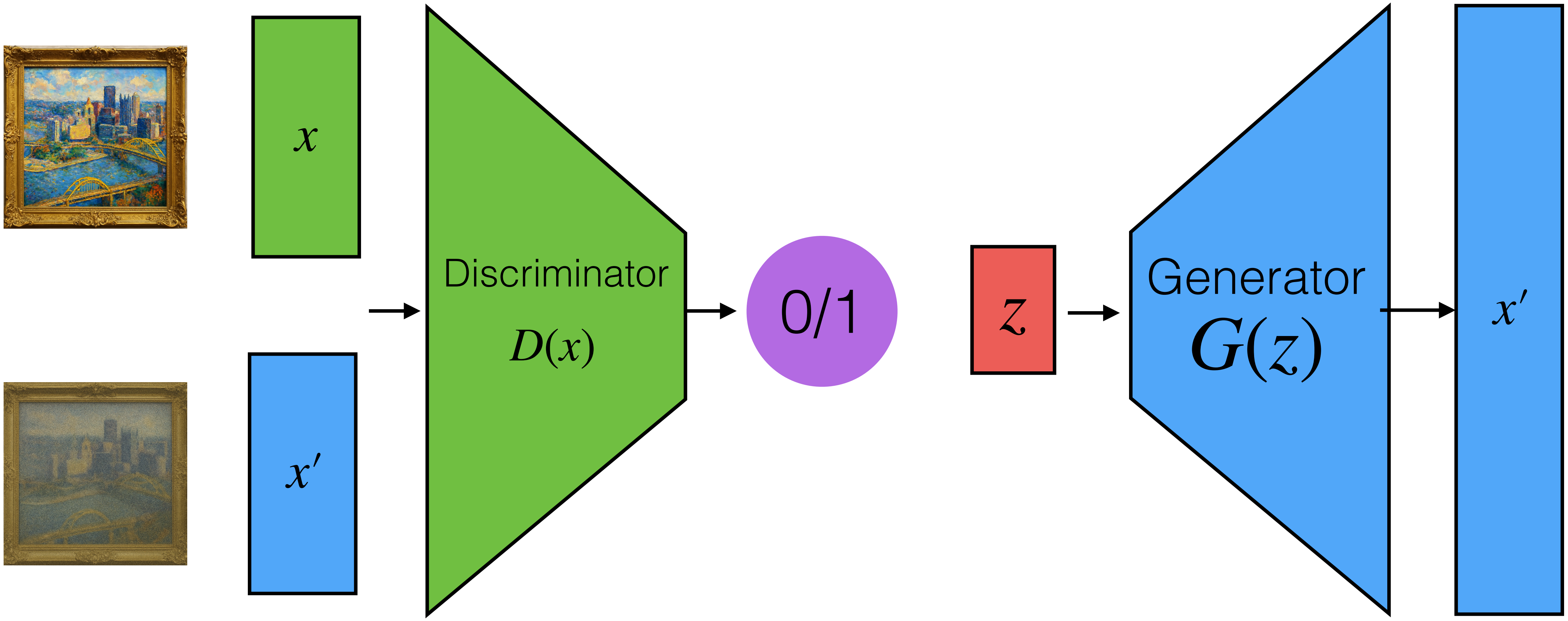
- Generate one dimension (e.g., token, pixel) at a time given the previous ones

Variational Auto-encoders (VAE)



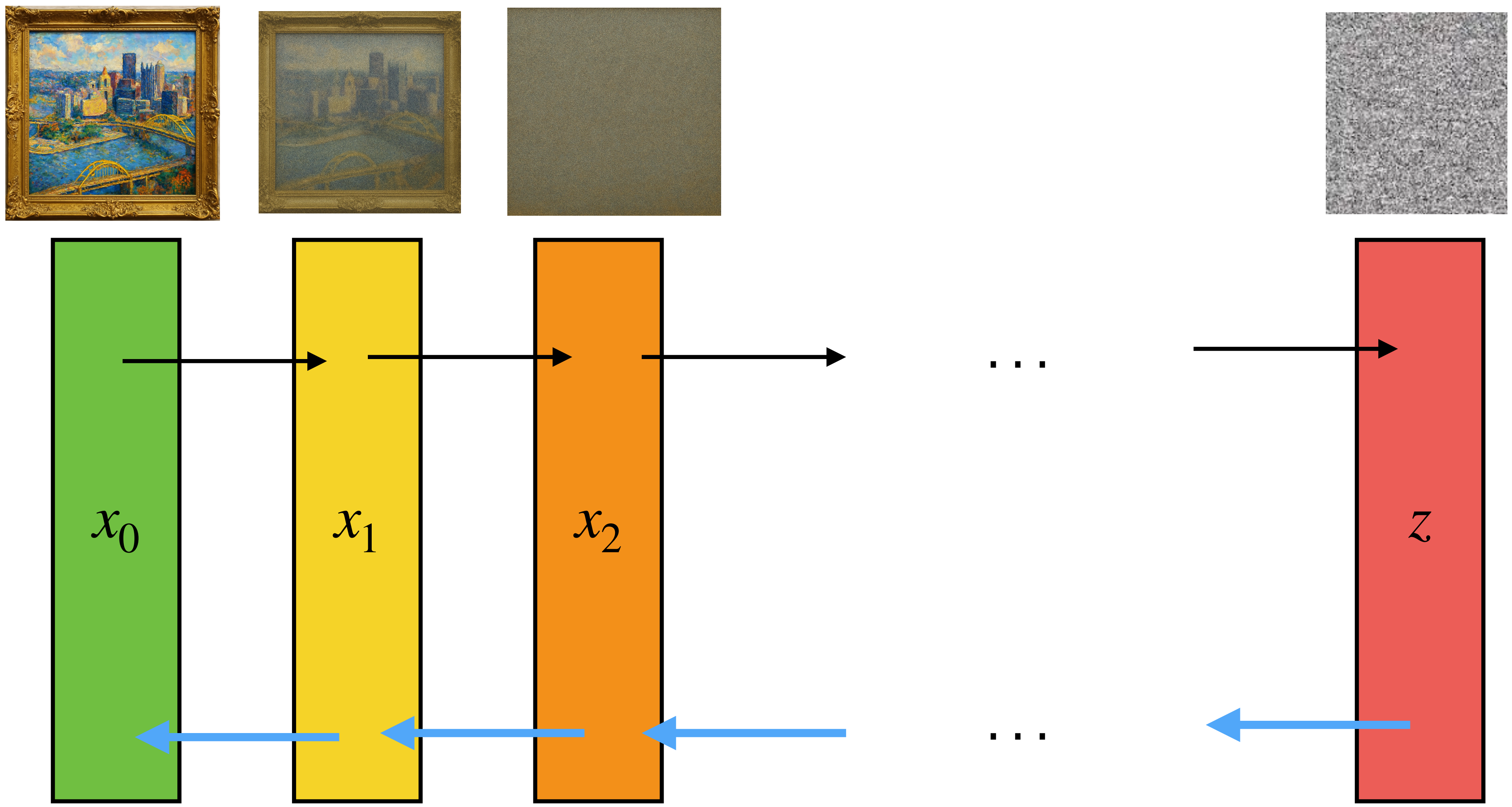
- Learn to reconstruct the image by compressing it into a vector and uncompressing it

Generative adversarial networks (GAN)



- Learn to generate an image by fooling a discriminator that detects whether an image is fake

