Machine Learning Department
School of Computer Science
Carnegie Mellon University

Direct Preference Optimization (DPO)

+

Latent Diffusion Models

(and other text-to-image models)

Matt Gormley & Pat Virtue

Lecture 12

Feb. 24, 2025

Reminders

- **Quiz 3**
  - **In class, Wed, Feb 26**
  - **Lectures 9, 10, 11, and only RLHF/DPO portion of 12**
- **Homework 3: Applying and Adapting LLMs**
  - **Out: Sun, Feb 23**
  - **Due: Thu, March 13 at 11:59pm**
  - **You are *not* expected to work on HW3 over Spring Break**

Project

- Goals:
  - Explore a generative modeling technique of your choosing
  - Deeper understanding of methods in real-world application
  - Work in teams of 3 students
- Project description on course website

RLHF (CONTINUED)

RLHF

Recall...

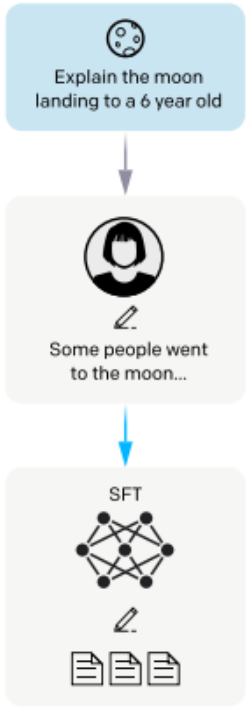
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



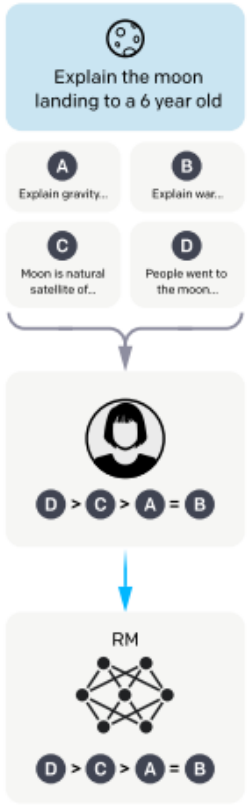
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

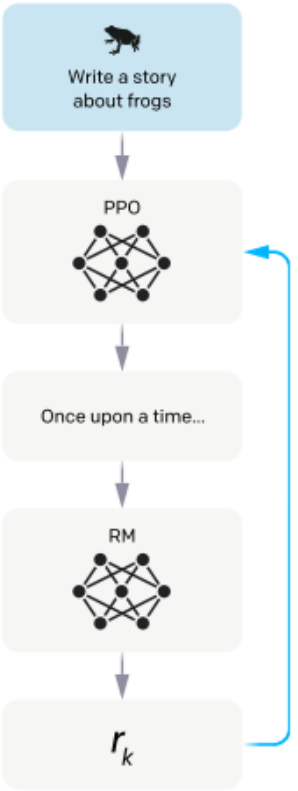
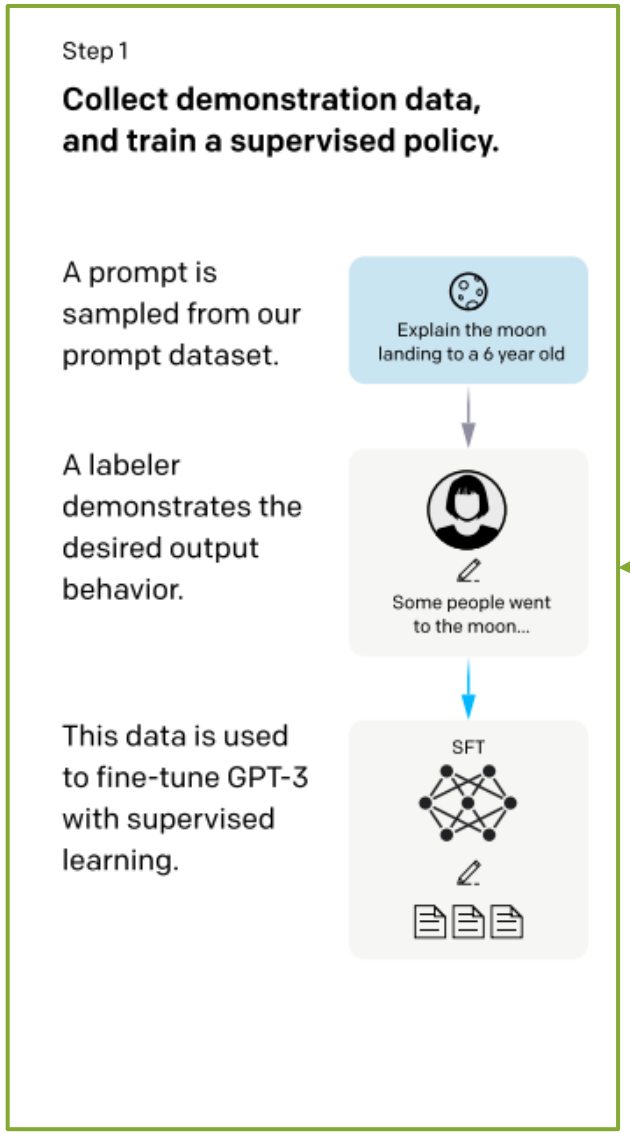


