10-423/623: Generative AI Lecture 12 – Text-to-Image Generation

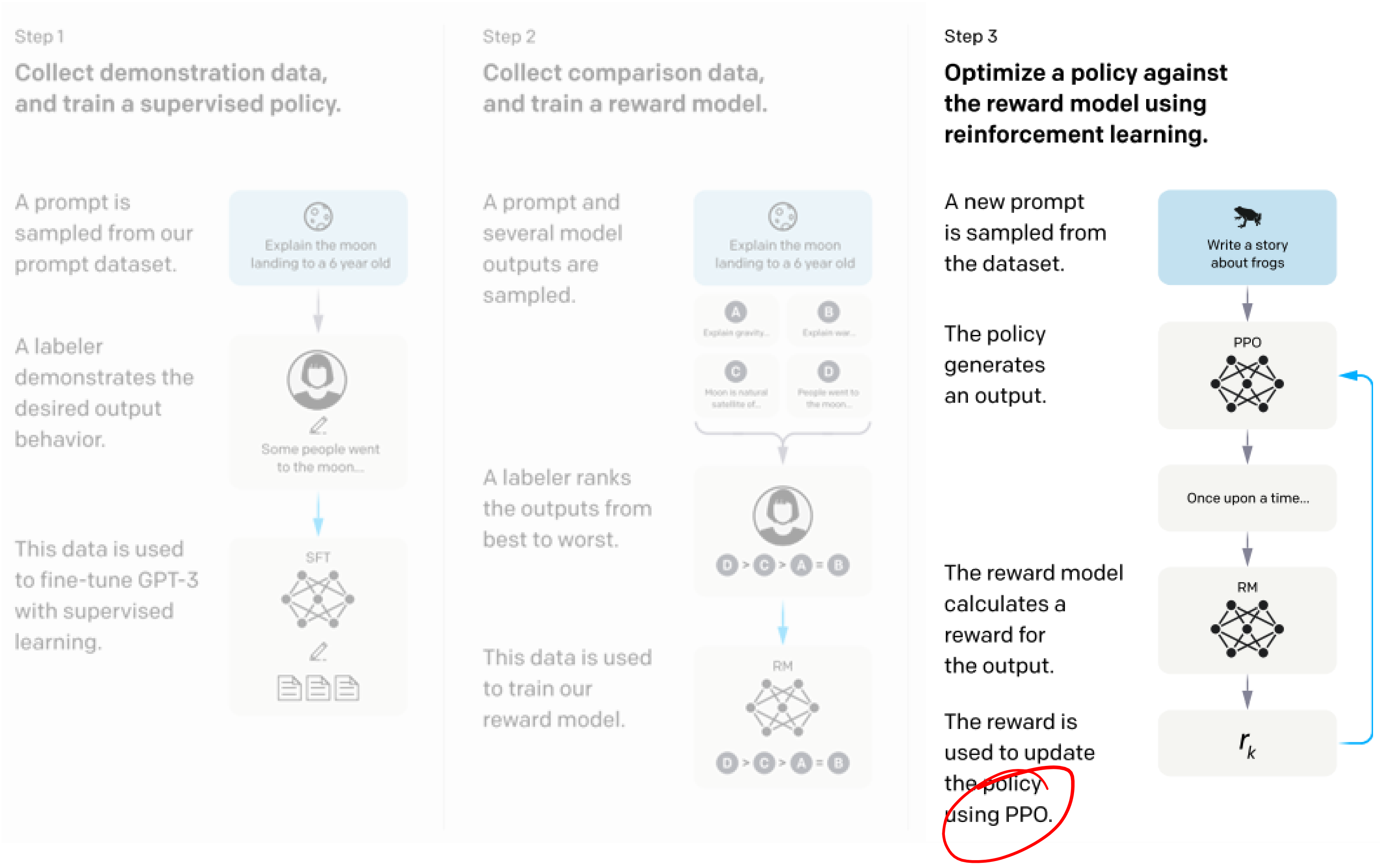
Henry Chai & Matt Gormley

10/7/24

Front Matter

- Announcements:
  - HW2 released ~~9/23~~ 9/24, due 10/7 (today!) at 11:59 PM
  - HW3 released 10/7 (today!), due 10/23 at 11:59 PM
    - You are *not* expected to work on HW3 over Fall Break
  - Quiz 3 ~~moved to 9/30 (Monday)~~ is on 10/9
    - Will cover Lectures 9 – 12 (only the RLHF/DPO portion of today's lecture)

Recall: Reinforcement Learning from Human Feedback

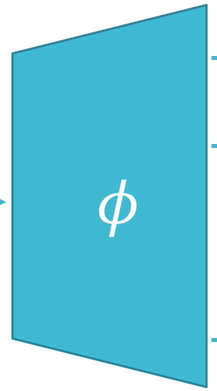


Source: https://arxiv.org/pdf/2203.02155

Reinforcement Learning: Object of Interest for Fine-tuning LLMs

- **The LLM to be fine-tuned,** $\pi_\phi(a \mid s)$
  - Specifies a distribution over next tokens given any input sequence

Model: $s_t \rightarrow$



$$p(a_1|s_t;\phi) \coloneqq \pi_\phi(a_1|s_t)$$
$$p(a_2|s_t;\phi) \coloneqq \pi_\phi(a_2|s_t)$$
$$\vdots$$
$$p(a_{|\mathcal{A}|}|s_t;\phi) \coloneqq \pi_\phi(a_{|\mathcal{A}|}|s_t)$$

- An *episode* $\mathrm{T} = \{x, a_0, s_1, a_1, \dots, s_T\}$ is one completion of the prompt x, ending in an EOS token
- The LLM induces a distribution over possible completions

$$p_\phi(\mathrm{T}) = p(\{a_0, s_1, a_1, \dots, s_T\} \mid x \coloneqq s_0)$$
$$= \prod_{t=0}^{T-1} \pi_\phi(a_t|s_t)$$

Objective function: $\ell(\phi) = -\mathbb{E}_{p_\phi(\mathrm{T})}[R_\theta(\mathrm{T})]$, the negative expected reward of a response

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\phi \ell(\phi) = \nabla_\phi\left(-\mathbb{E}_{p_\phi(\mathrm{T})}[R_\theta(\mathrm{T})]\right) = \nabla_\phi\left(-\int R_\theta(\mathrm{T})p_\phi(\mathrm{T})\,d\mathrm{T}\right)$$

$$= -\int R_\theta(\mathrm{T})\nabla_\phi p_\phi(\mathrm{T})d\mathrm{T} = -\int R_\theta(\mathrm{T})\nabla_\phi\underbrace{(\log p_\phi(\mathrm{T}))p_\phi(\mathrm{T})}d\mathrm{T}$$

$$= -\mathbb{E}_{p_\phi(\mathrm{T})}[R_\theta(\mathrm{T})\nabla_\phi(\log p_\phi(\mathrm{T}))]$$

$$\approx -\frac{1}{N}\sum_{n=1}^{N} R_\theta(\mathrm{T}^{(n)})\,\nabla_\phi(\log p_\phi(\mathrm{T}^{(n)}))$$

(where $\mathrm{T}^{(n)} = \left\{a_0^{(n)}, s_1^{(n)}, a_1^{(n)}, \dots, s_{T^{(n)}}^{(n)}\right\}$ is a sampled completion of x)

$$= -\frac{1}{N}\sum_{n=1}^{N} r_\theta\left(x, \left[a_0^{(n)}, \dots, a_{T^{(n)}}^{(n)}\right]\right)\left(\sum_{t=0}^{T^{(n)}-1}\nabla_\phi \log \pi_\phi\left(a_t^{(n)}\middle|s_t^{(n)}\right)\right)$$

Proximal Policy Optimization (Schulman et al., 2017)

- There are two high-level modifications to get from REINFORCE to proximal policy optimization (PPO):
   1. Sampled trajectories/rewards can be highly variable, which leads to unstable estimates of the expectation
      - Instead of working with R_θ, PPO considers a trajectory's *advantage* over some *baseline*
      - The baseline is typically defined in terms of the *value function* at each state in the trajectory

Source: https://arxiv.org/pdf/1707.06347

Proximal Policy Optimization (Schulman et al., 2017)

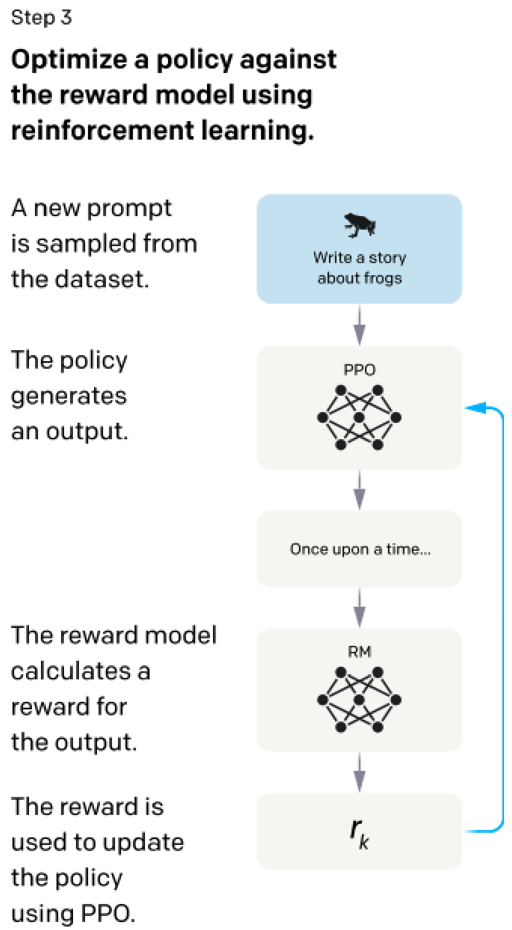
- There are two high-level modifications to get from REINFORCE to proximal policy optimization (PPO):
  2. Policy gradient methods are *on-policy*: the policy being optimized is also being used to generate the trajectories used in training
     - This can also lead to instability/poor convergence if the policy ever becomes bad
     - Intuition: ensure that the policy remains "close to" some policy known to be good
       - In RLHF, we can just use the original (instruction fine-tuned) LLM!

Source: https://arxiv.org/pdf/1707.06347

Reinforcement Learning from Human Feedback: PPO

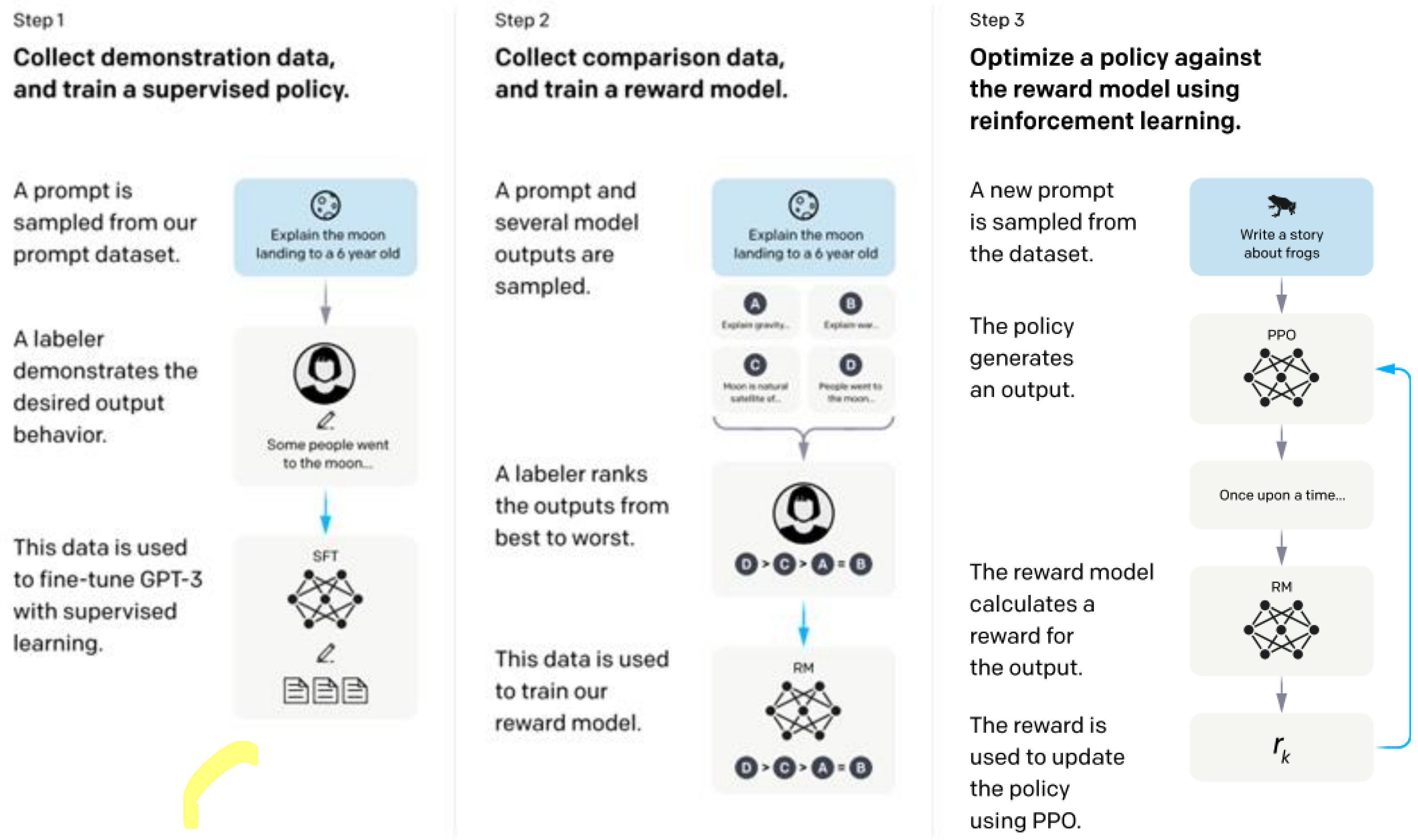
- Step 3 fine-tunes the LLM's parameters using the PPO objective *plus a pre-training loss* term:

$$\ell(\phi) = -\mathbb{E}_{p_{\phi}(\mathrm{T})}\left[R_{\theta}(\mathrm{T}) - \beta \log \frac{\pi_{\phi}^{RL}(\mathrm{T})}{\pi^{SFT}(\mathrm{T})}\right]$$
$$-\gamma \mathbb{E}_{x \sim D_{pretrain}}\left[\log \pi_{\phi}^{RL}(\mathrm{x})\right]$$