Diffusion models



- Gradually add noise to an image, then learn to de-noise

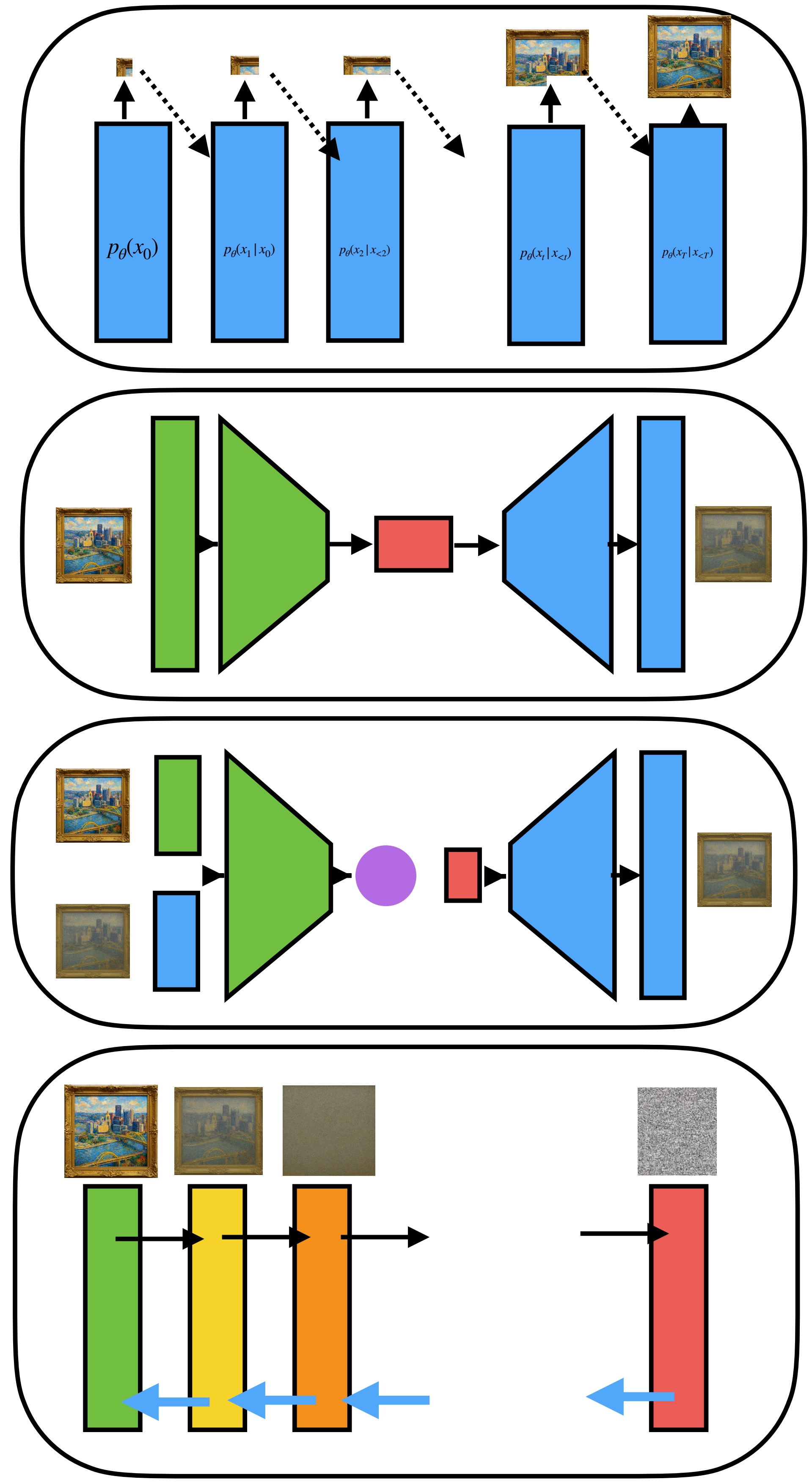
Generative modeling paradigms

Autoregressive

VAE

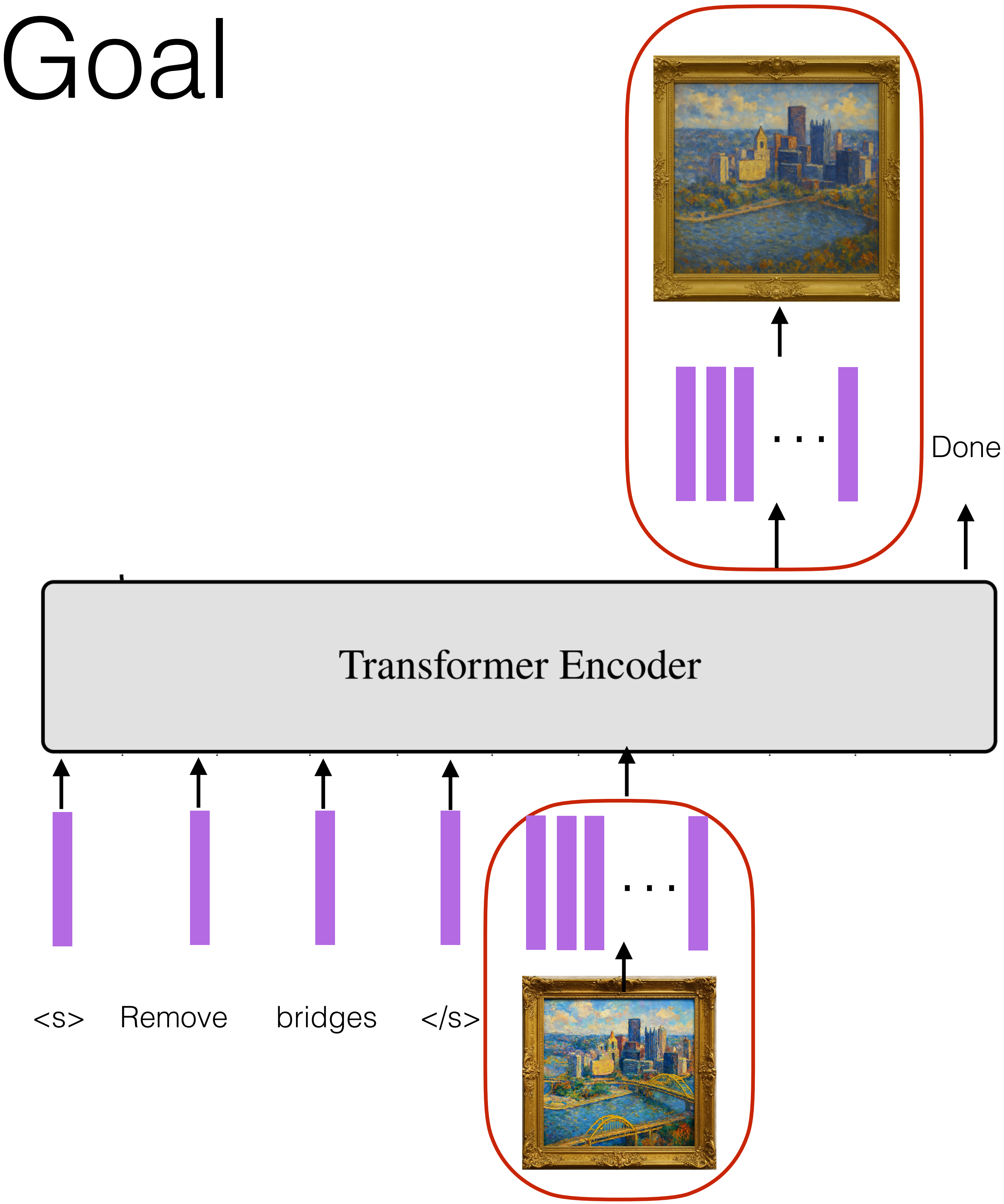
GAN

Diffusion



Goal

- Image tokenizer
- Image “de-tokenizer”
- Single Transformer for text and image



Roadmap

- **Attempt 1**: use a standard auto-regressive model
  - Tokenizer: individual pixels
  - De-tokenizer: generate individual pixels, no de-tokenizer needed
- **Attempt 2**: learn a discrete tokenizer / de-tokenizer
  - Vector-quantized VAE (VQ-VAE)
  - VQ-GAN

Auto-regressive model

- **Basic idea**: treat an image as a sequence of pixels
  - Learn a language model over the pixel sequences
- $x_{\text{img}} \in \mathbb{R}^{H \times W \times C} \rightarrow x_1, \ldots, x_T$
  - $x_t \in \{1, 2, \ldots, 256\}$
- Given a dataset of sequences:
  - $\mathscr{L}_{MLE} = \sum_{t=1}^{T} -\log p_\theta(x_t \mid x_{<t})$

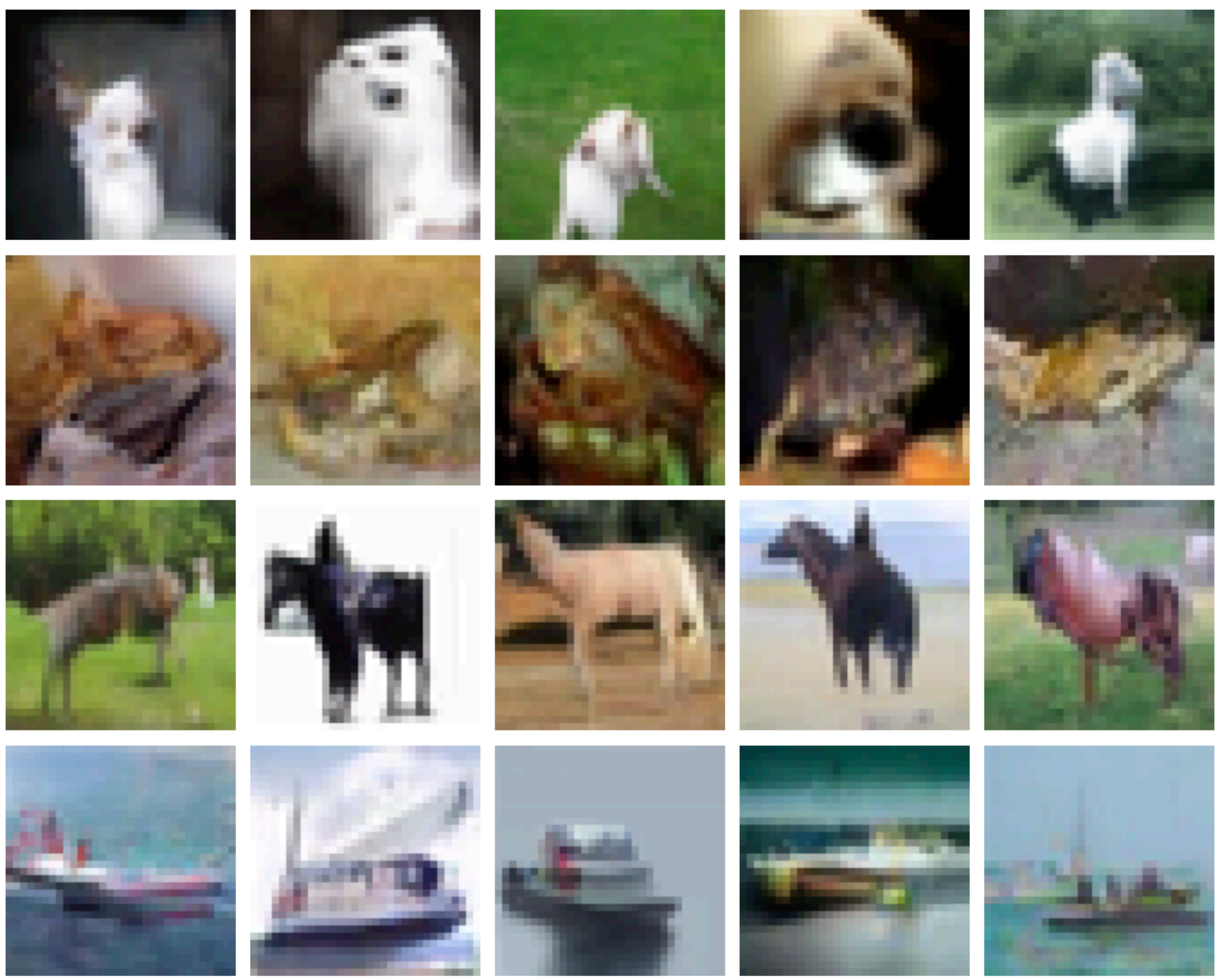
Example: PixelRNN

[van den Oord 2016]

Figure 1. Image completions sampled from a PixelRNN.

Example: Image Transformer

[Parmar et al 2018]



Example: iGPT

[OpenAI et al 2020]