Figure from https://arxiv.org/pdf/2203.02155.pdf

RLHF

Recall…



- Step 1 performs instruction fine-tuning on 13k training examples
- This aligns the model behavior with what we would expect of a chat agent
- But the diversity of the interactions might still be limited by the contents of the training data

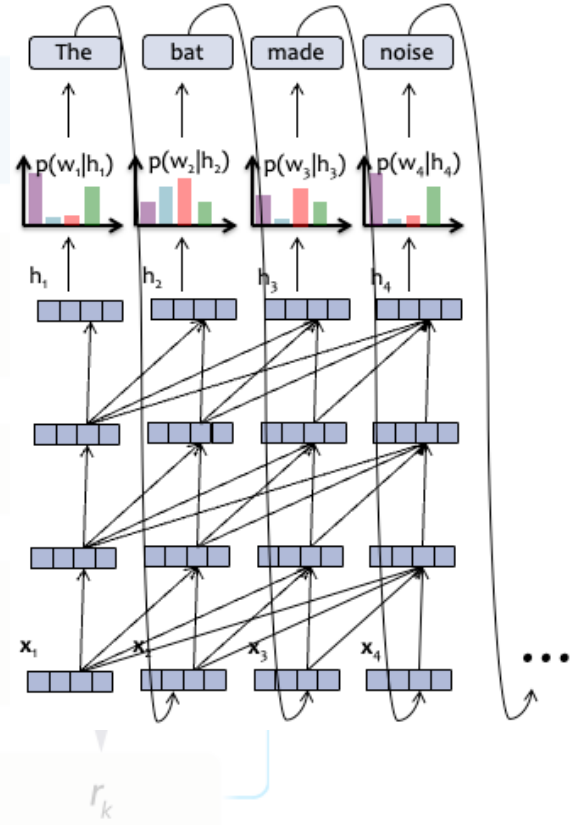


Figure from https://arxiv.org/pdf/2203.02155.pdf

RLHF

Recall…

Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

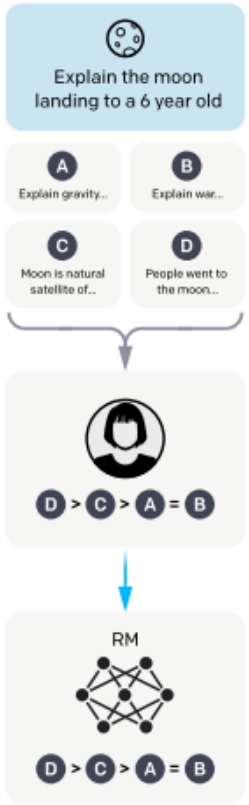
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