Source: https://arxiv.org/pdf/2203.02155

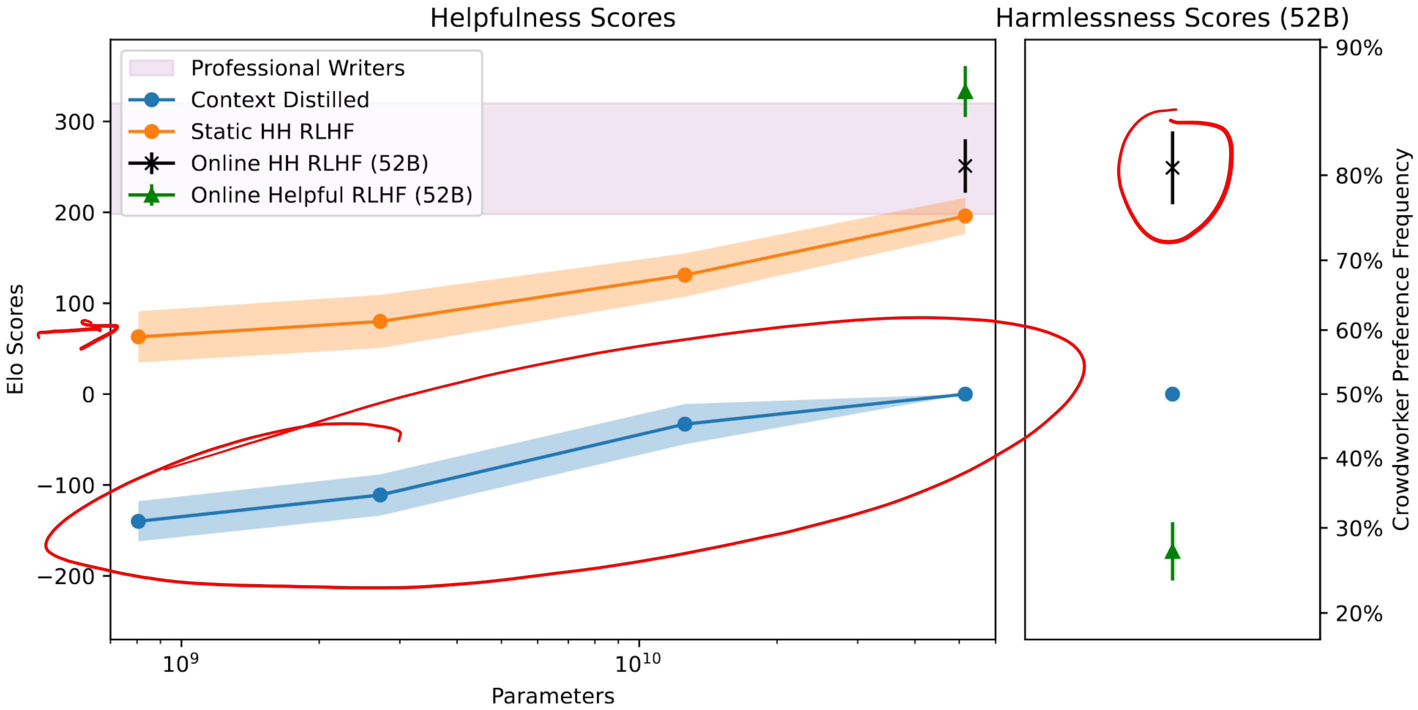
Alright, so what does all of this get us?



Source: https://arxiv.org/pdf/2203.02155

Reinforcement Learning from Human Feedback: Results

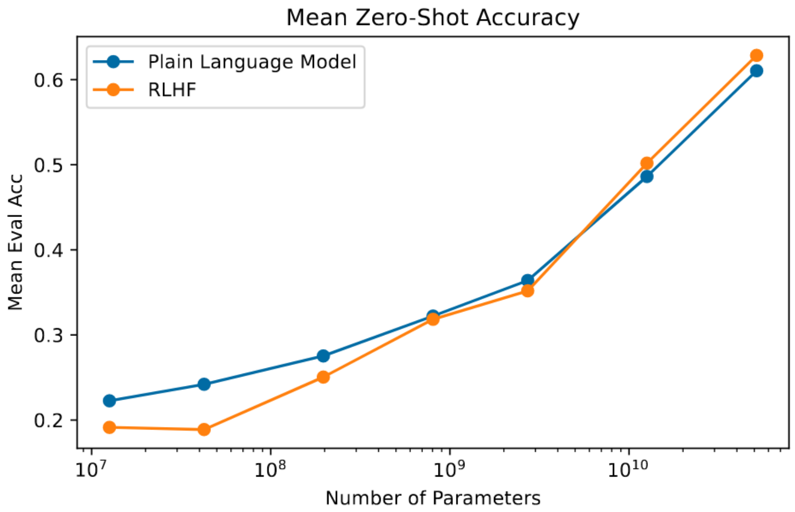
- Reinforcement learning from human feedback
  1. **increases perceived helpfulness and harmlessness** ("context distilled" corresponds to an instruction fine-tuned LLM, tune for helpfulness and harmlessness)

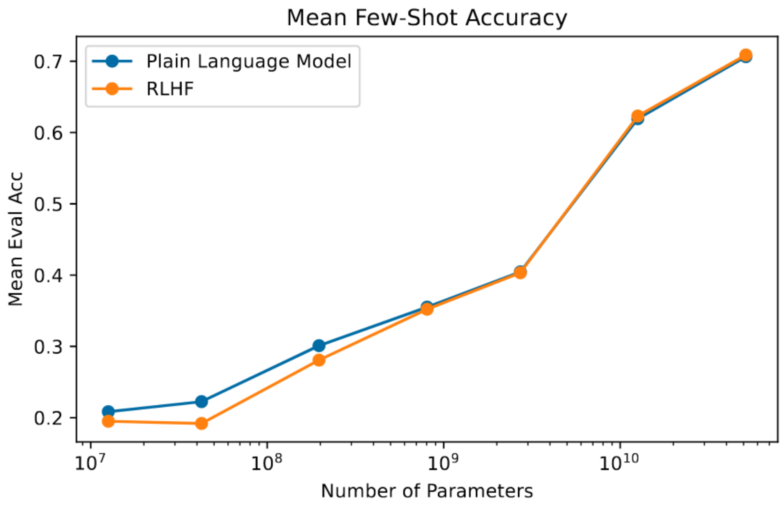


Source: http://arxiv.org/abs/2204.05862

Reinforcement Learning from Human Feedback: Results

- Reinforcement learning from human feedback
  1. increases perceived helpfulness and harmlessness
  2. **does not (significantly) decrease zero-shot or few-shot performance on most tasks**

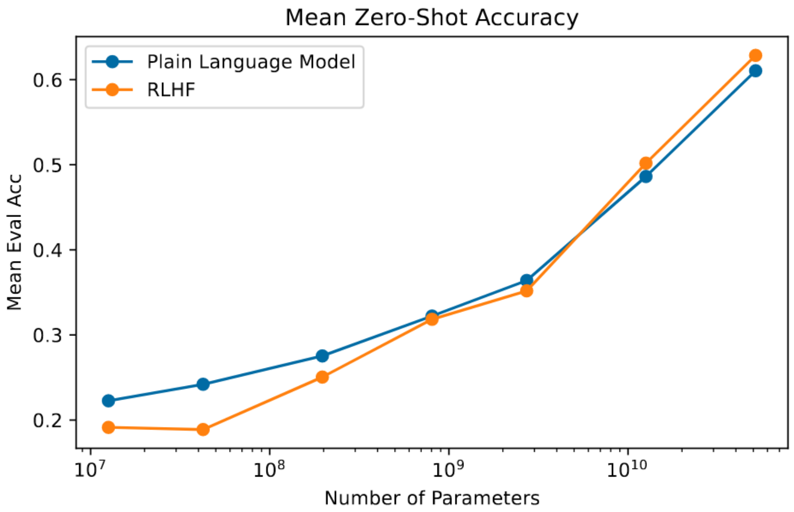


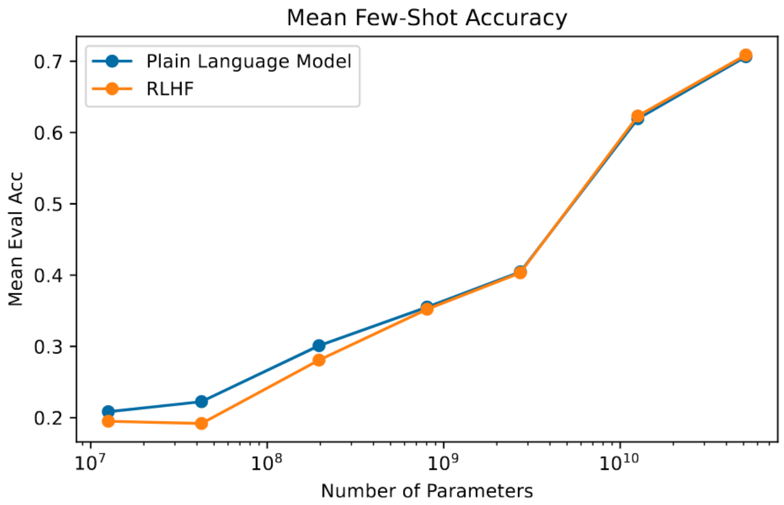


Source: http://arxiv.org/abs/2204.05862

Man, reinforcement learning seems hard; couldn't we do something easier?

- Reinforcement learning from human feedback
  1. increases perceived helpfulness and harmlessness
  2. **does not (significantly) decrease zero-shot or few-shot performance on most tasks**





Source: http://arxiv.org/abs/2204.05862

Direct Preference Optimization (Rafailov et al., 2023)

- Intuition: in some sense, the reinforcement learning problem we defined for fine-tuning LLMs to human preferences is very "simple"
  - All of the dynamics (the state space, action space, transition function, reward model) are all known a priori and deterministic
- Idea: instead of optimizing a learned reward model, fine-tune the LLM using the stated preferences directly
  - Increase the likelihood of higher-ranking responses, y_w, and decrease the likelihood of lower-ranking responses, y_l.