Generative Pretraining from Pixels

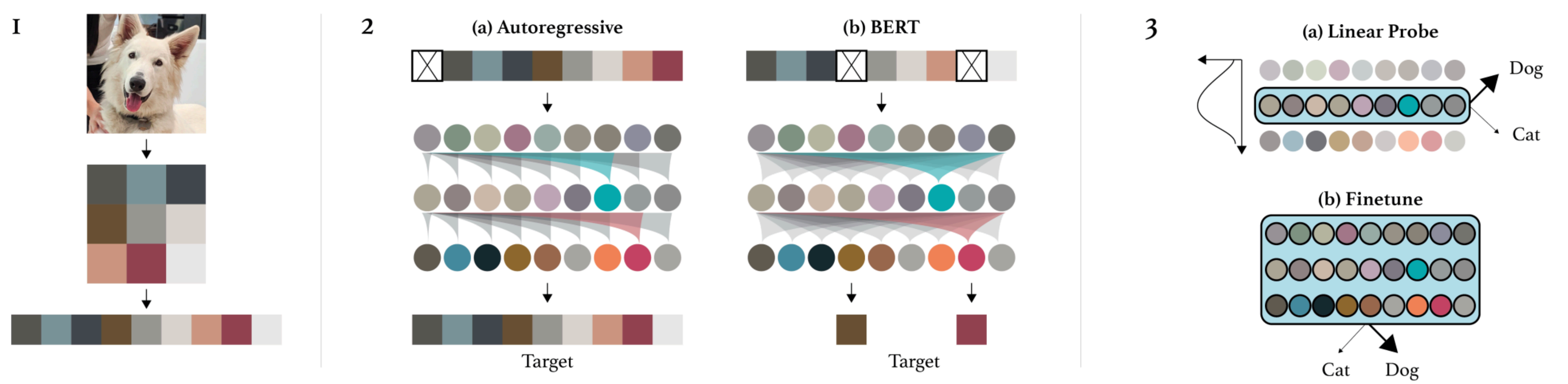
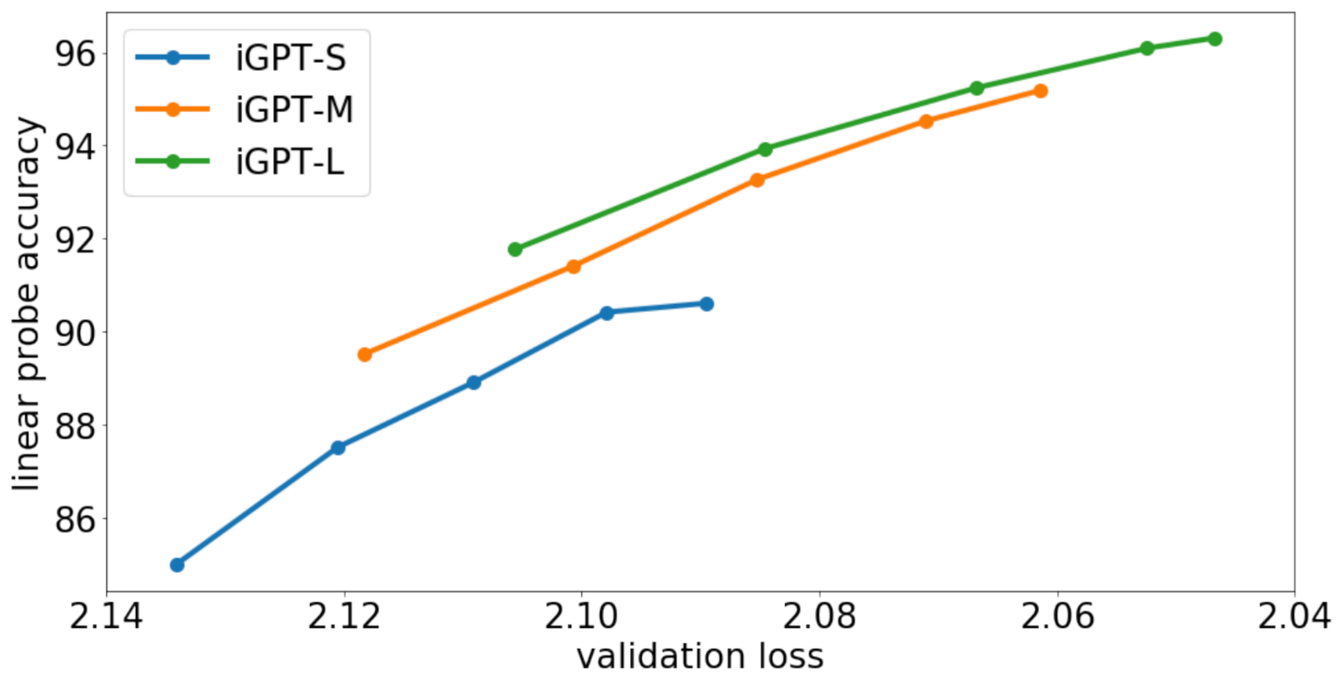
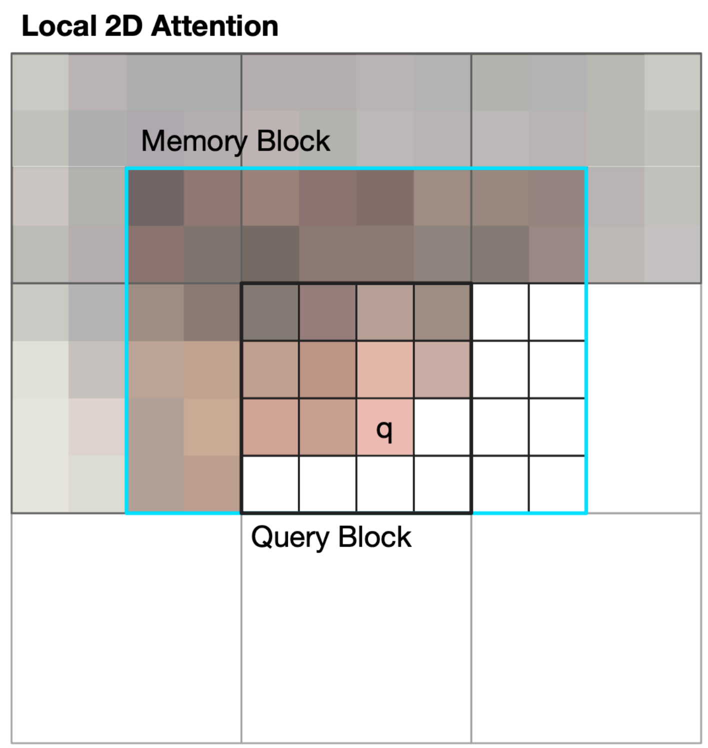


Figure 1. An overview of our approach. First, we pre-process raw images by resizing to a low resolution and reshaping into a 1D sequence. We then chose one of two pre-training objectives, auto-regressive next pixel prediction or masked pixel prediction. Finally, we evaluate the representations learned by these objectives with linear probes or fine-tuning.



Auto-regressive model

- **Challenge 1**: sequence length
  - 1024 x 1024 image with 3 channels (RGB)
    - 3 million tokens
- **Challenge 2**: each pixel carries relatively little semantic information compared to a word token
  - => lots of data needed



Typically involved alternative memory-efficient attention patterns

Auto-regressive model scaling

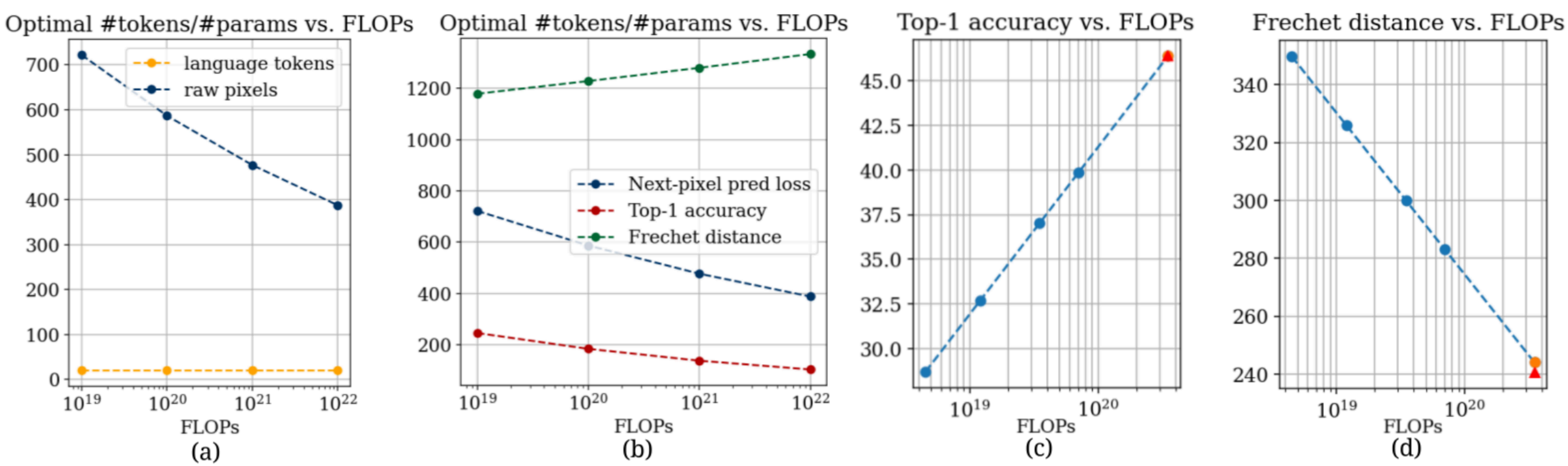


Figure 1. Key findings on the scaling properties of next-pixel prediction, based on training Transformers on 32 × 32 images. (a) Learning on raw pixels (blue line) requires $10 - 20\times$ higher optimal token-to-parameter ratio than learning on language tokens (yellow line). (b) The optimal scaling strategy varies: generation quality (Fréchet Distance, green) requires more training data optimally than classification (Top-1 accuracy, red) or the next-pixel prediction loss (blue). (c)-(d) The optimal-token and optimal-parameter setup is further verified by training models following the scaling prediction. Given 3.5e20 training FLOPs (with 5 × more compute), we project to reach 46.39% accuracy (vs. 46.41% in reality) and 244 Fréchet Distance (vs. 240 Fréchet Distance in reality).

Rethinking generative image pretraining:
How far are we from scaling up next-pixel prediction?

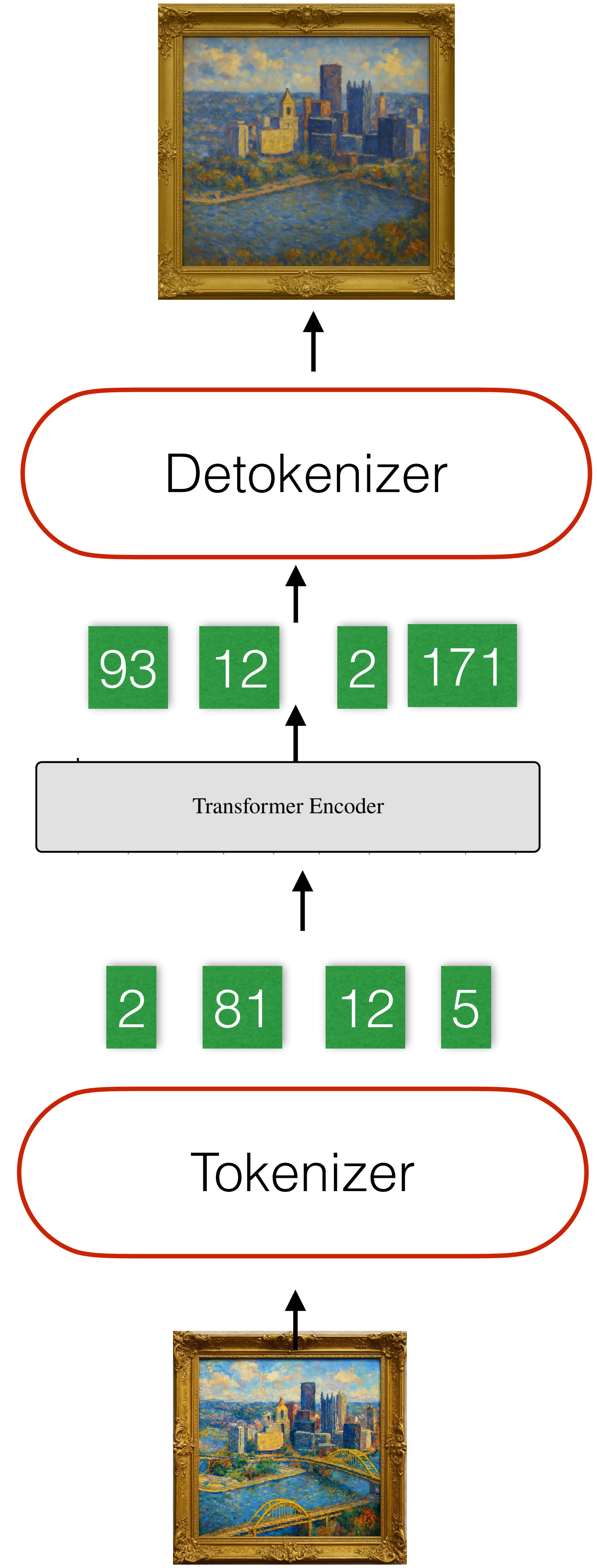
Xinchen Yan [1*] **Chen Liang** [1*] **Lijun Yu** [1] **Adams Wei Yu** [1] **Yifeng Lu** [1] **Quoc V. Le** [1]