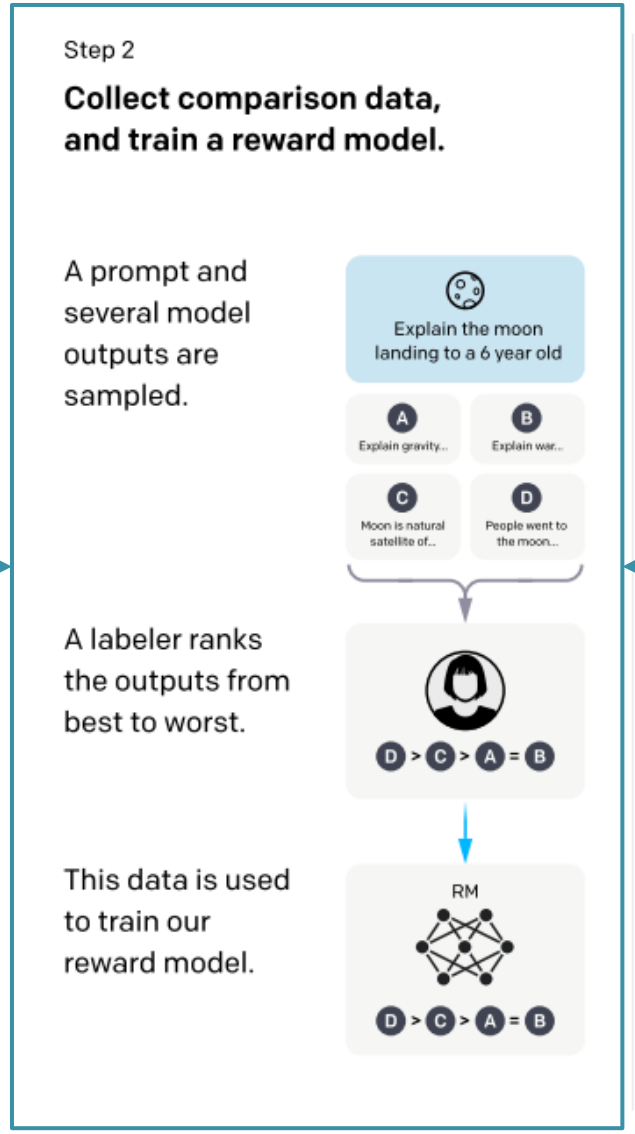
- In Step 2, takes 33k prompts and samples a collection of responses from the instruction fine-tuned model for each one
- The human labeler ranks the K ∈ {4,…,9} responses

Figure from https://arxiv.org/pdf/2203.02155.pdf

RLHF

Recall…

- The reward model is a copy of the Step-1 LLM, but with the softmax over words replaced so that it outputs a single scalar value, i.e. the reward
- The model is trained so that rewards of the higher ranking (winning) responses are larger than those of the lower ranking (losing) responses



- In Step 2, takes 33k prompts and samples a collection of responses from the instruction fine-tuned model for each one
- The human labeler ranks the $K \in \{4,\ldots,9\}$ responses

Figure from https://arxiv.org/pdf/2203.02155.pdf

RLHF

Recall…

Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

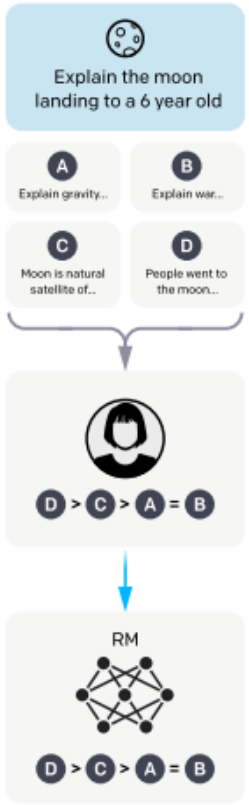
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Figure from https://arxiv.org/pdf/2203.02155.pdf

RLHF

Recall…

- Step 3 trains the model from Step 1 using reinforcement learning
- Instead of having a human or some expert model provide rewards, we take the reward model from Step 2 as ”ground truth” for the rewards
- Reinforcement learning uses (state, action, reward) tuples as training data
  - state = prompt
  - action = response
  - reward = scalar from regression reward model
  - each episode lasts exactly one turn
- RL objective is combined with pre-training objective:

$$\text{objective}(\phi) = \mathbb{E}_{(x,y)\sim D_{\pi_\phi^{RL}}}\left[r_\theta(x,y) - \beta \log\left(\frac{\pi_\phi^{RL}(y|x)}{\pi_\phi^{SFT}(y|x)}\right)\right] + \gamma \mathbb{E}_{x\sim D_{\text{pretrain}}}\left[\log\left(\pi_\phi^{RL}(x)\right)\right]$$