Source: https://arxiv.org/pdf/2305.18290

Direct Preference Optimization (Rafailov et al., 2023)

- **Assume** there exists a (universal) latent reward model, r^*, that is responsible for the observed preferences according to

$$p(y_w \succ y_l \mid x) = \frac{\exp r^*(x, y_w)}{\exp r^*(x, y_w) + \exp r^*(x, y_l)}$$

- If we knew this true reward model, the objective function RLHF would try to optimize (without the pre-training loss) is

$$\ell(\phi) = -\mathbb{E}_{p_\phi(y|x)}\left[r^*(x, y) - \beta \log \frac{\pi_\phi(y|x)}{\pi^{SFT}(y|x)}\right]$$

- It can be shown that the optimal policy satisfies

$$\pi_{\phi^*}(y|x) = \frac{1}{Z(x)} \pi^{SFT}(y|x) \exp\left(\frac{r^*(x, y)}{\beta}\right)$$

for some normalizing factor $Z(x)$

Source: https://arxiv.org/pdf/2305.18290

Direct Preference Optimization (Rafailov et al., 2023)

- **Assume** there exists a (universal) latent reward model, r^*, that is responsible for the observed preferences according to

$$p(y_w \succ y_l \mid x) = \frac{\exp r^*(x, y_w)}{\exp r^*(x, y_w) + \exp r^*(x, y_l)}$$

- If we knew this true reward model, the objective function RLHF would try to optimize (without the pre-training loss) is

$$\ell(\phi) = -\mathbb{E}_{p_\phi(y|x)}\left[r^*(x, y) - \beta \log \frac{\pi_\phi(y|x)}{\pi^{SFT}(y|x)}\right]$$

- It can be shown that the optimal policy satisfies

$$\pi_{\phi^*}(y|x) = \frac{1}{Z(x)} \pi^{SFT}(y|x) \exp\left(\frac{r^*(x, y)}{\beta}\right)$$

solving this for r^* and plugging it into the probability above…

Source: https://arxiv.org/pdf/2305.18290

Direct Preference Optimization (Rafailov et al., 2023)

- **Assume** that the LLM π_{ϕ^*}

  is responsible for the observed preferences according to

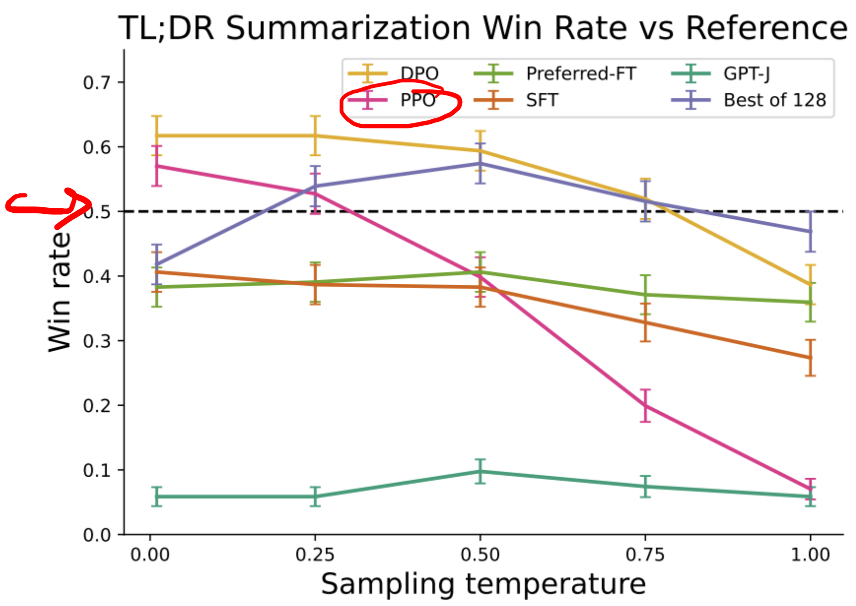
$$p(y_w \succ y_l \mid x) = \frac{1}{1 + \exp\left(\beta \log \frac{\pi_{\phi^*}(y_l|x)}{\pi^{SFT}(y_l|x)} - \beta \log \frac{\pi_{\phi^*}(y_w|x)}{\pi^{SFT}(y_w|x)}\right)}$$

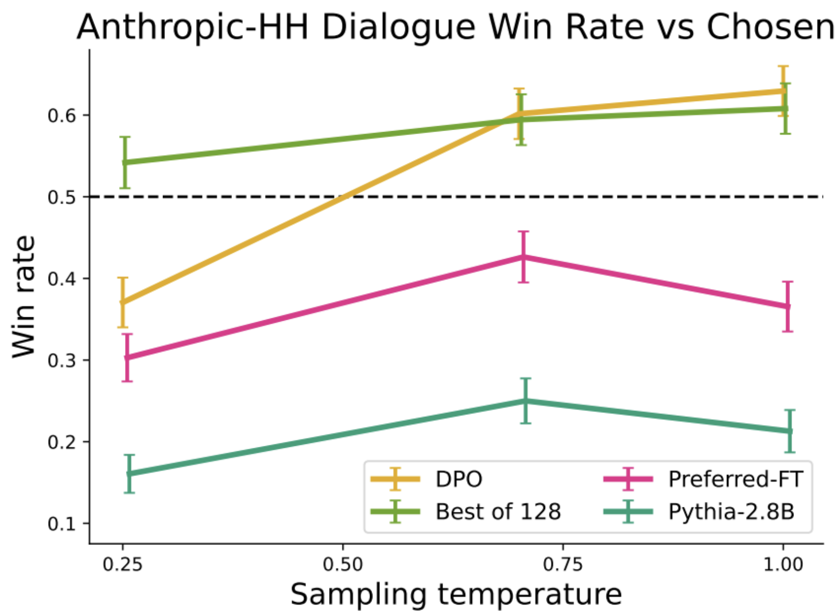
  - "Your language model is secretly a reward model"

- Key takeaway: we can directly optimize the LLM parameters, ϕ, by maximizing this probability over samples (x, y_w, y_l) from the human labelled preferences dataset $\mathcal{D}$!

Source: https://arxiv.org/pdf/2305.18290

Direct Preference Optimization (Rafailov et al., 2023)





- "For summarization, we use reference summaries in the test set as the baseline; for dialogue, we use the preferred response in the test dataset as the baseline"
- Key caveat: "we evaluate algorithms with their win rate against a baseline policy, *using GPT-4 as a proxy for human evaluation…"*

Source: https://arxiv.org/pdf/2305.18290

Recall: Image Generation

Prompt: A propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese.

- Given a text description, sample an image that depicts the prompt

- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- **Text-to-image (TTI) generation**

Source: https://arxiv.org/pdf/2307.01952.pdf

Timeline: Text-to-Image Generation

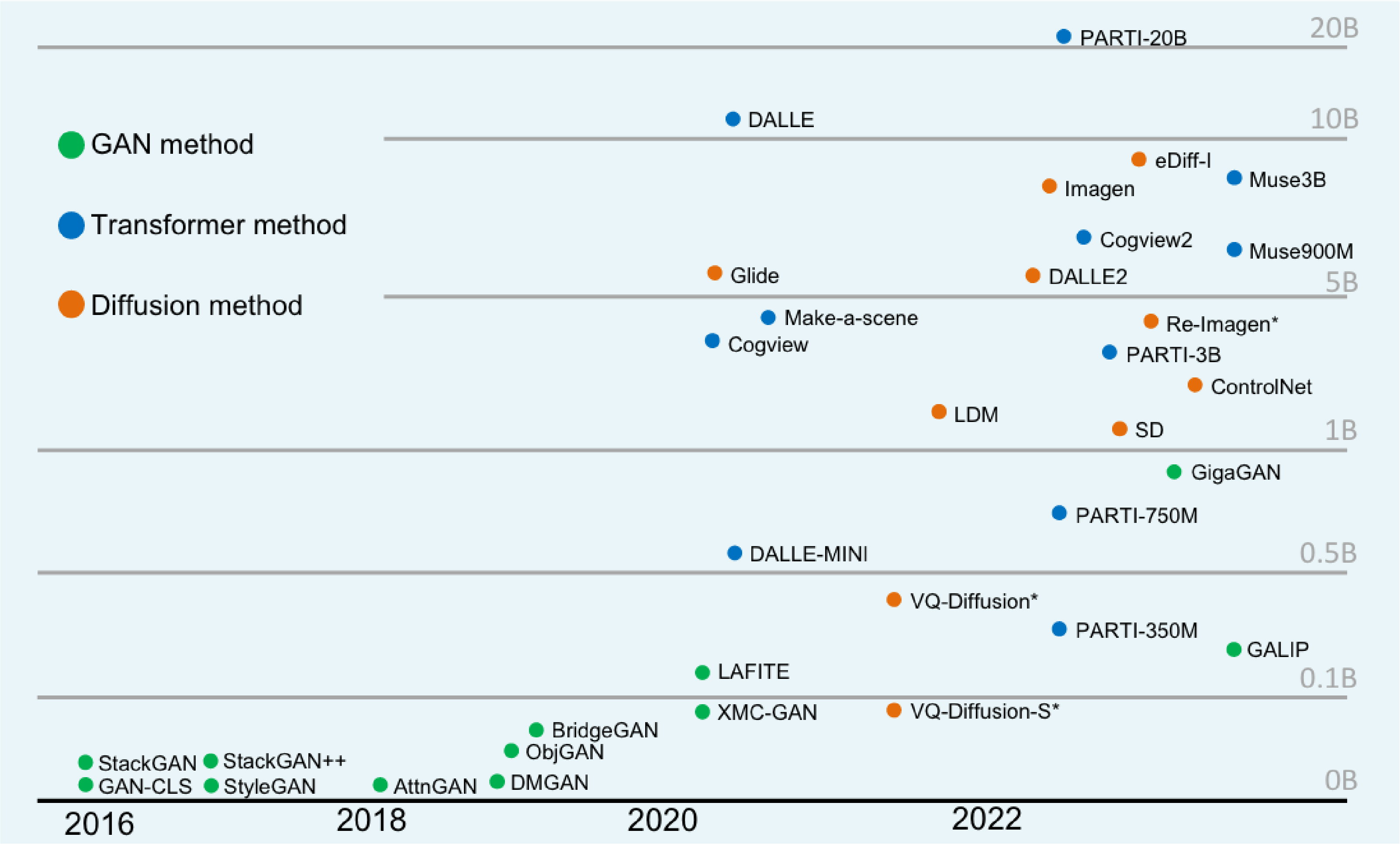
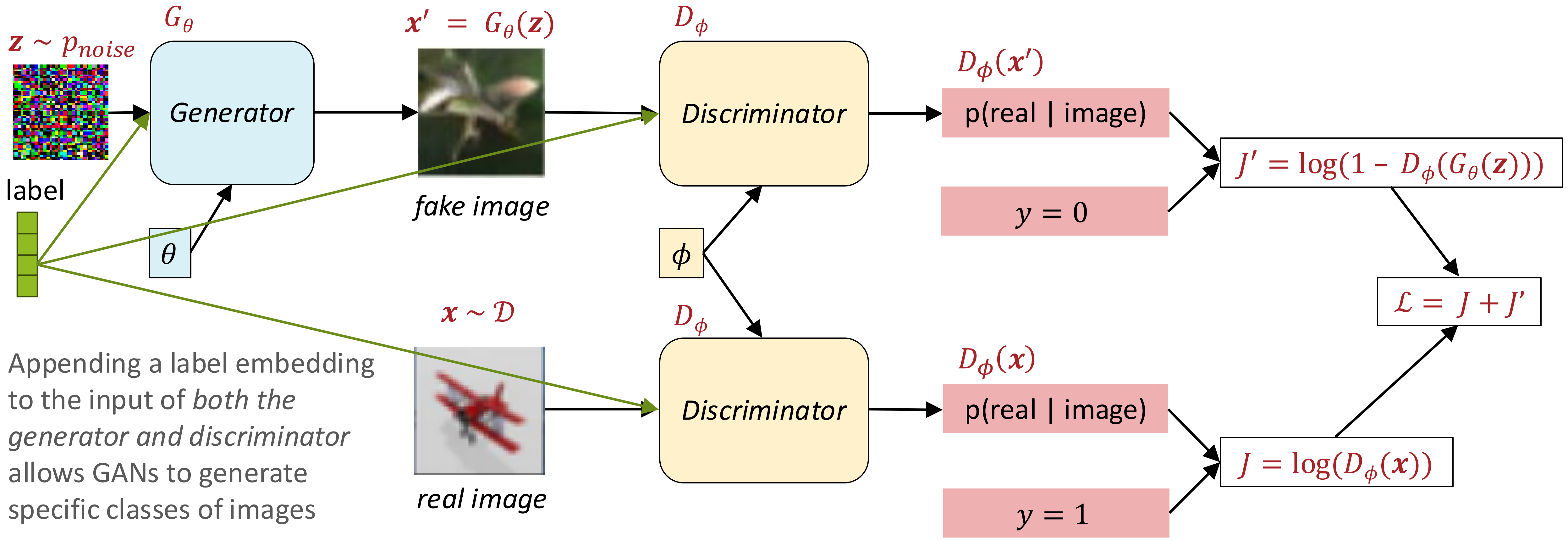


Fig. 5. Timeline of TTI model development, where green dots are GAN TTI models, blue dots are autoregressive Transformers and orange dots are Diffusion TTI models. Models are separated by their parameter, which are in general counted for all their components. Models with asterisk are calculated without the involvement of their text encoders.