Roadmap

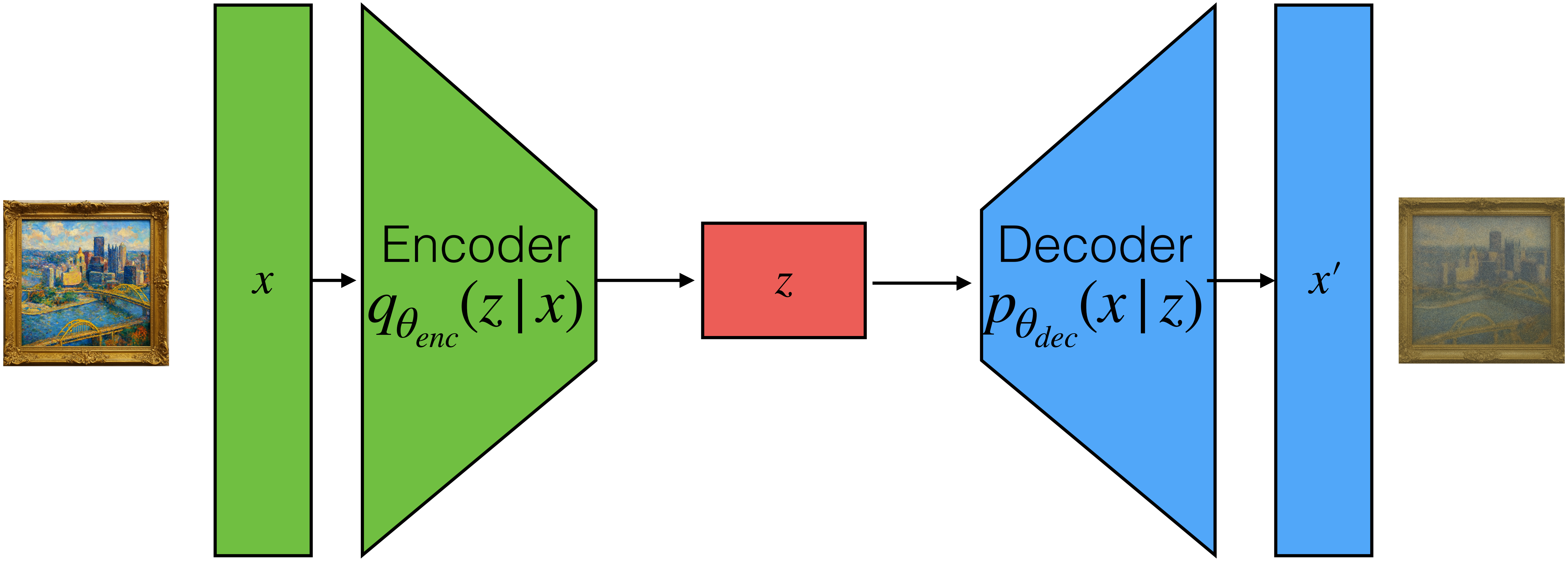
- Attempt 1: use a standard auto-regressive model
- **Attempt 2**: learn a discrete tokenizer / de-tokenizer
  - Vector-quantized VAE (VQ-VAE)
  - VQ-GAN

Idea

- Learn a tokenizer that compactly represents an image as a sequence of discrete tokens.
- Example:
  - Vocabulary: 8,000 token ids
  - Image: sequence of 1024 tokens
- Approach: VQ-VAE

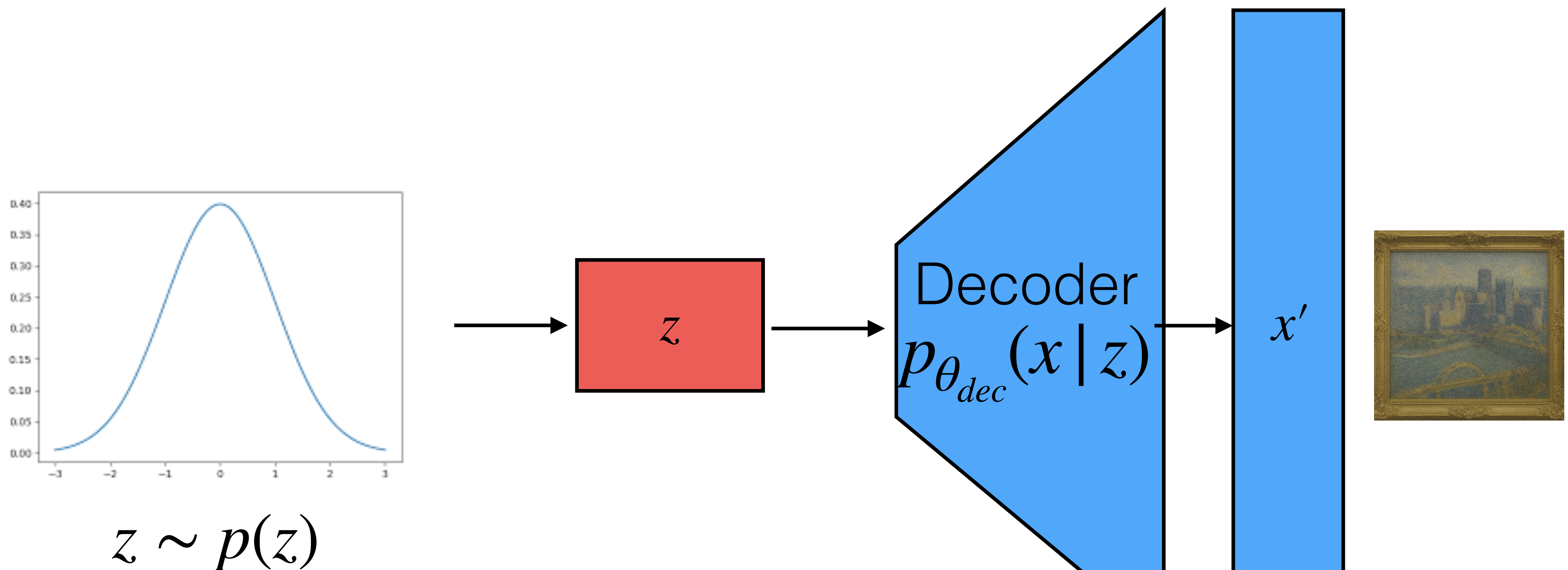


Variational Autoencoder (VAE)



- Learn to reconstruct the image by compressing it into a vector and uncompressing it

Variational Autoencoder (VAE)



- After training, generate by sampling a latent vector from the prior and passing it to the decoder

Standard VAE

[Kingma & Welling 2013]

- **Goal**: learn to compress data (like images) into a low-dimensional representation z, and then reconstruct or generate new data with it
- Assume that data is generated as follows:
  - First sample a latent variable $z \sim p(z)$
  - Turn it into data x with a decoder, $p_{\theta_{dec}}(x \mid z)$
- The encoder maps data x into a latent variable, $q_{\theta_{enc}}(z \mid x)$

Standard VAE

[Kingma & Welling 2013]

- Train by balancing two goals:
  - **Reconstruct the input well**:

    Reconstruction loss: $-\log p_{\theta_{dec}}(x \mid z)$

  - **Keep z values well behaved**:

    Regularization: $D_{KL}\left(q_{\theta_{enc}}(z \mid x) \| \mathcal{N}(0,I)\right)$

Makes the encoder's output distribution close to a standard Gaussian

Standard VAE

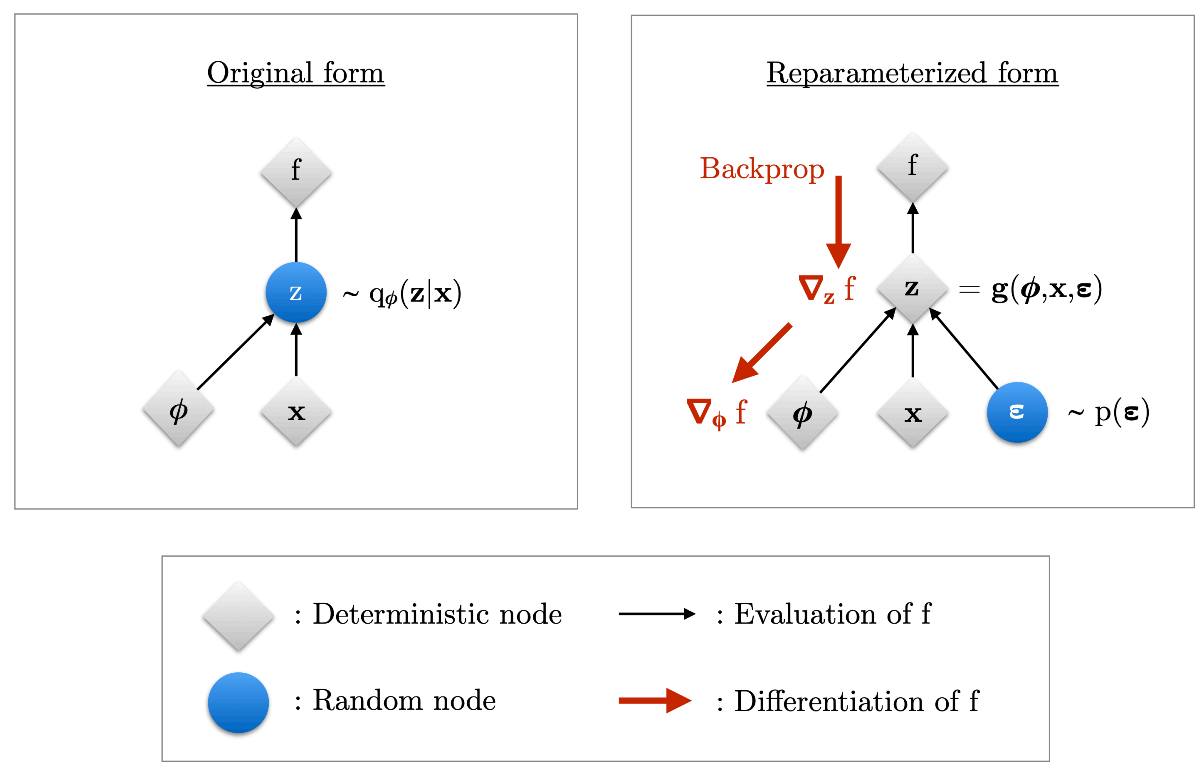
[Kingma & Welling 2013]

- Final loss:

$$\mathscr{L}_{VAE}(x) = -\mathbb{E}_{q_{\theta_{enc}}(z|x)}\left[\log p_{\theta_{dec}}(x\,|\,z)\right] + D_{KL}(q_{\theta_{enc}}(z\,|\,x)||p(z))$$

where $p(z)$ is $\mathscr{N}(0,I)$

"Reparameterization trick"