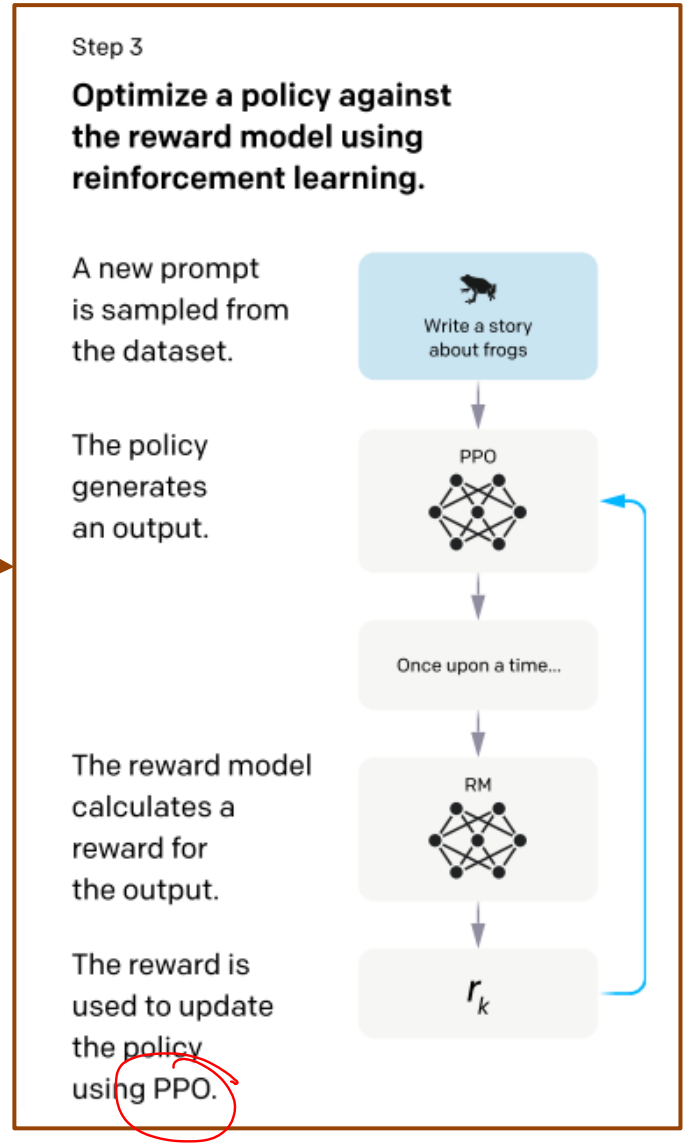


Figure from https://arxiv.org/pdf/2203.02155.pdf

RLHF Objective Function

Recall…

$$\text{objective}(\phi) = \mathbb{E}_{(x,y)\sim D_{\pi_\phi^{RL}}}\left[r_\theta(x,y) - \beta \log\left(\frac{\pi_\phi^{RL}(y|x)}{\pi_\phi^{SFT}(y|x)}\right)\right] + \gamma \mathbb{E}_{x\sim D_{\text{pretrain}}}\left[\log\left(\pi_\phi^{RL}(x)\right)\right]$$

The objective function used here is modeled off of the (rather popular) PPO algorithm. That algorithm, in turn, is a type of policy gradient method and motivated by the objective functions for trust region policy optimization (TRPO). But the (super high level) intuition behind the objective function is as follows:

1. The expectation of the reward says that on samples from the RL trained model π^{RL}, we want the probability of that sample π^{RL} to be high when the reward r_θ is high and for it to be low otherwise.
2. The expectation of the beta term says that we don't want the RL trained model probabilities π^{RL} to stray to far from the supervised fine-tuned (SFT) model π^{SFT} -- this is instantiated as a KL divergence penalty.
3. The expectation under the pretraining distribution $D_{pretrain}$ is just the standard log-likelihood of a training sample that we use for supervised fine-tuning, but applied here to the RL trained model as well.

Note that in practice, we don't compute these expectations exactly, we approximate each with a Monte Carlo approximation (i.e. a sum over a very small number of samples).