Source: http://arxiv.org/abs/2309.00810



Appending a label embedding to the input of *both the generator and discriminator* allows GANs to generate specific classes of images

Recall: Class-conditional GANs

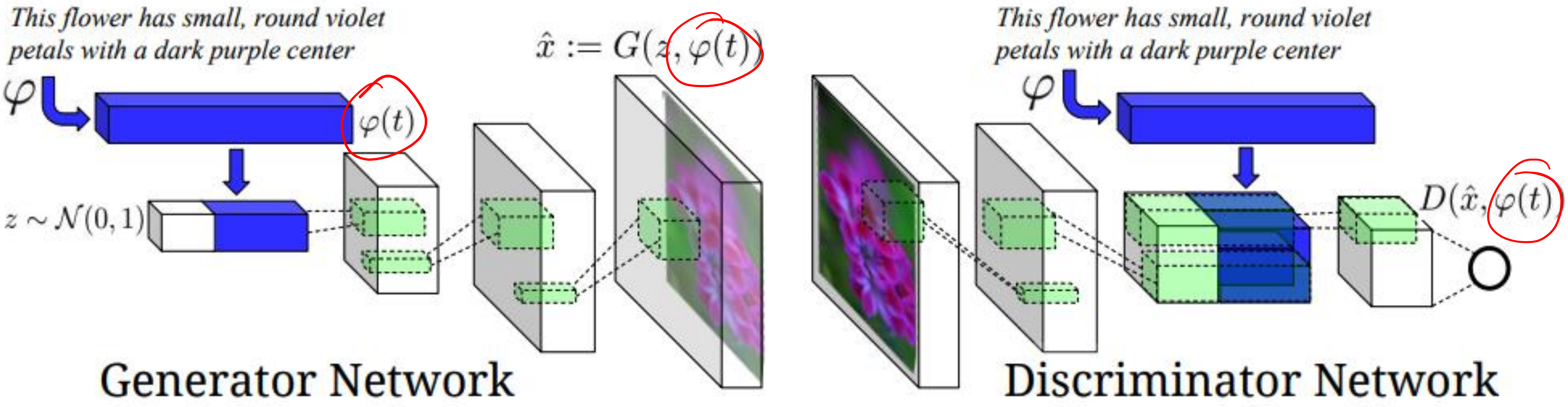
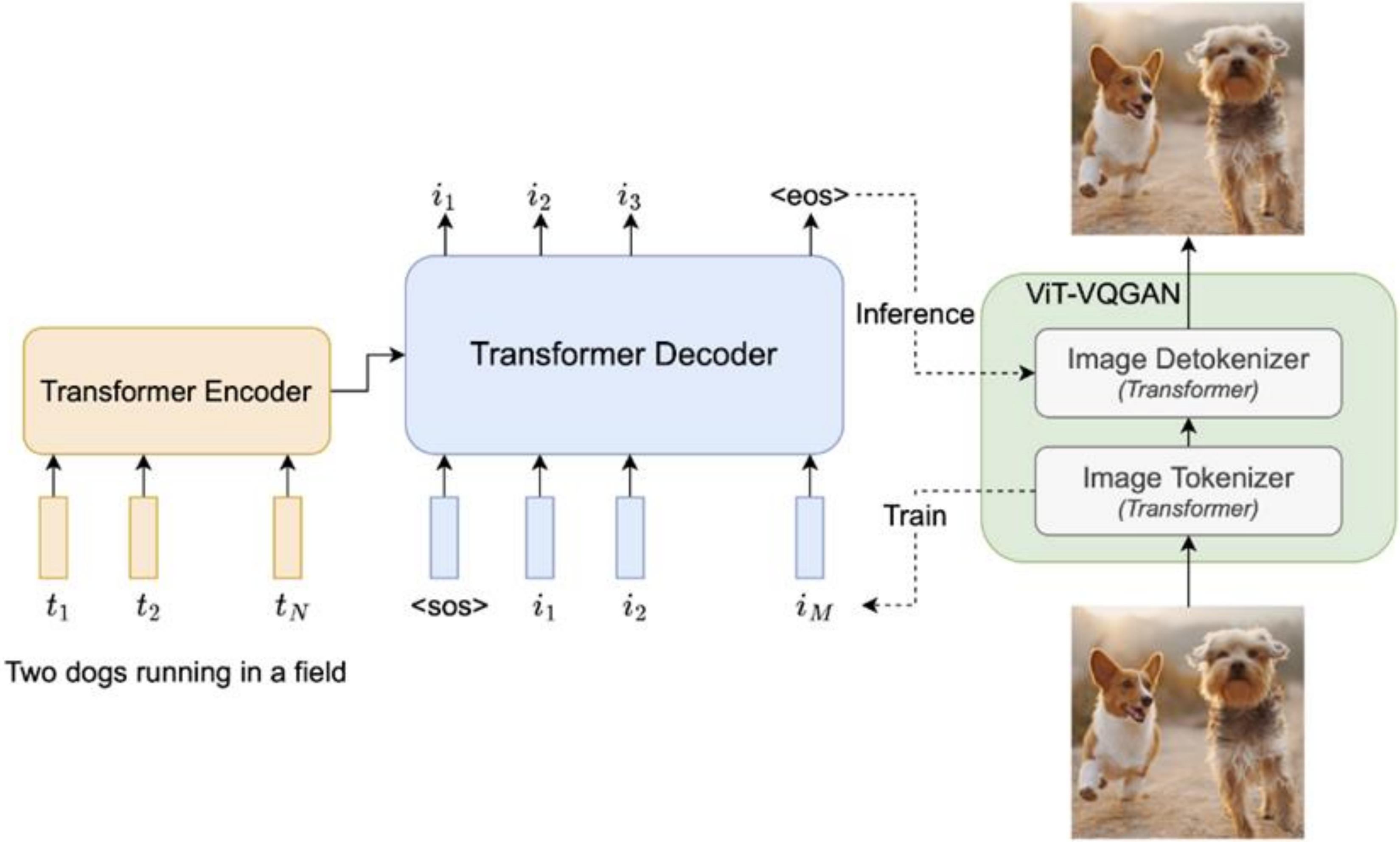


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Generative adversarial text to image synthesis

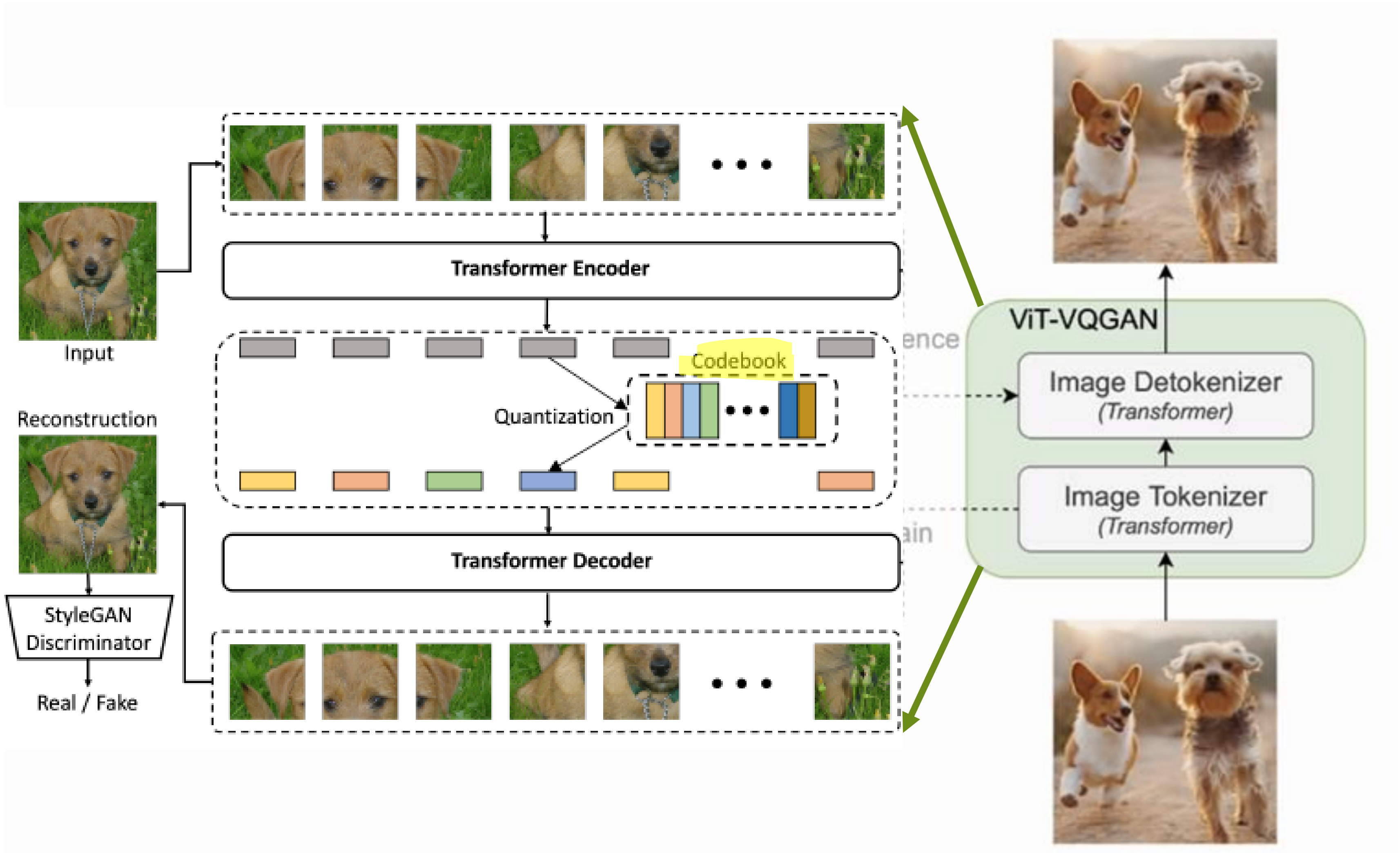
Source: https://arxiv.org/pdf/1605.05396

Pathways Autoregressive Text-to-Image (Parti)



Source: https://arxiv.org/pdf/2206.10789

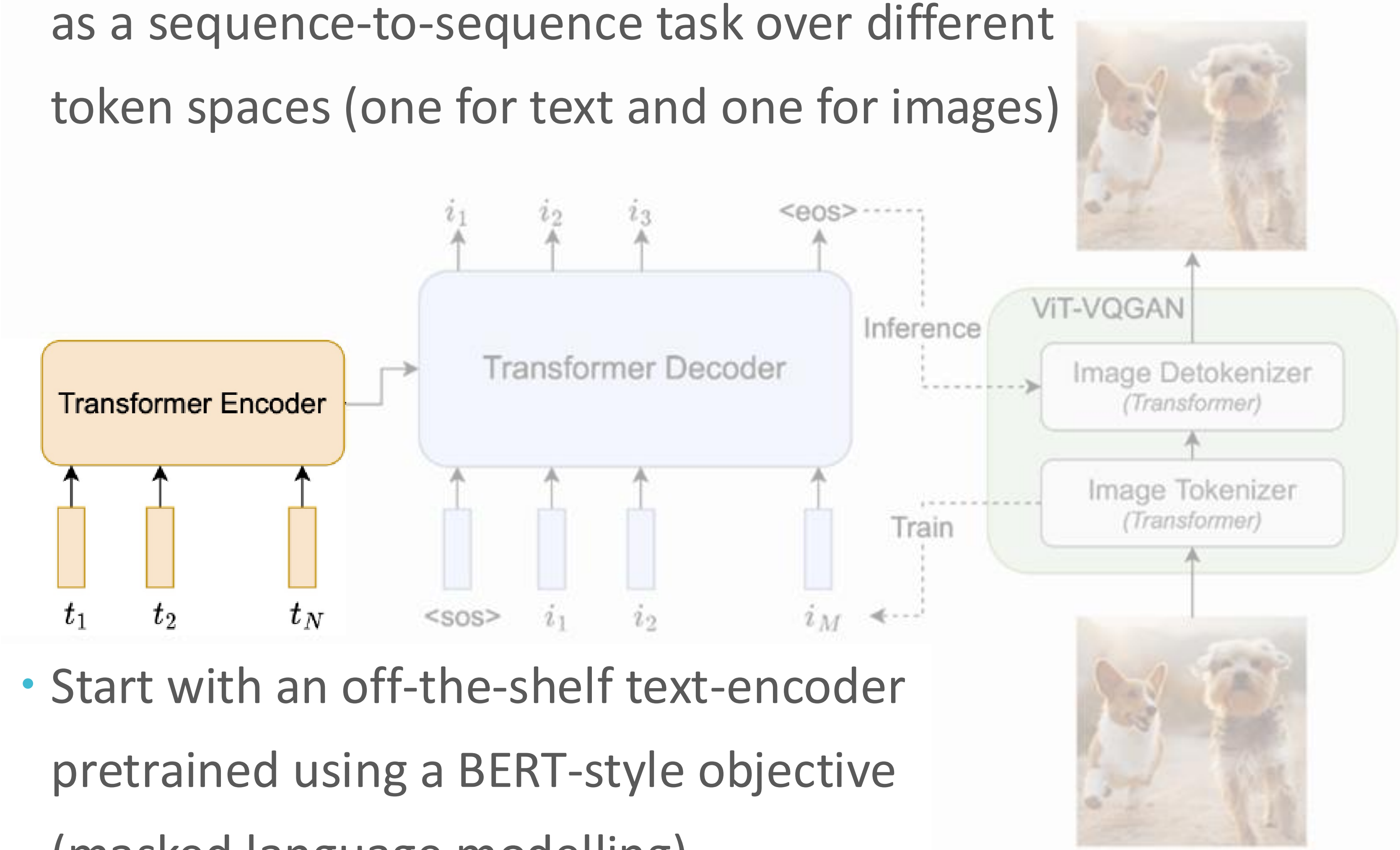
Pathways Autoregressive Text-to-Image (Parti): Step 1. Image Tokenization