VAE Loss: Evidence Lower Bound

- $\log p(x) = \log \int p(x,z)dz$

$$= \log \int q(z\,|\,x)\frac{p(x,z)}{q(z\,|\,x)}dz$$

$$= \log \mathbb{E}_{q(z|x)}\left[\frac{p(x,z)}{q(z\,|\,x)}\right]$$

$$\geq \mathbb{E}_{q(z|x)}\left[\log \frac{p(x,z)}{q(z\,|\,x)}\right]$$

$$= \mathbb{E}_{q(z|x)}\left[\log p(x\,|\,z)\right] + \mathbb{E}_{q(z|x)}\left[\log \frac{p(z)}{q(z\,|\,x)}\right]$$

$$= \mathbb{E}_{q(z|x)}\left[\log p(x\,|\,z)\right] - \mathrm{D_{KL}}(q(z\,|\,x)\|p(z))$$

$$= -\mathscr{L}_{VAE}$$

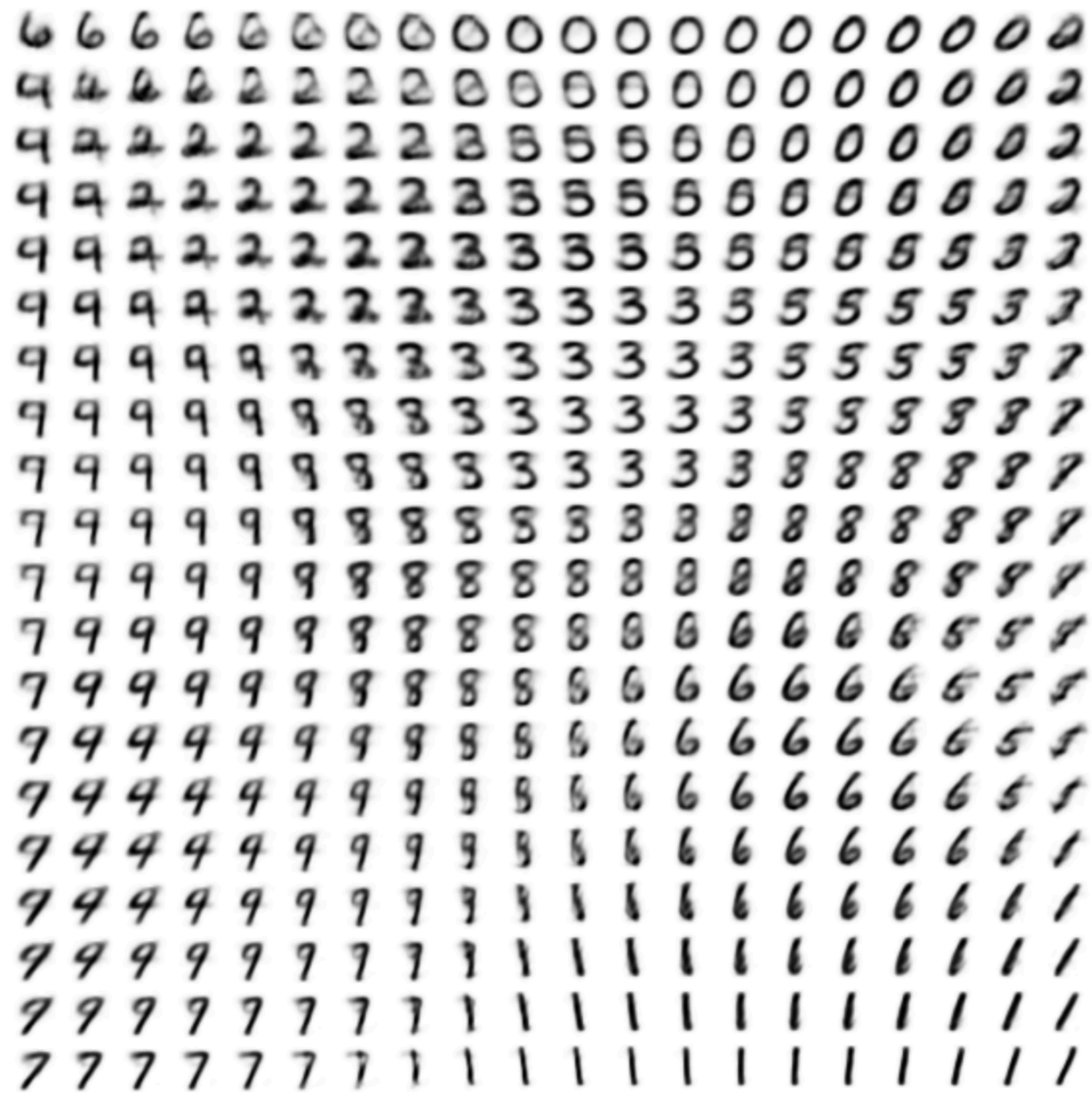
Importance sampling

Jensen's inequality

Evidence **L**ower **Bo**und ("ELBO")

Standard VAE

(a) Learned Frey Face manifold

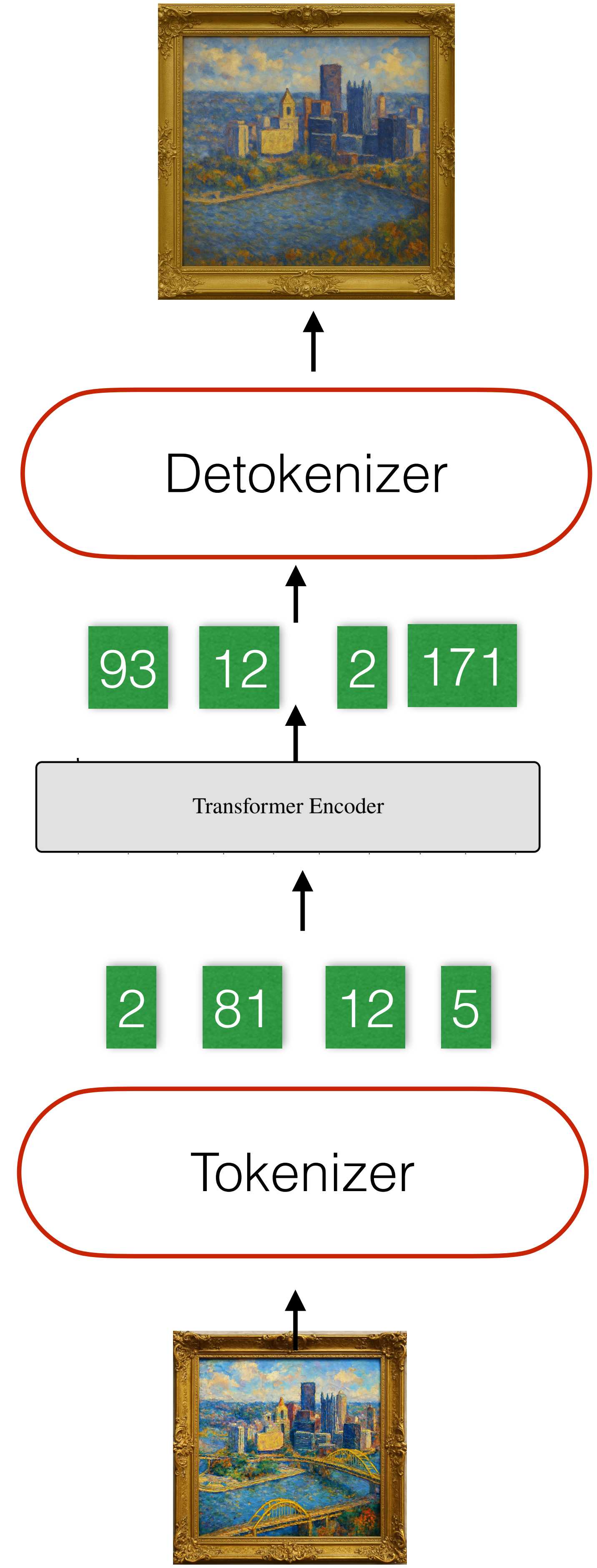


(b) Learned MNIST manifold

From continuous to discrete

- Standard VAEs give us a way to encode and decode images using a continuous vector

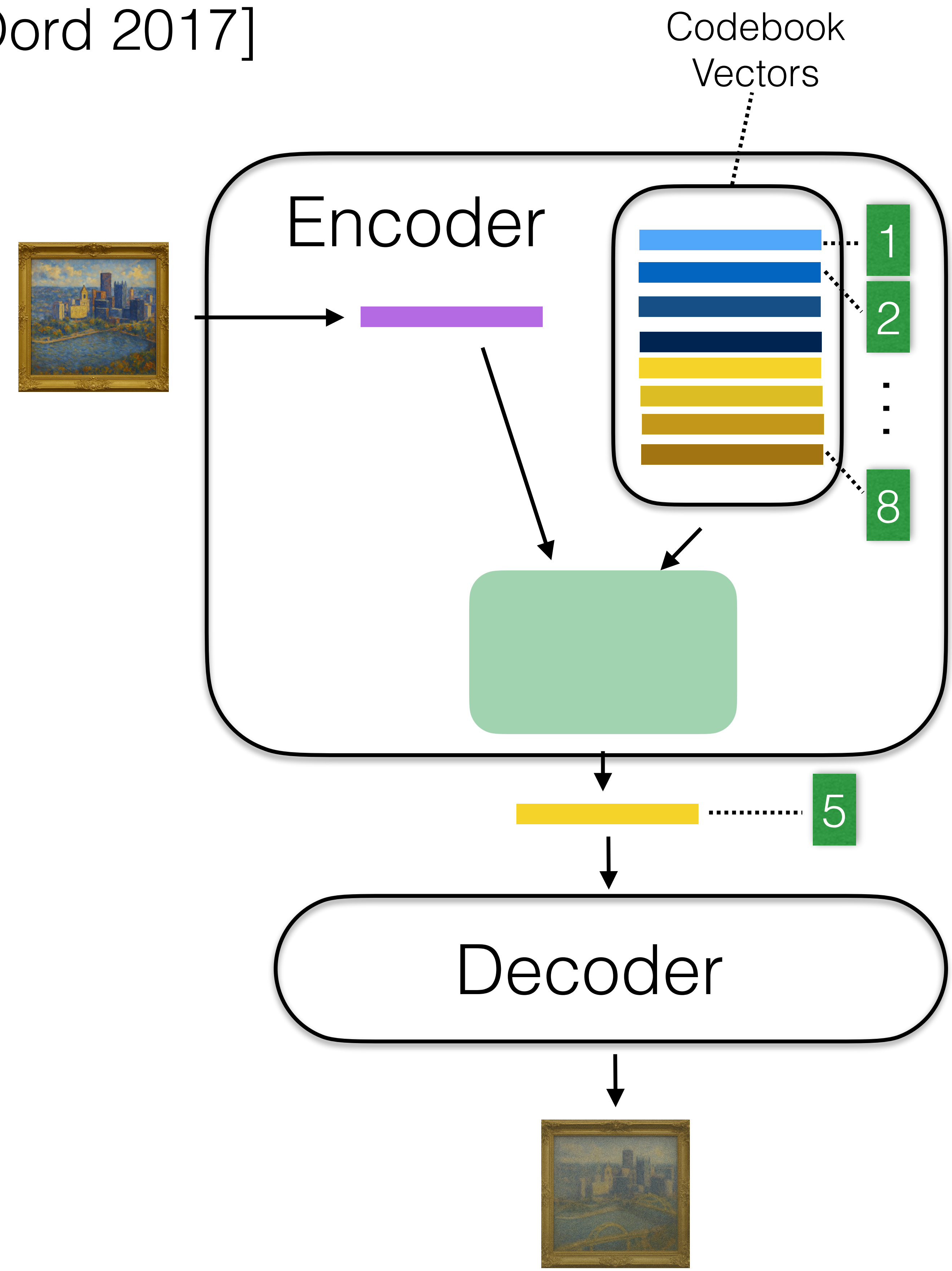
- Idea: make a *discrete* VAE to let us:
    - encode an image as a sequence of *discrete tokens*
    - decode an image from a sequence of discrete tokens



VQ-VAE

[van den Oord 2017]

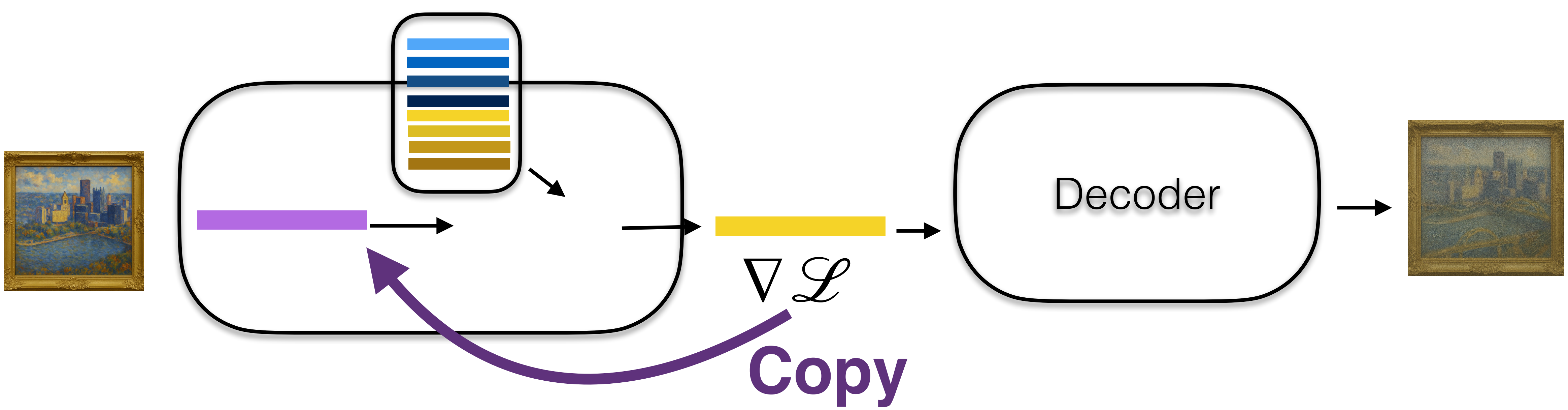
- Encoder: $q(z|x)$
  - $x \rightarrow z_e(x) \in \mathbb{R}^d$
  - Quantizer: $z_e(x) \rightarrow z_q(x)$
    - Compare $z_e(x)$ to a set of learned codebook vectors
    - $z_q(x) = e_{k^*}$, where $k^* = \arg\min_{j \in \{1,\ldots,K\}} \|z_e(x) - e_j\|_2$
  - $q(z = k|x) = \begin{cases} 1 & k = k^* \\ 0 & \text{otherwise} \end{cases}$
- Decoder:
  - $p(x|z_q(x))$



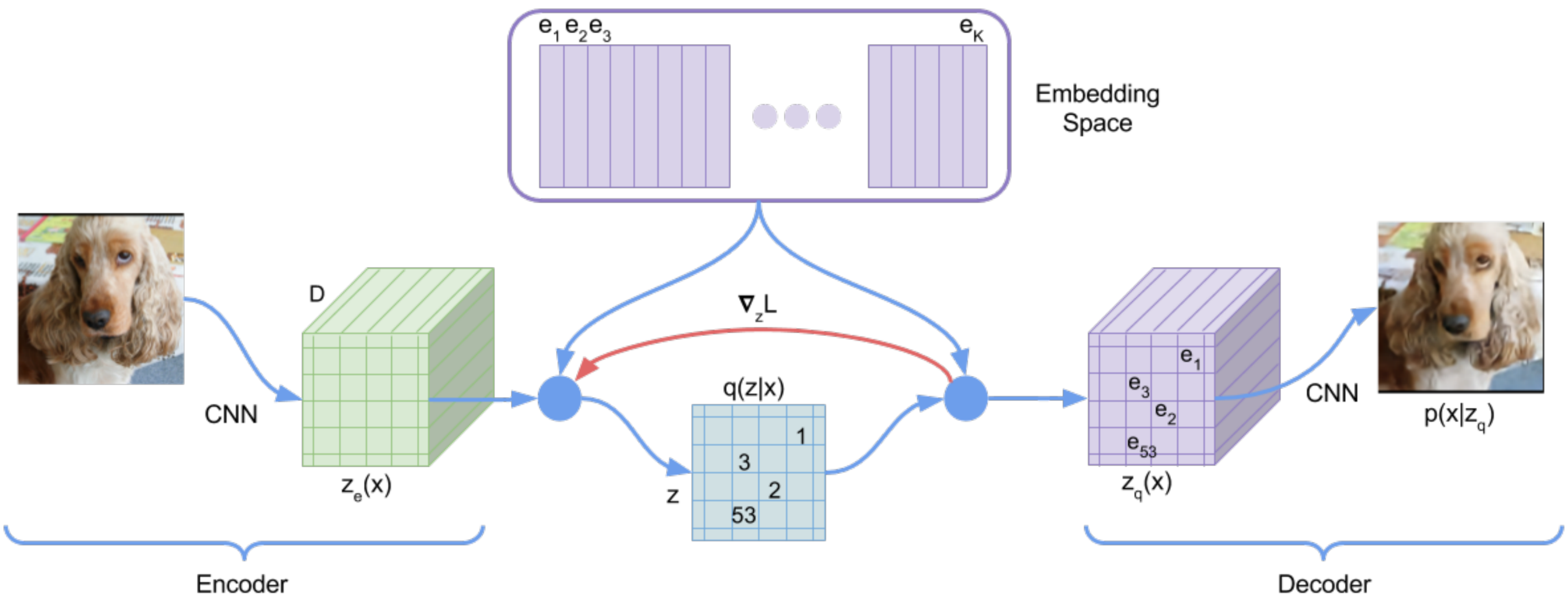
VQ-VAE training

- $\mathscr{L} = \log p(x \mid z_q(x)) + \|\text{sg}[z_e(x)] - e\|_2^2 + \beta \|z_e(x) - \text{sg}[e]\|_2^2$

- For reconstruction loss, use a **straight-through gradient estimator** since the codebook lookup is non-differentiable



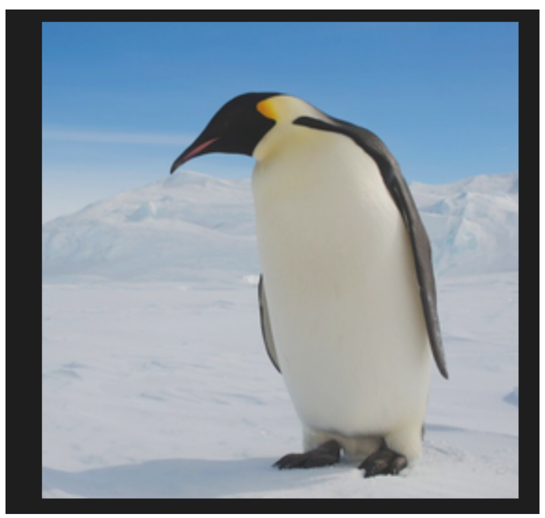
VQ-VAE



$e_1\ e_2 e_3$
e_K
Embedding Space
D
CNN
$z_e(x)$
$\nabla_z L$
$q(z|x)$
1
3
2
53
z
e_1
e_3
e_2
e_{53}
$z_q(x)$
CNN
$p(x|z_q)$
Encoder
Decoder

Example

- DALL-E tokenizer



```
import torch.nn.functional as F

z_logits = enc(x)
z_tokens = torch.argmax(z_logits, axis=1)
z = F.one_hot(z_tokens, num_classes=enc.vocab_size).per
z_tokens, z_tokens.shape
```

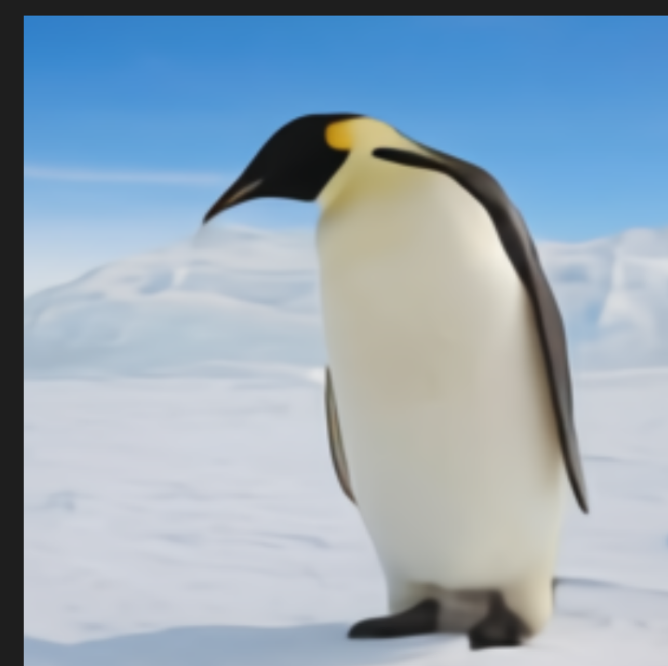
✓ 0.5s

```
(tensor([[[7522,  741,  119,  ...,  629, 4695, 4695],
          [ 782, 7459, 4762,  ..., 6913, 5215, 5887],
          [1580, 4066, 5768,  ..., 5677,  224, 2913],
          ...,
          [3990, 7130, 5047,  ..., 6027, 5770, 1212],
          [4898, 5659, 6296,  ..., 5788, 6476, 6397],
          [2757, 3243, 5504,  ..., 5016, 1144, 7261]]]),
 torch.Size([1, 32, 32]))
```

```
x_stats = dec(z).float()
x_rec = unmap_pixels(torch.sigmoid(x_stats[:, :3]))
x_rec = T.ToPILImage(mode='RGB')(x_rec[0])

display_markdown('Reconstructed image:')
display(x_rec)
```

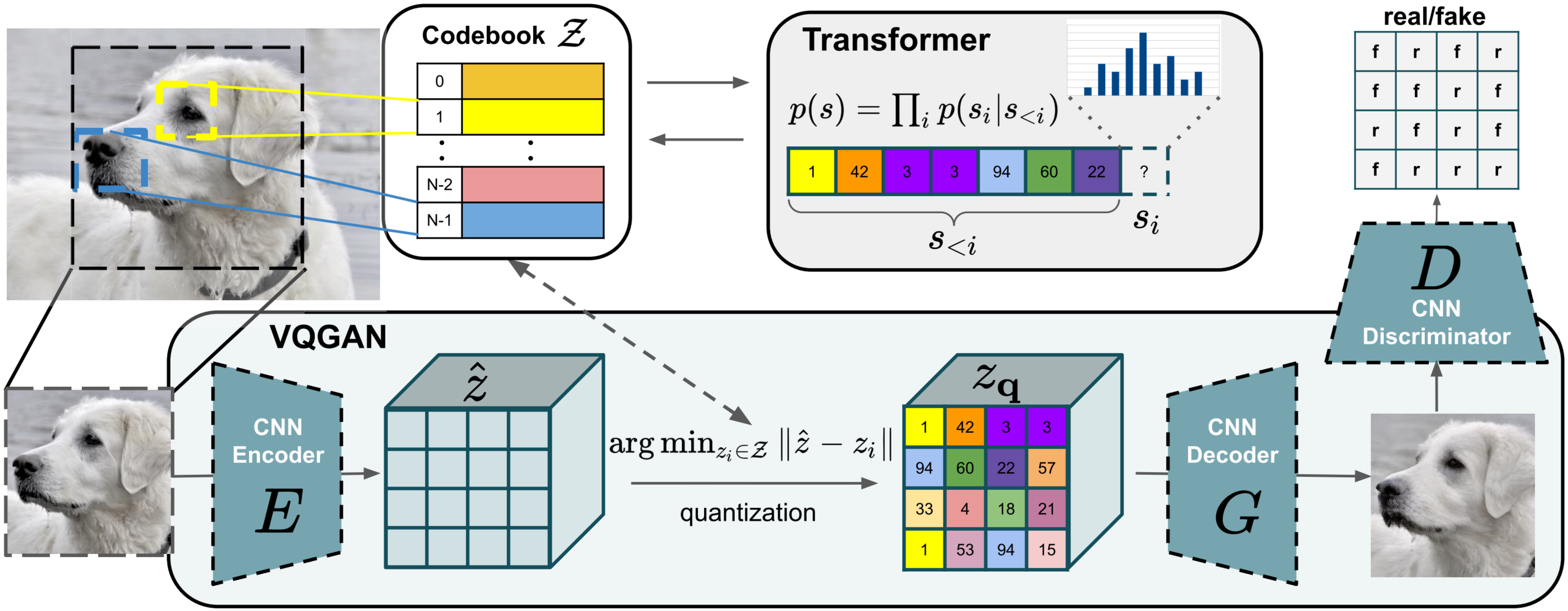
✓ 0.3s



https://github.com/openai/DALL-E

VQ-GAN

[Esser et al 2021]



- Modify VQ-VAE objective: use a GAN objective and perceptual loss

- Train an autoregressive transformer ("language model") on the resulting discrete tokenized images. Generate images with it.