Source: https://arxiv.org/pdf/2110.04627

Pathways Autoregressive Text-to-Image (Parti): Step 2. Training

- Idea: treat the task of text-to-image generation as a sequence-to-sequence task over different token spaces (one for text and one for images)

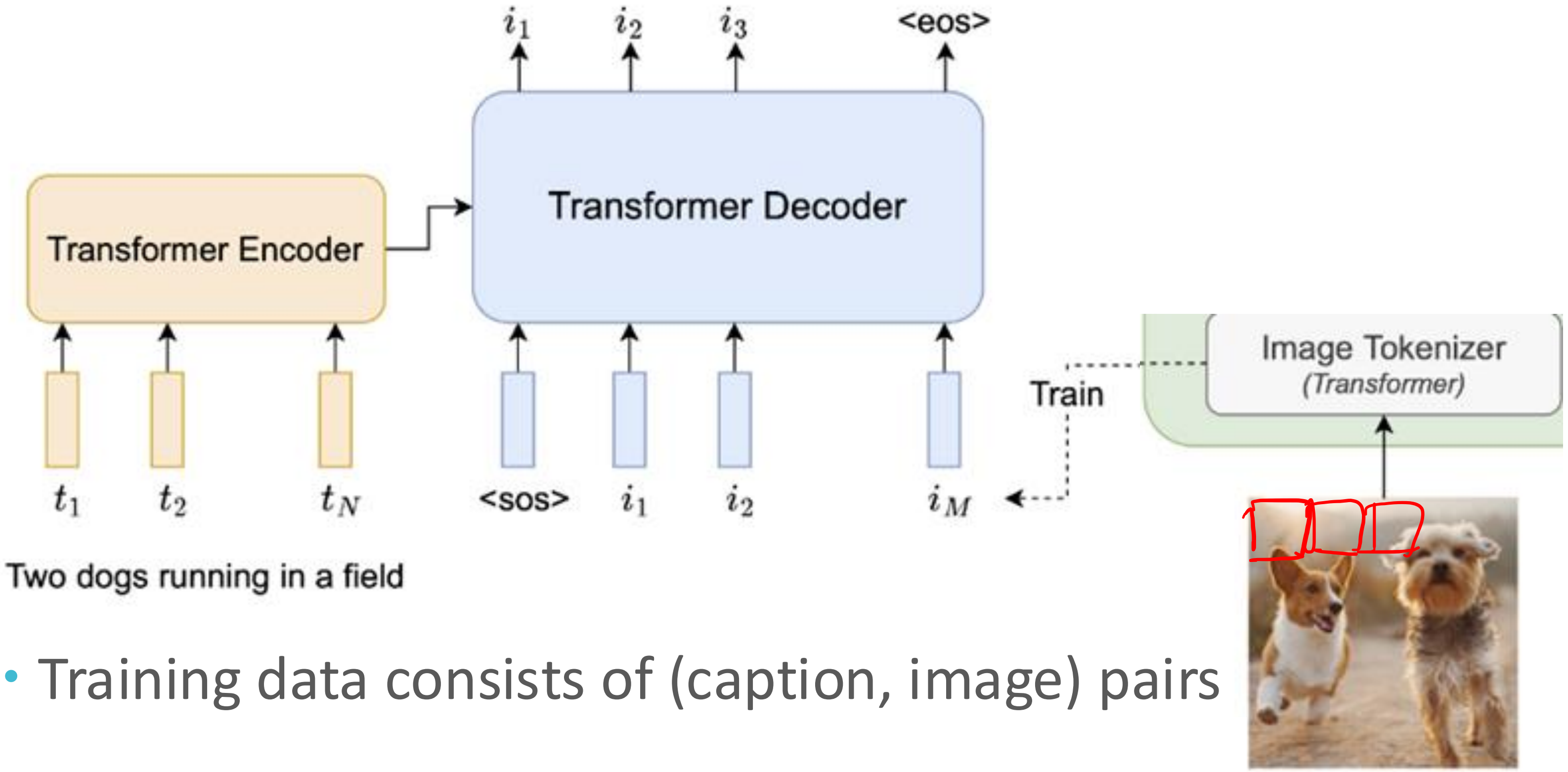


- Start with an off-the-shelf text-encoder pretrained using a BERT-style objective (masked language modelling)

Source: https://arxiv.org/pdf/2206.10789

Pathways Autoregressive Text-to-Image (Parti): Step 2. Training

- Idea: treat the task of text-to-image generation as a sequence-to-sequence task over different token spaces (one for text and one for images)

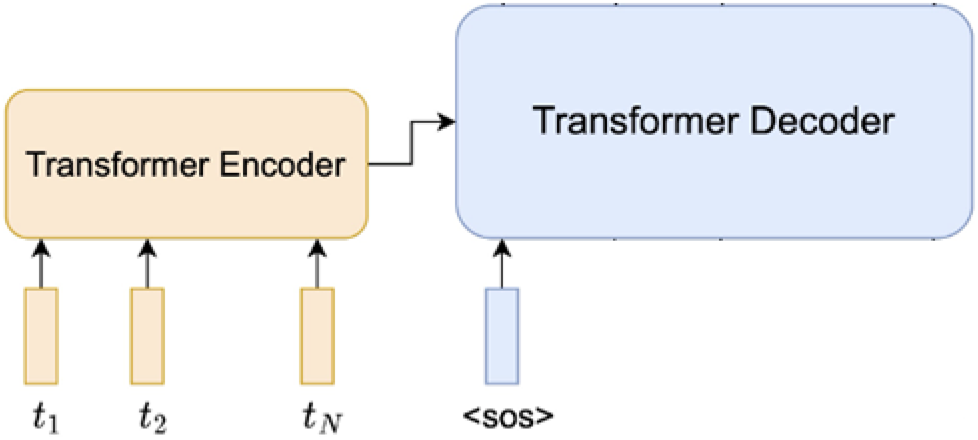


- Training data consists of (caption, image) pairs
- Images are tokenized and the decoder is trained to predict the next image-token

Source: https://arxiv.org/pdf/2206.10789

Pathways Autoregressive Text-to-Image (Parti): Step 3. Generation

- Idea: treat the task of text-to-image generation as a sequence-to-sequence task over different token spaces (one for text and one for images)



B. *A portrait of a statue of the Egyptian god Anubis wearing aviator goggles, white t-shirt and leather jacket. The city of Los Angeles is in the background. Hi-res DSLR photograph.*

- To perform generation, tokens are sampled from the decoder iteratively until the EOS token is generated. Then the sequence is then passed into the trained detokenizer.

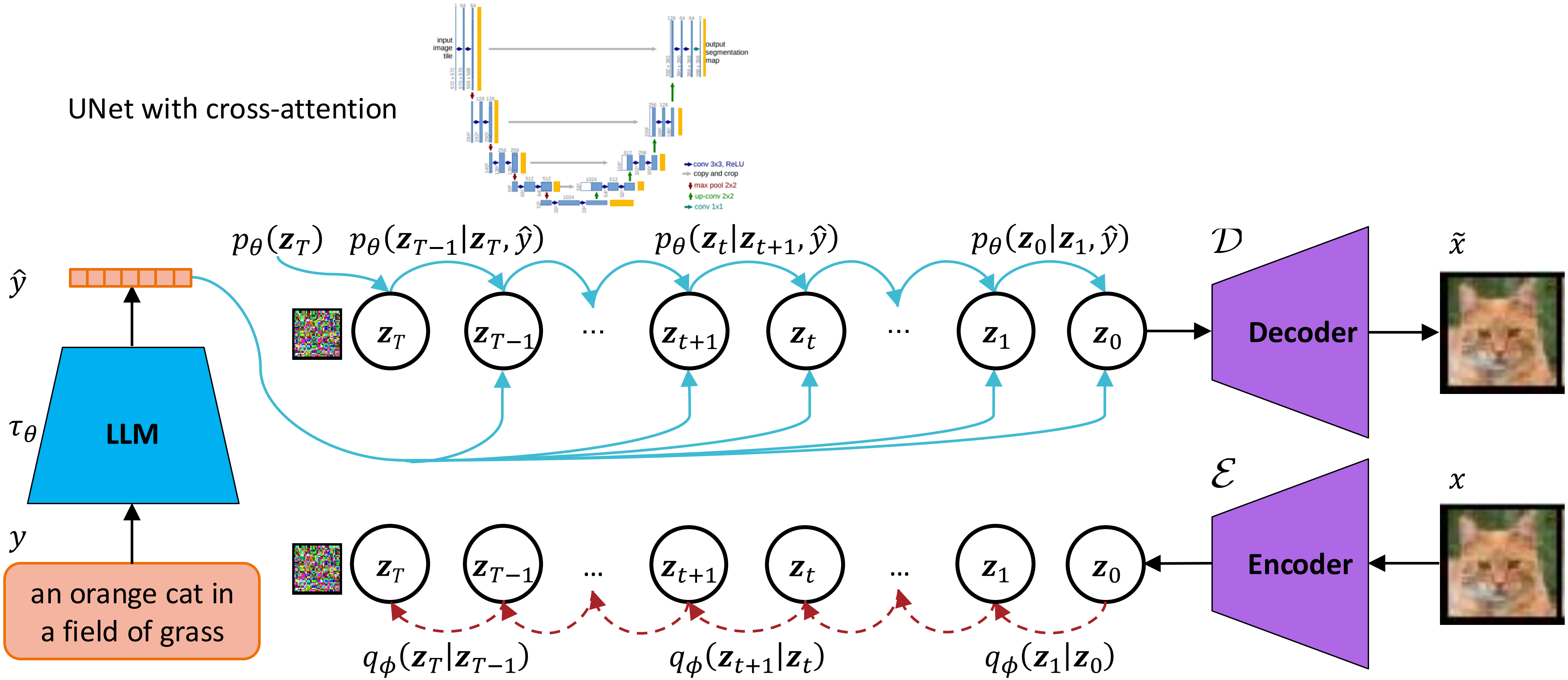
Source: https://arxiv.org/pdf/2206.10789

Latent Diffusion Models

- Issue: diffusion models typically operate in pixel space where training and inference are both *incredibly* slow
  - Training:
    - Guided Diffusion: 150 – 1000 V100 days
    - Imagen: 256 TPU-v4s for 4 days = 1000 TPU days
  - Inference:
    - Guided Diffusion: 50k samples in 5 days on A100

Latent Diffusion Models

- Issue: diffusion models typically operate in pixel space where training and inference are both *incredibly* slow
- Idea: instead of working in pixel space, first project the images down to some **lower-dimensional latent space**, then fit a diffusion model in this latent space
    - This also makes *conditioning* the diffusion model on arbitrary vector inputs y (e.g., embedded captions) much faster
    - Conditioning can be done via **cross-attention** in the UNet layers