VQ-GAN

[Esser et al 2021]

Training the encoder/decoder:

- $\hat{x} = G(\text{select}(E(x))$
  - G: generator
  - E: encoder
- $\mathscr{L}_{\text{GAN}} = \log D(x) + \log(1 - D(\hat{x}))$
  - D: discriminator

VQ-GAN

[Esser et al 2021]

Training the transformer language model:

- Tokenize lots of images using the tokenizer (encoder) $x_{img} \rightarrow (x_1, x_2, \ldots, x_T)$

- Train an autoregressive transformer:

- $$\mathscr{L} = \mathbb{E}_{x_{img}}[-\log \prod_{t=1}^{T} p(x_t \mid x_{<t})]$$

  - Example T: 32 x 32 = 1024

    - Much less than modeling pixels (256x256 = 65,536)!

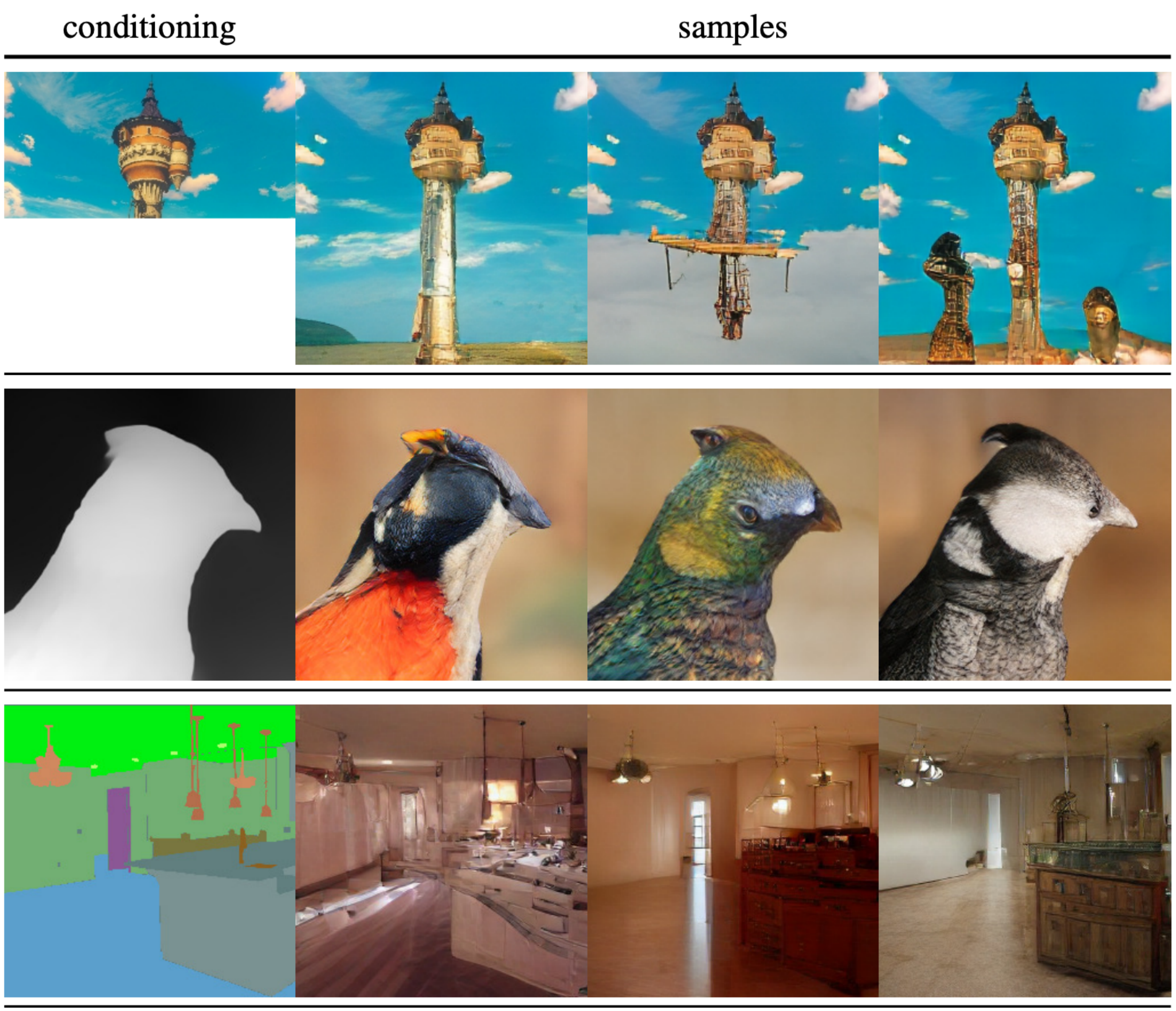
VQ-GAN

[Esser et al 2021]

Figure 8. Samples from our class-conditional ImageNet model trained on 256 × 256 images.

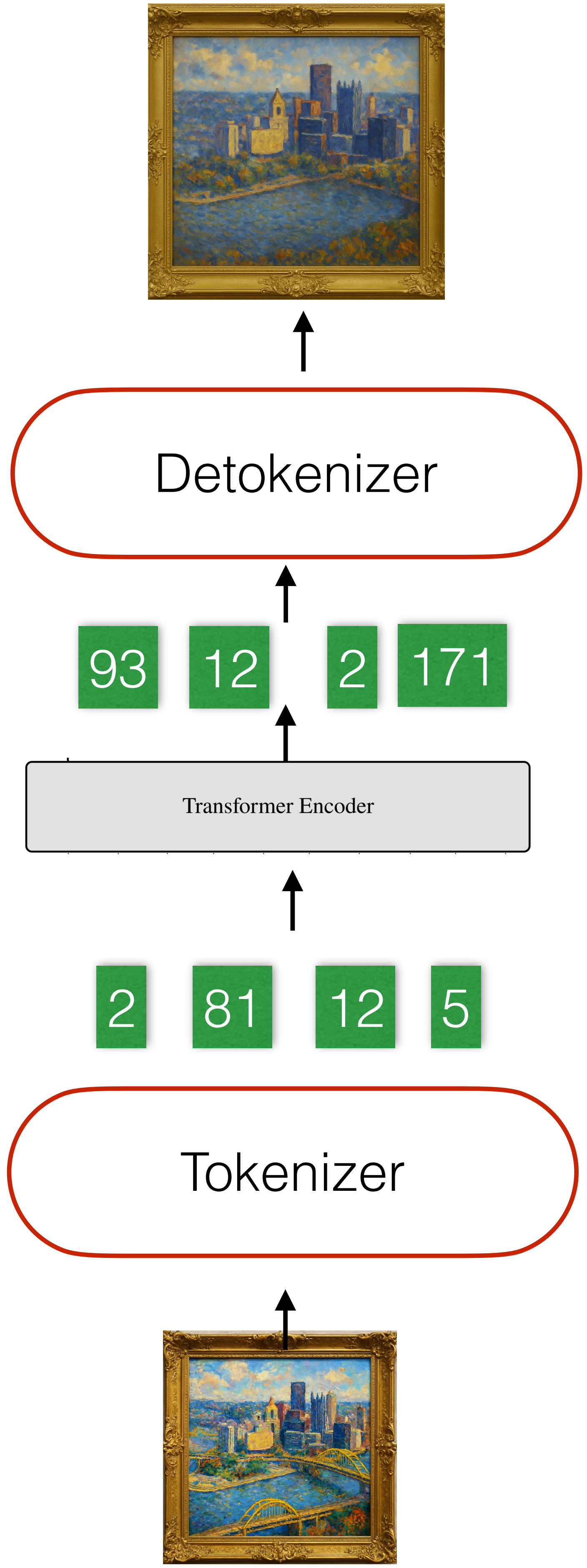
VQ-GAN

[Esser et al 2021]



Recap

- Tokenizer: VQ-VAE/VQ-GAN encoder
- De-tokenizer: VQ-VAE/VQ-GAN decoder
- Next:
  - Add the tokens to a language model's vocabulary
  - Train / fine-tune the model on both text and image tokens

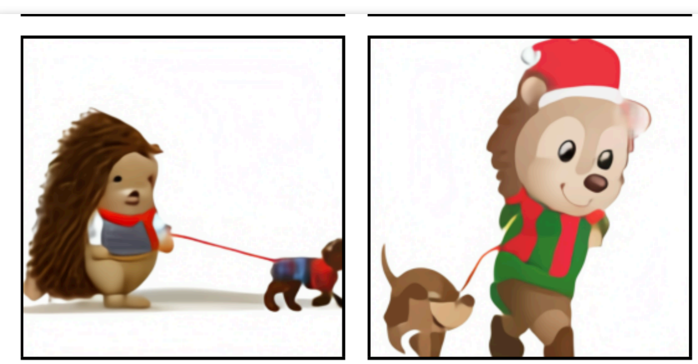


Example: Dall-E

[OpenAI 2021]

- **Stage 1.** We train a discrete variational autoencoder (dVAE)[1] to compress each 256×256 RGB image into a 32×32 grid of image tokens, each element of which can assume 8192 possible values. This reduces the context size of the transformer by a factor of 192 without a large degradation in visual quality (see Fig-