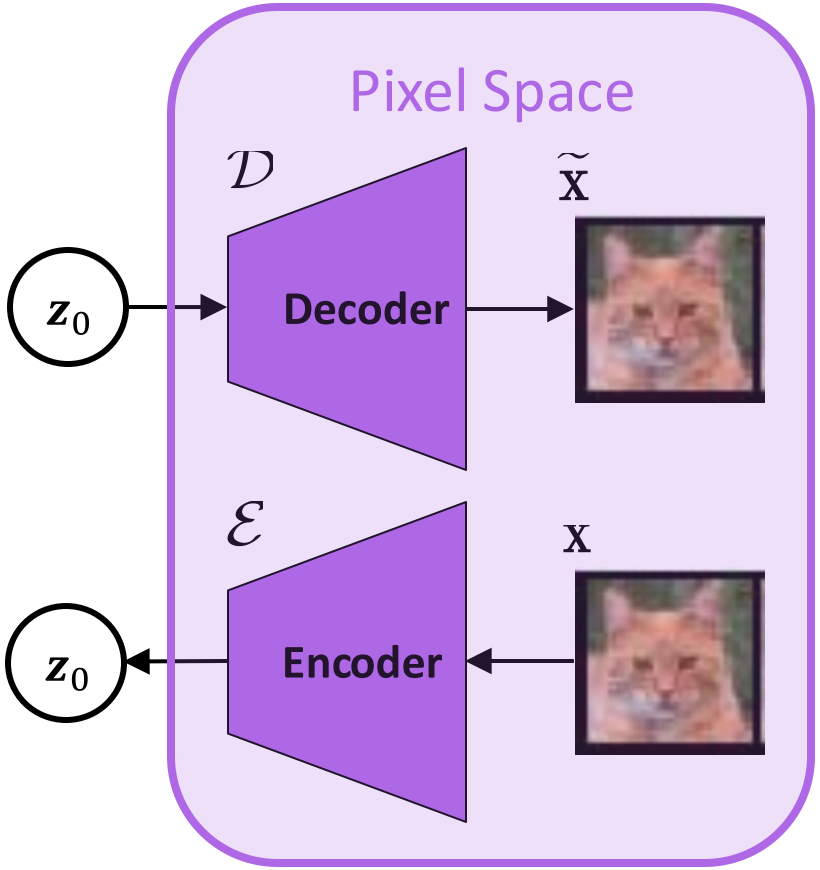


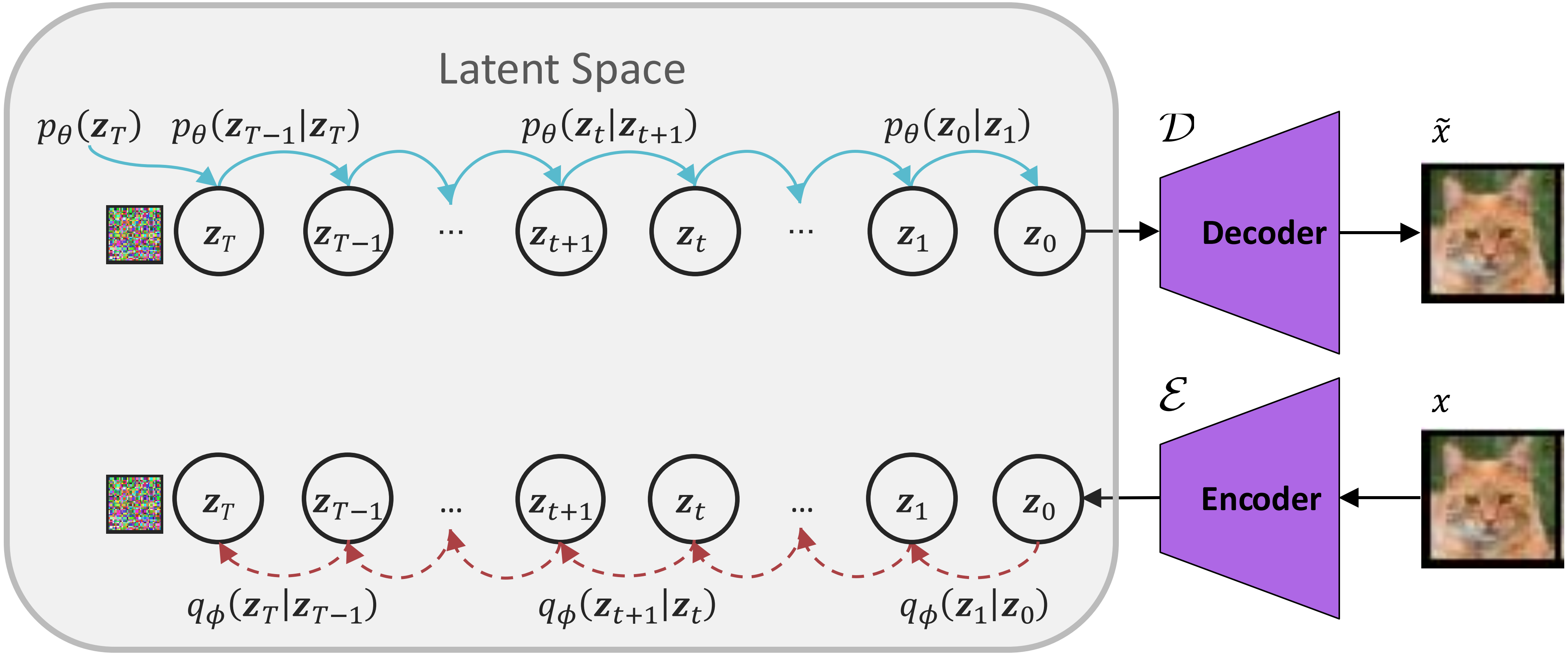
Latent Diffusion Models

Figure courtesy of Matt Gormley

LDMs: Autoencoder

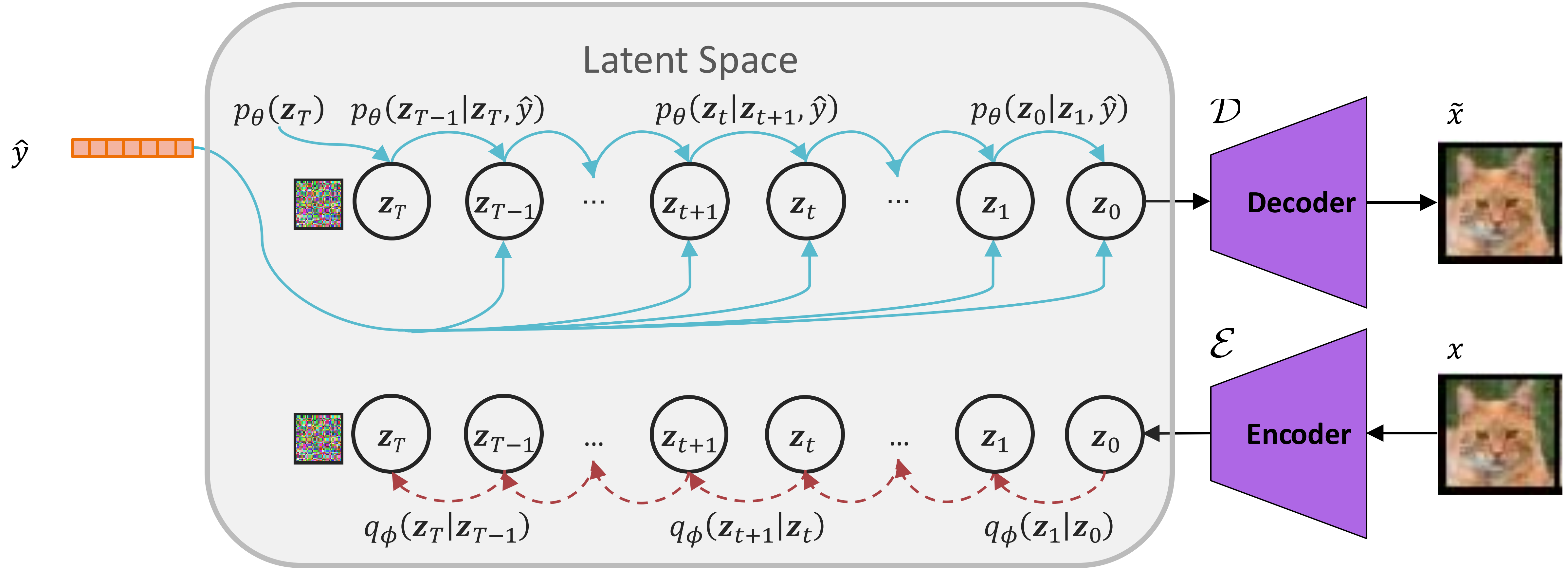
- The autoencoder projects high dimensional images (e.g., 1024x1024 pixels) down to a lower-dimensional **latent space** and faithfully projects back up to **pixel space**
- The original LDM paper considered two options:
  1. a VAE-like model (regularizes the latent distribution towards a Gaussian)
  2. a VQGAN (performs vector quantization in the decoder i.e., uses a discrete codebook)
  - This model is trained ahead of time just on raw images and then kept frozen while training the LDM





LDMs: DDPM

Figure courtesy of Matt Gormley

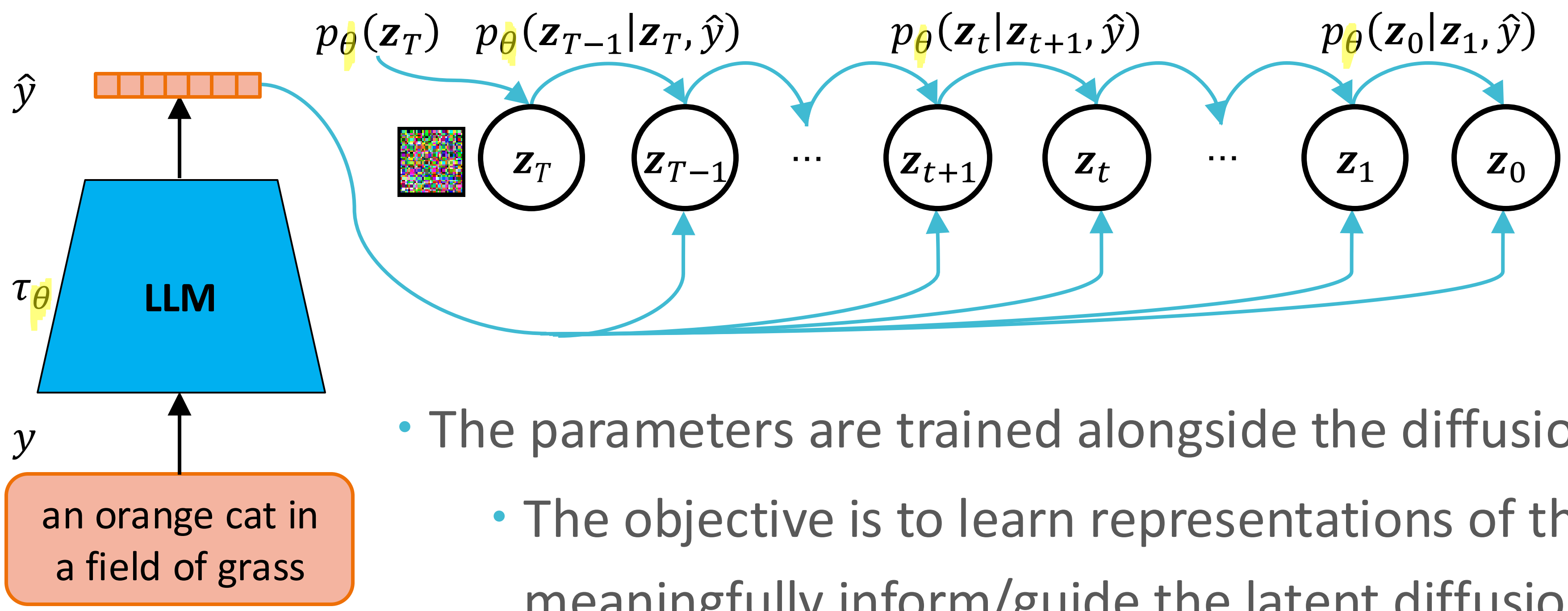


LDMs: Conditioning

Figure courtesy of Matt Gormley

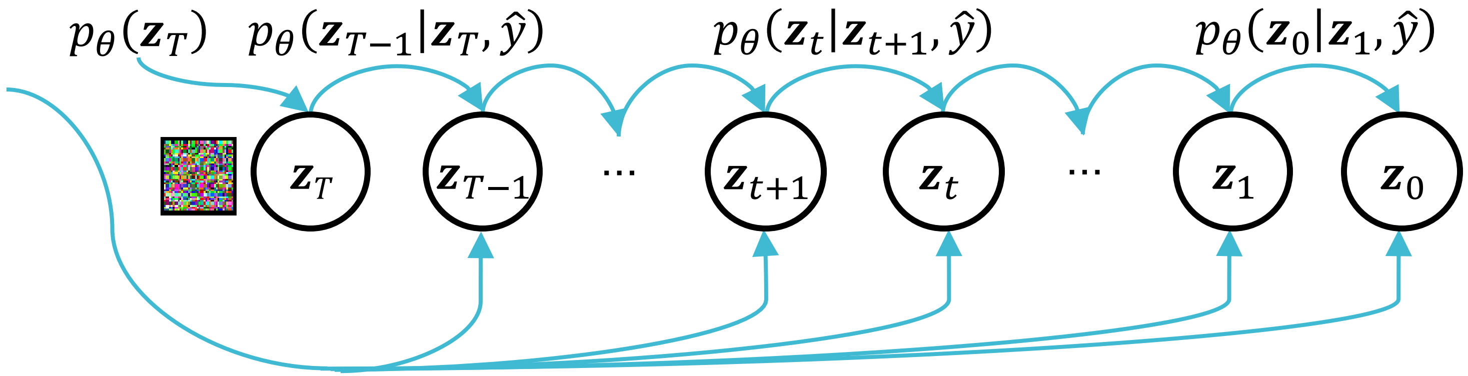
LDMs: Prompt Model

- The prompt model is just an encoder-only transformer



- The parameters are trained alongside the diffusion model's parameters
  - The objective is to learn representations of the text prompts that meaningfully inform/guide the latent diffusion model

Figure courtesy of Matt Gormley



(Learned) Reverse Process:

$$p_\theta(\mathbf{z}_{1:T}) = p_\theta(\mathbf{z}_T) \prod_{t=1}^{T} p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y))$$

$$p_\theta(\mathbf{z}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y)) \sim \mathcal{N}(\mu_\theta(\mathbf{z}_t, t, \tau_\theta(y)), \mathbf{\Sigma}_\theta(\mathbf{z}_t, t))$$

LDMs: Prompt Model

Figure courtesy of Matt Gormley

Recall: Parameterizing the Learned Reverse Process

- $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \sim \mathcal{N}\left(\mu_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t)\right)$
- Idea #1: Rather than learn $\Sigma_\theta(\boldsymbol{x}_t, t)$, just use what we know about $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \sim \mathcal{N}\left(\tilde{\mu}_q(\boldsymbol{x}_t, \boldsymbol{x}_0), \sigma_t^2 I\right)$ and set

$$\Sigma_\theta(\boldsymbol{x}_t, t) = \sigma_t^2 I$$

- Idea #2: We want $\mu_\theta(\boldsymbol{x}_t, t)$ to be close to $\tilde{\mu}_q(\boldsymbol{x}_t, \boldsymbol{x}_0)$
  - Option C: Learn a network that approximates the $\boldsymbol{\epsilon}$ that gave rise to $\boldsymbol{x}_t$ from $\boldsymbol{x}_0$ in the forward process:

$$\mu_\theta(\boldsymbol{x}_t, t) = \alpha_t^{(0)} \boldsymbol{x}_\theta^{(0)}(\boldsymbol{x}_t, t) + \alpha_t^{(t)} \boldsymbol{x}_t$$

where $\boldsymbol{x}_\theta^{(0)}(\boldsymbol{x}_t, t) = \dfrac{\boldsymbol{x}_t + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)}{\sqrt{\bar{\alpha}_t}}$

where $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) = \mathrm{UNet}_\theta(\boldsymbol{x}_t, t)$

Parameterizing the Learned Conditional Reverse Process

- $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \sim \mathcal{N}\left(\mu_\theta(\boldsymbol{x}_t, t, \tau_\theta(y)), \Sigma_\theta(\boldsymbol{x}_t, t)\right)$
- Idea #1: Rather than learn $\Sigma_\theta(\boldsymbol{x}_t, t)$, just use what we know about $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \sim \mathcal{N}\left(\tilde{\mu}_q(\boldsymbol{x}_t, \boldsymbol{x}_0), \sigma_t^2 I\right)$ and set
$$\Sigma_\theta(\boldsymbol{x}_t, t) = \sigma_t^2 I$$
- Idea #2: We want $\mu_\theta\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right)$ to be close to $\tilde{\mu}_q(\boldsymbol{x}_t, \boldsymbol{x}_0)$
  - Option C: Learn a network that approximates the $\boldsymbol{\epsilon}$ that gave rise to $\boldsymbol{x}_t$ from $\boldsymbol{x}_0$ in the forward process:

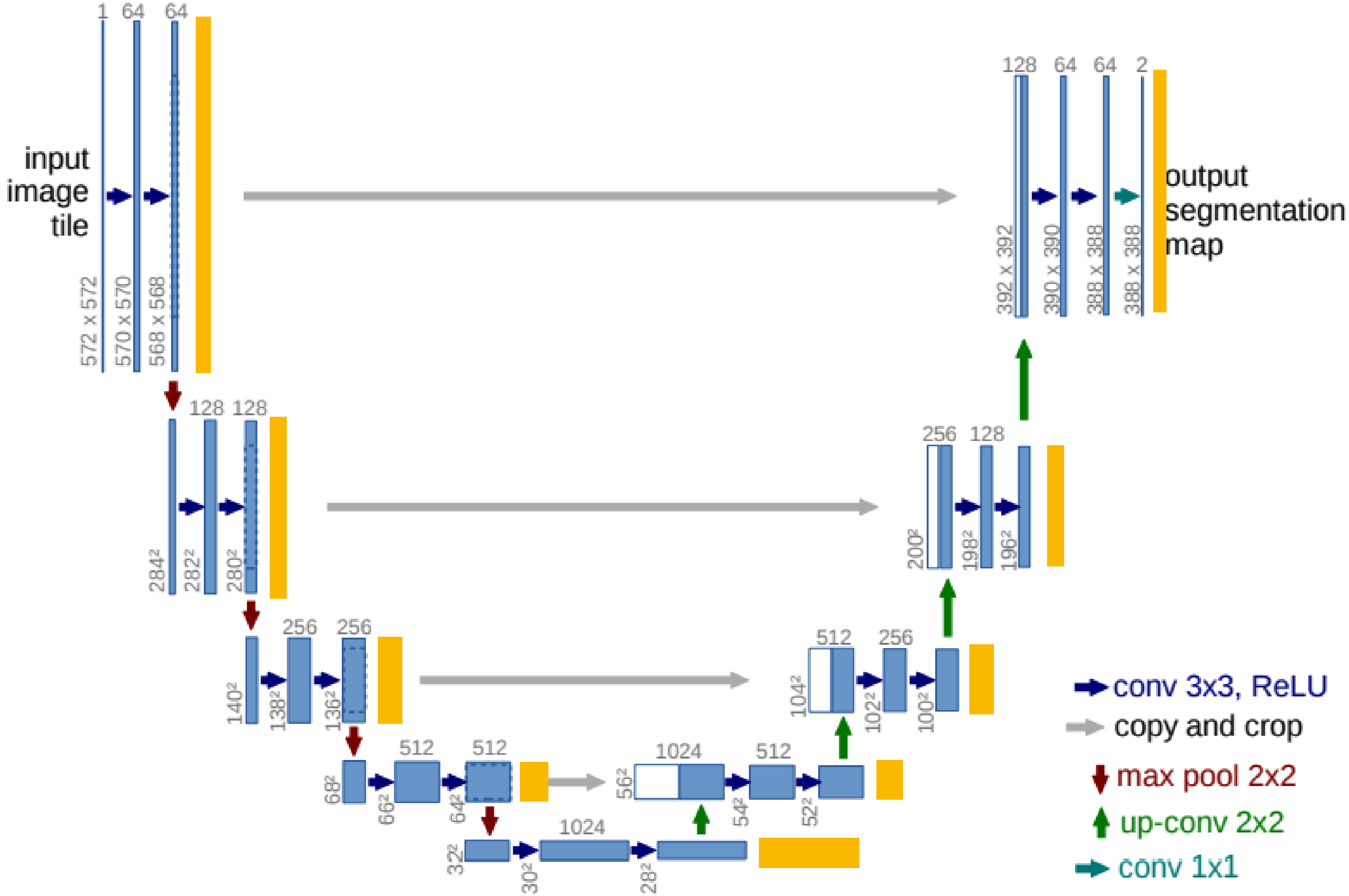
$$\mu_\theta\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right) = \alpha_t^{(0)} \boldsymbol{x}_\theta^{(0)}\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right) + \alpha_t^{(t)} \boldsymbol{x}_t$$

where $\boldsymbol{x}_\theta^{(0)}\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right) = \dfrac{\boldsymbol{x}_t + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right)}{\sqrt{\bar{\alpha}_t}}$

where $\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right) = \mathrm{UNet}_\theta\left(\boldsymbol{x}_t, t, \tau_\theta(y)\right)$

- The noise model includes **cross attention** (yellow boxes) between the UNet layers and the representation of the prompt text
- During training we optimize both the parameters of the UNet noise model and the parameters of the LLM simultaneously

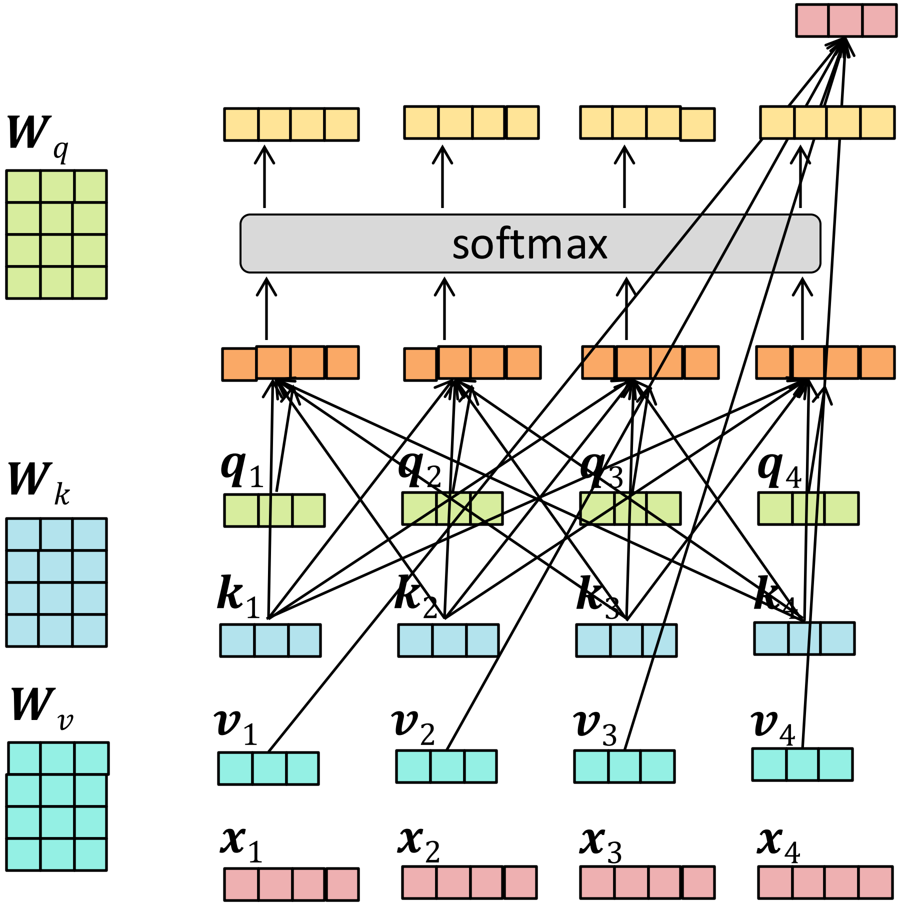
$$\epsilon_\theta(\boldsymbol{x}_t, t, \tau_\theta(y)) = \mathrm{UNet}_\theta(\boldsymbol{x}_t, t, \tau_\theta(y))$$



LDM: Noise Model

Figure courtesy of Matt Gormley

Recall: Scaled Dot-Product Attention



$$X' = AV = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

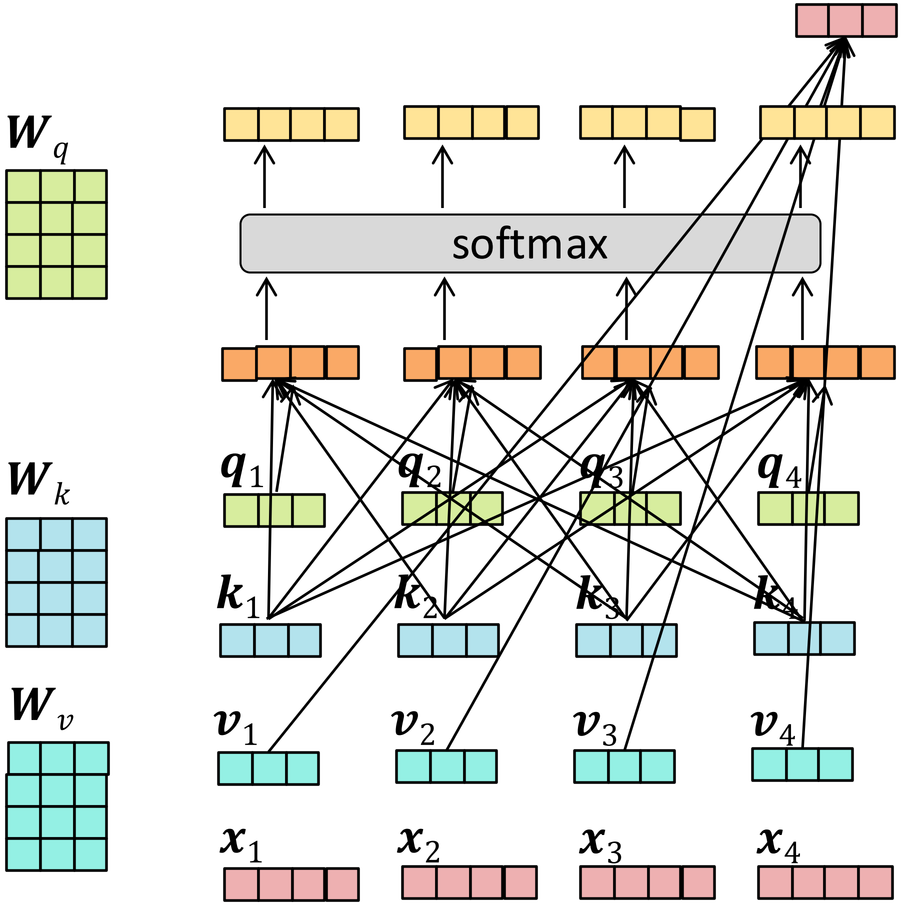
$$A = \text{softmax}(S)$$

$$S = \frac{QK^T}{\sqrt{d_k}}$$

$$Q = X\boldsymbol{W}_q$$

$$K = X\boldsymbol{W}_k$$

$$V = X\boldsymbol{W}_v$$



$$X' = AV = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$A = \text{softmax}(S)$$