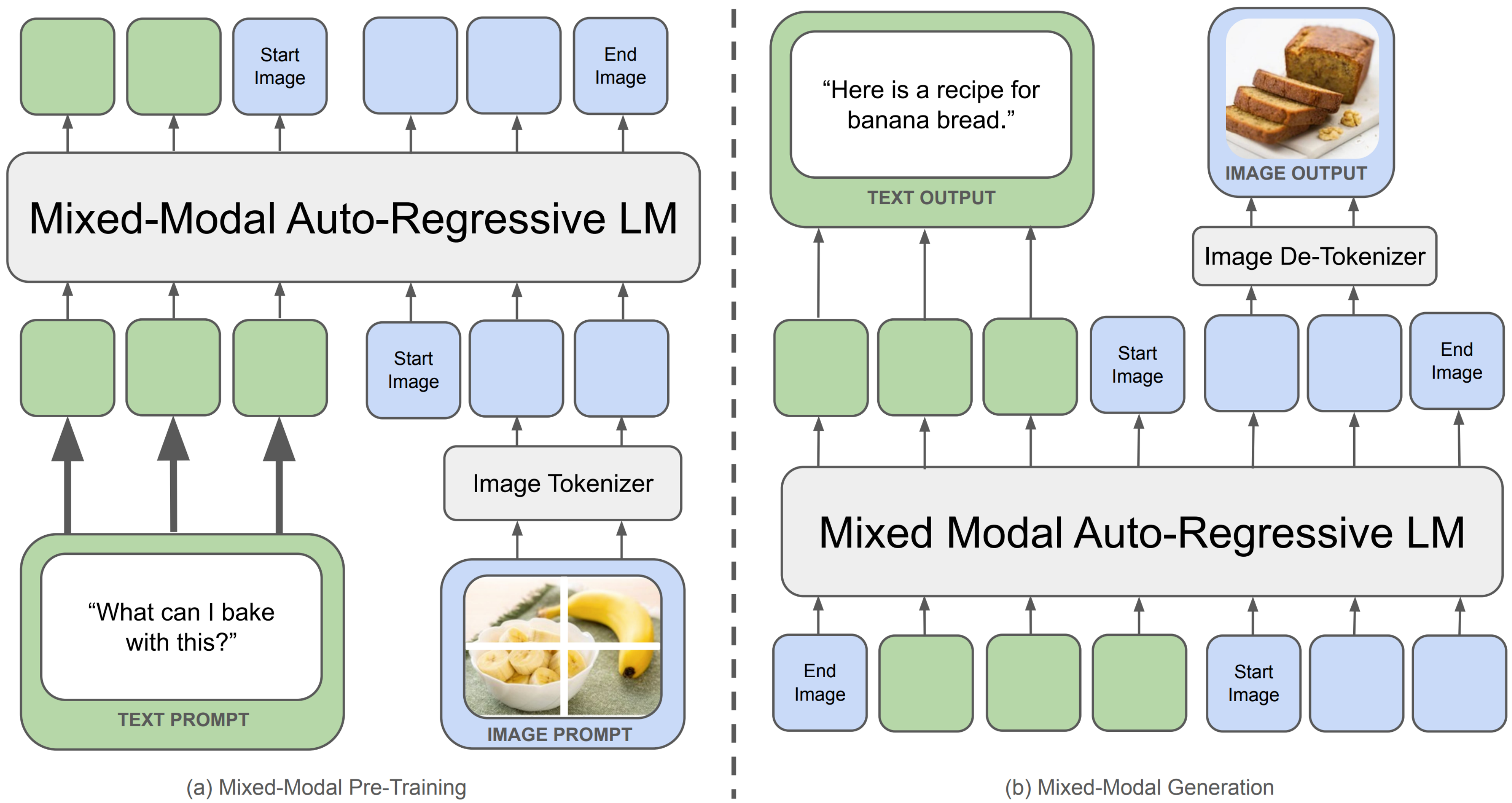
- **Stage 2.** We concatenate up to 256 BPE-encoded text tokens with the $32 \times 32 = 1024$ image tokens, and train an autoregressive transformer to model the joint distribution over the text and image tokens.



(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

Example: Chameleon

[Meta 2024]



- Train a VQ-GAN tokenizer/de-tokenizer
- Then train a language model on 10T tokens

Example: Chameleon

[Meta 2024]

Prompt: **<img>** While chameleons are masters of camouflage in the natural world, let's see how they might fare in a man-made setting.
1. Analyze the colors, patterns, and textures in this image and identify the most challenging spot for a chameleon to blend in.
2. Then generate an image of a chameleon with the optimal pattern to camouflage itself.
3. Then, rate and explain how successful each attempt is on a scale from 1-10.

Response: The image is of a stop sign in a forested area, with some sunlight filtering through the trees. The sign has a red octagonal shape with a white border and the word "STOP" written in white letters. In the background, there are green leaves and trees.

Challenging Spot for a Chameleon: The stop sign would be the most challenging spot for a chameleon to blend in due to its bright red color and bold white letters, which would stand out in the forested environment.

Optimal Pattern for Camouflage: To camouflage itself on the stop sign, a chameleon would need to have a pattern that would break up its outline and make it less conspicuous. A pattern with shades of brown and red would help the chameleon blend in more effectively. Additionally, a mottled or dappled pattern would help to scatter light and further reduce the chameleon's visibility.

<img>

Rating and Explanation: A rating of 2/10 would be given for the chameleon's ability to blend in on the stop sign. While the chameleon's green and brown pattern helps to break up its outline, the bright red color of the stop sign will still make it stand out in the forested environment.

Pros and Cons

- **Pros**: unified generative model (autoregressive, everything is tokens)
- **Cons**: information loss from the image tokenizer/ detokenizer
- To generate SOTA-quality images we'll use a different paradigm (next lecture!)

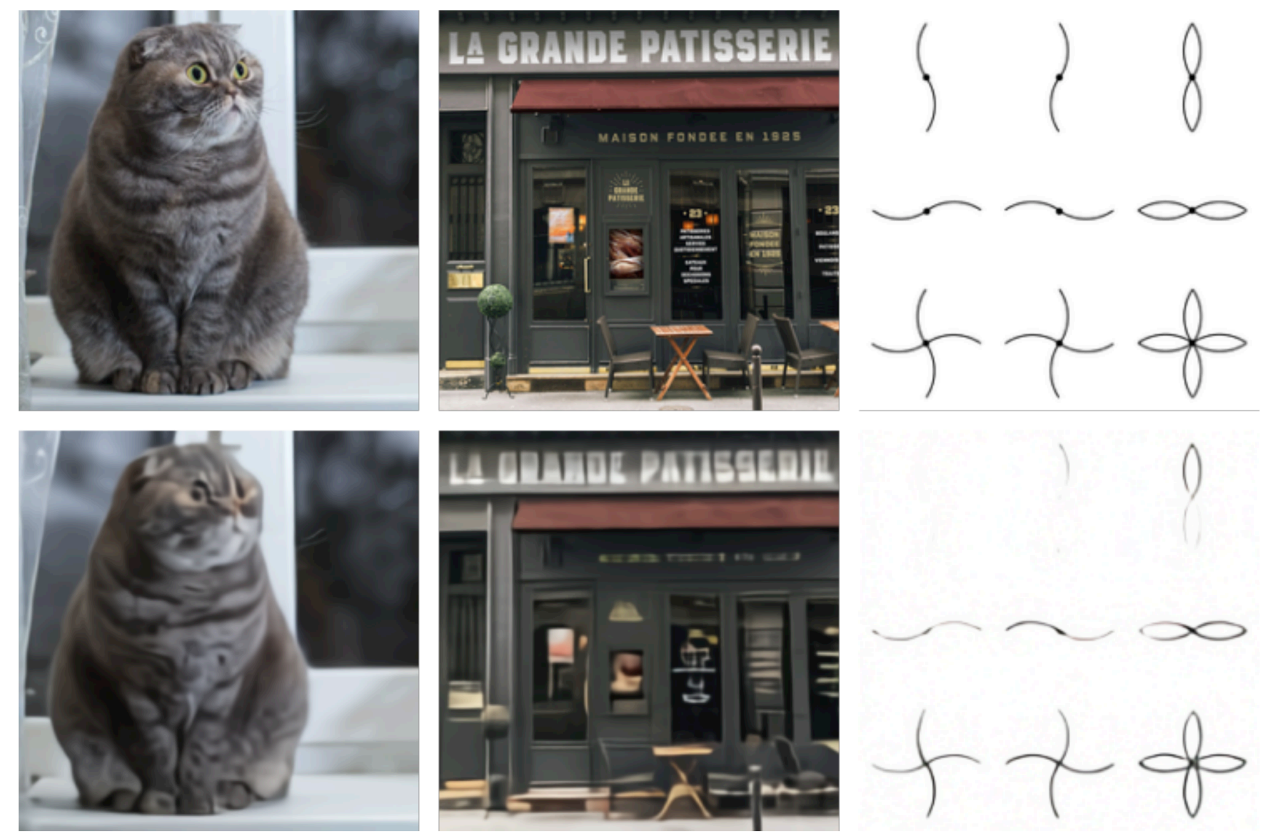
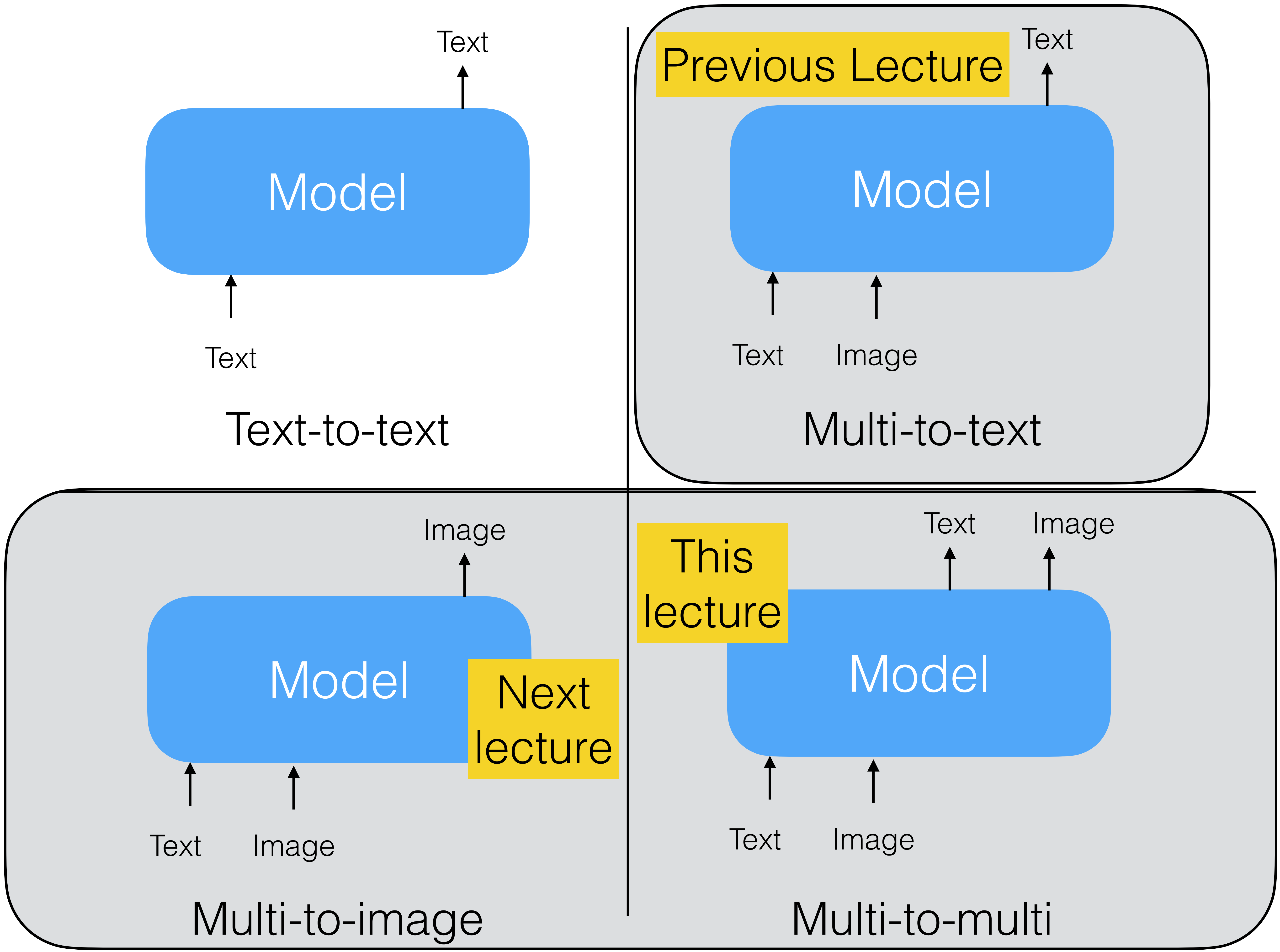


Figure 1. Comparison of original images (top) and reconstructions from the discrete VAE (bottom). The encoder downsamples the

[OpenAI 2021]


Text
Model
Text
Text-to-text
Previous Lecture
Text
Model
Text
Image
Multi-to-text
Image
Model
Next lecture
Text
Image
Multi-to-image
This lecture
Text
Image
Model
Text
Image
Multi-to-multi

Thank you!