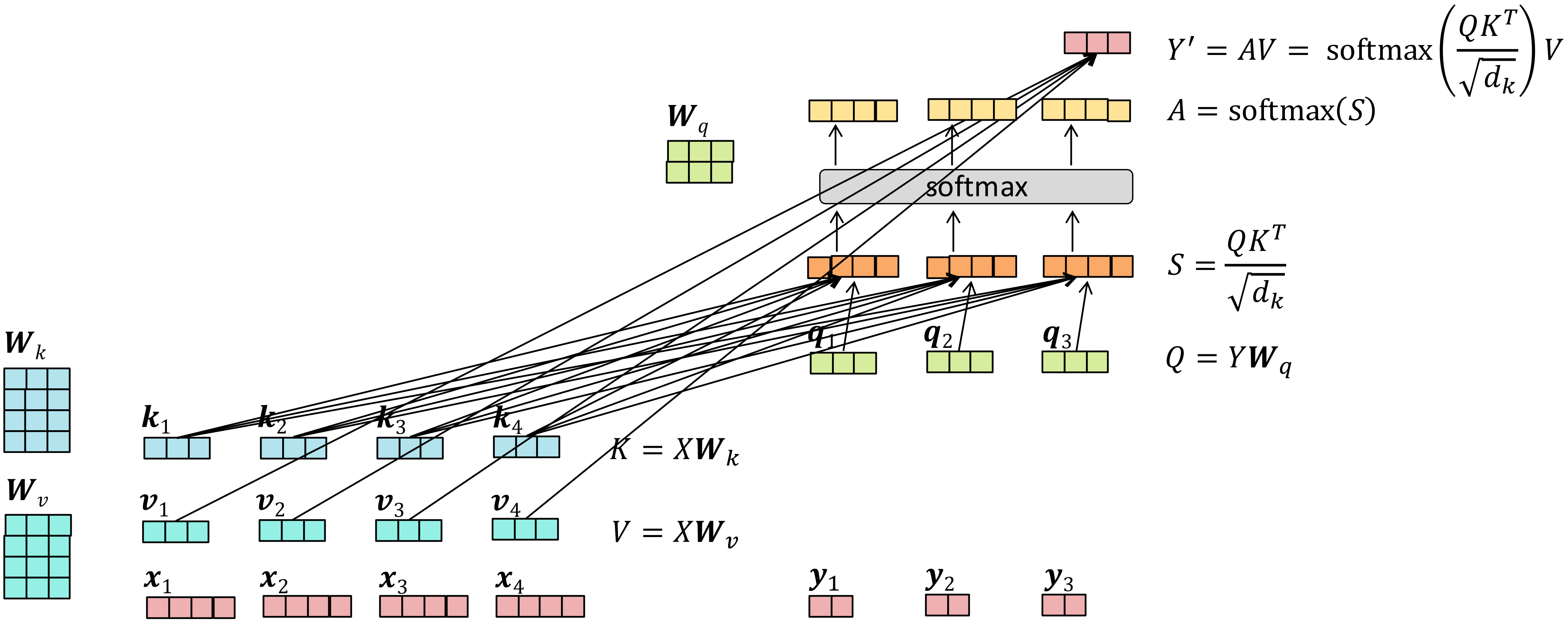
$$S = \frac{QK^T}{\sqrt{d_k}}$$

$$Q = X\boldsymbol{W}_q$$

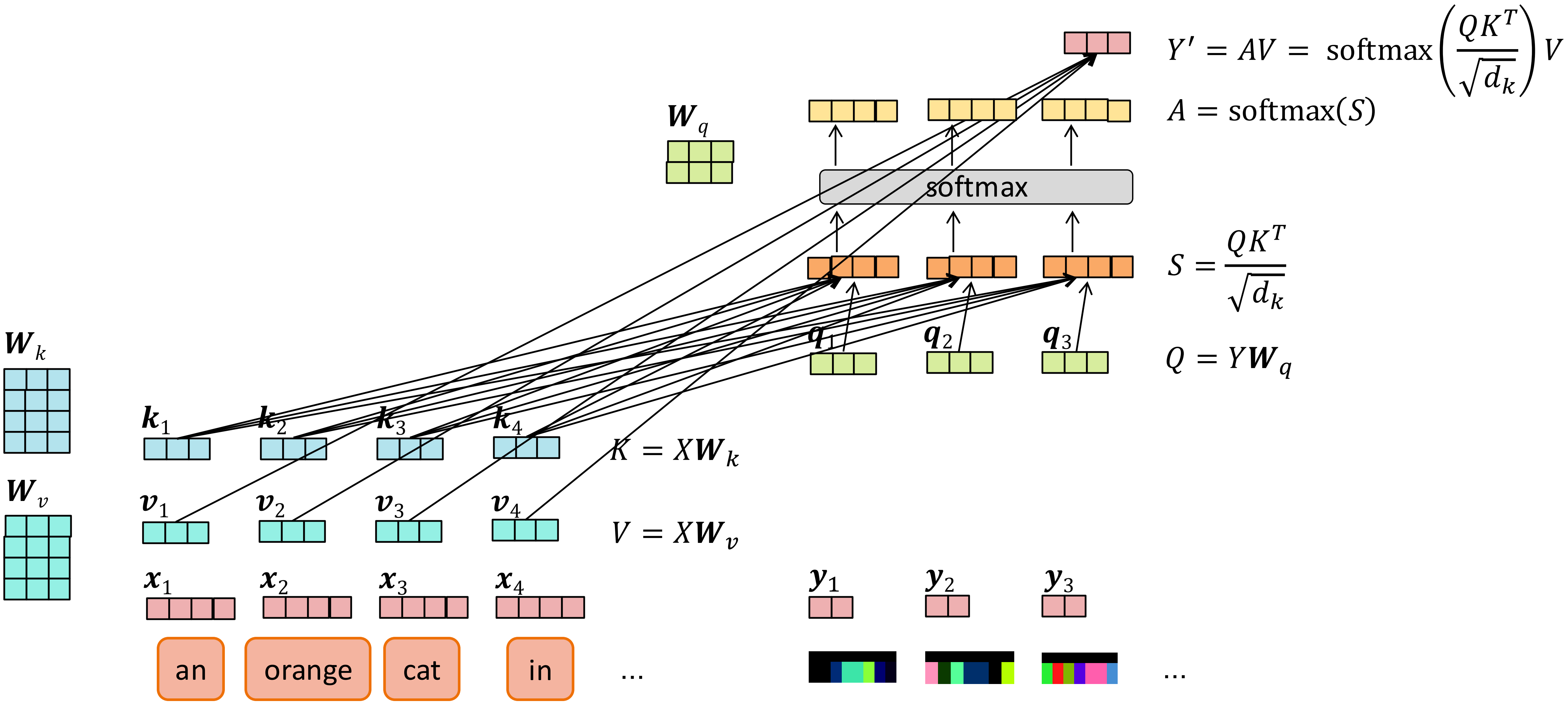
$$K = X\boldsymbol{W}_k$$

$$V = X\boldsymbol{W}_v$$

Self Attention



Cross Attention



LDMs: Cross Attention

LDMs: Cross Attention

- The cross-attention in the UNet is placed within a larger Transformer block

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

$$Q = W_Q^{(i)} \cdot \varphi_i(z_t),\ K = W_K^{(i)} \cdot \tau_\theta(y),\ V = W_V^{(i)} \cdot \tau_\theta(y).$$

Here, $\varphi_i(z_t) \in \mathbb{R}^{N \times d_\epsilon^i}$ denotes a (flattened) intermediate representation of the UNet implementing ϵ_θ and $W_V^{(i)} \in \mathbb{R}^{d \times d_\epsilon^i}$, $W_Q^{(i)} \in \mathbb{R}^{d \times d_\tau}$ & $W_K^{(i)} \in \mathbb{R}^{d \times d_\tau}$ are learnable projection matrices [36, 97].

input	$\mathbb{R}^{h \times w \times c}$
LayerNorm	$\mathbb{R}^{h \times w \times c}$
Conv1x1	$\mathbb{R}^{h \times w \times d \cdot n_h}$
Reshape	$\mathbb{R}^{h \cdot w \times d \cdot n_h}$
$\times T$ { SelfAttention	$\mathbb{R}^{h \cdot w \times d \cdot n_h}$
$\times T$ { MLP	$\mathbb{R}^{h \cdot w \times d \cdot n_h}$
$\times T$ { CrossAttention	$\mathbb{R}^{h \cdot w \times d \cdot n_h}$
Reshape	$\mathbb{R}^{h \times w \times d \cdot n_h}$
Conv1x1	$\mathbb{R}^{h \times w \times c}$

Source: https://arxiv.org/pdf/2112.10752

Text-to-Image Synthesis on LAION. 1.45B Model.

'A street sign that reads "Latent Diffusion" '	'A zombie in the style of Picasso'	'An image of an animal half mouse half octopus'	'An illustration of a slightly conscious neural network'	'A painting of a squirrel eating a burger'	'A watercolor painting of a chair that looks like an octopus'	'A shirt with the inscription: "I love generative models!" '

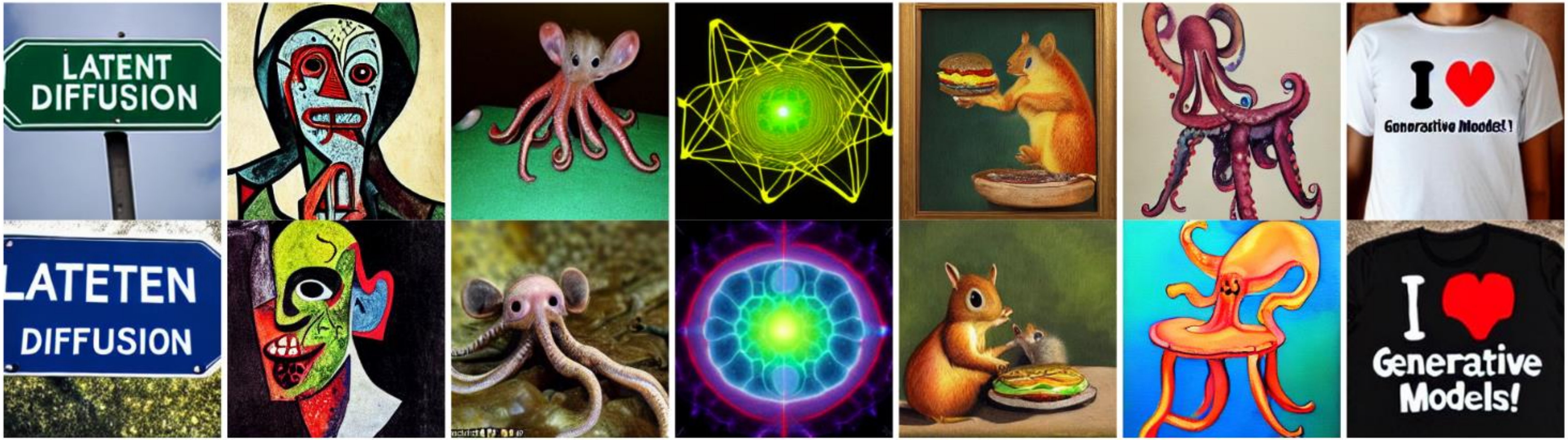


Figure 5. Samples for user-defined text prompts from our model for text-to-image synthesis, *LDM-8 (KL)*, which was trained on the LAION [78] database. Samples generated with 200 DDIM steps and $\eta = 1.0$. We use unconditional guidance [32] with $s = 10.0$.

LDMs: Results

Source: https://arxiv.org/pdf/2112.10752

LDMs: Results

Source: https://arxiv.org/pdf/2112.10752

LDMs: Results

- **Key takeaway:** LDMs can generate very high-quality images (in terms of FID / IS scores) with many fewer parameters than competing models because the most computationally intensive step happens in low dimensional latent space, instead of high dimensional pixel space

Text-Conditional Image Synthesis				
Method	FID ↓	IS↑	N_{params}	
CogView† [17]	27.10	18.20	4B	self-ranking, rejection rate 0.017
LAFITE† [109]	26.94	<u>26.02</u>	75M	
GLIDE* [59]	<u>12.24</u>	-	6B	277 DDIM steps, c.f.g. [32] $s = 3$
Make-A-Scene* [26]	**11.84**	-	4B	c.f.g for AR models [98] $s = 5$
LDM-KL-8	23.31	$20.03_{\pm 0.33}$	1.45B	250 DDIM steps
*LDM-KL-8-G**	12.63	$\mathbf{30.29}_{\pm 0.42}$	1.45B	250 DDIM steps, c.f.g. [32] $s = 1.5$

Table 2. Evaluation of text-conditional image synthesis on the 256 × 256-sized MS-COCO [51] dataset: with 250 DDIM [84] steps our model is on par with the most recent diffusion [59] and autoregressive [26] methods despite using significantly less parameters. †/*:Numbers from [109]/ [26]

Source: https://arxiv.org/pdf/2112.10752