(Slides from Henry Chai)

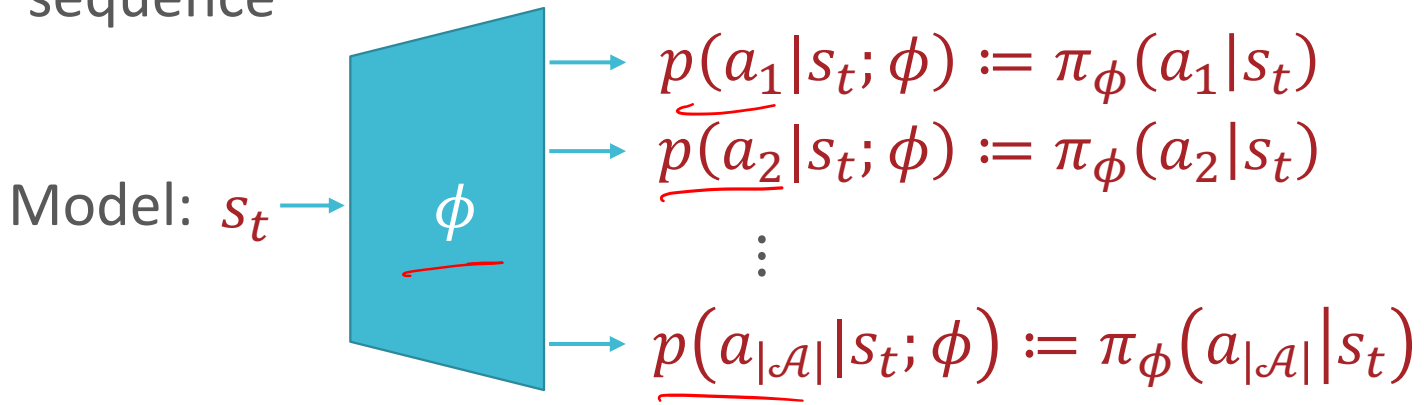
PROXIMAL POLICY OPTIMIZATION

Reinforcement Learning: Problem Formulation for Fine-tuning LLMs

- State space, $\mathcal{S} = \{\text{all possible sequences of tokens}\}$
- Action space, $\mathcal{A} = \{\text{vocabulary of next tokens}\}$
- Reward function
  - Stochastic, $p(r \mid s, a)$
  - Deterministic reward based on reward model trained on human feedback, R_θ
    - R_θ is a bit of weird reward function from an RL perspective: it returns $0 \; \forall \; a \neq \text{EOS}$ and $r_\theta(x, [s, a] - x)$ otherwise
- Transition function
  - Stochastic, $p(s' \mid s, a)$
  - Deterministic, $\delta(s, a) = [s, a]$

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



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

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

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

Policy Gradient Methods

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

$$= -\int R_\theta(\tau) \nabla_\phi \left(\prod_{t=0}^{T-1} \pi_\phi(a_t|s_t)\right) d\tau$$

- Issue: $\nabla_\phi p_\phi(\tau)$ involves taking the gradient of a (hideous) product

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

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

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

$$= -\int R_\theta(\tau) \nabla_\phi \left(\prod_{t=0}^{T-1} \pi_\phi(a_t|s_t)\right) d\tau$$

- Insight:

$$\nabla_\phi p_\phi(\tau) = \frac{p_\phi(\tau)}{p_\phi(\tau)} \nabla_\phi p_\phi(\tau) = p_\phi(\tau) \nabla_\phi \big(\log p_\phi(\tau)\big)$$

$$\log p_\phi(\tau) = \sum_{t=0}^{T-1} \log \pi_\phi(a_t|s_t)$$

$$\nabla_\phi \big(\log p_\phi(\tau)\big) = \sum_{t=0}^{T-1} \nabla_\phi \log \pi_\phi(a_t \mid s_t)$$

Source: https://link.springer.com/article/10.1007/bf00992696

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

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

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

$$= -\int R_\theta(\tau) \nabla_\phi p_\phi(\tau) d\tau = -\int R_\theta(\tau) \nabla_\phi \left(\log p_\phi(\tau)\right) p_\phi(\tau) d\tau$$

$$= -\mathbb{E}_{p_\phi(\tau)}\left[R_\theta(\tau) \nabla_\phi \left(\log p_\phi(\tau)\right)\right]$$

Monte Carlo Estimate

$$\approx -\frac{1}{N} \sum_{n=1}^{N} R_\theta\left(\tau^{(n)}\right) \nabla_\phi \left(\log p_\phi\left(\tau^{(n)}\right)\right)$$

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

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

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)

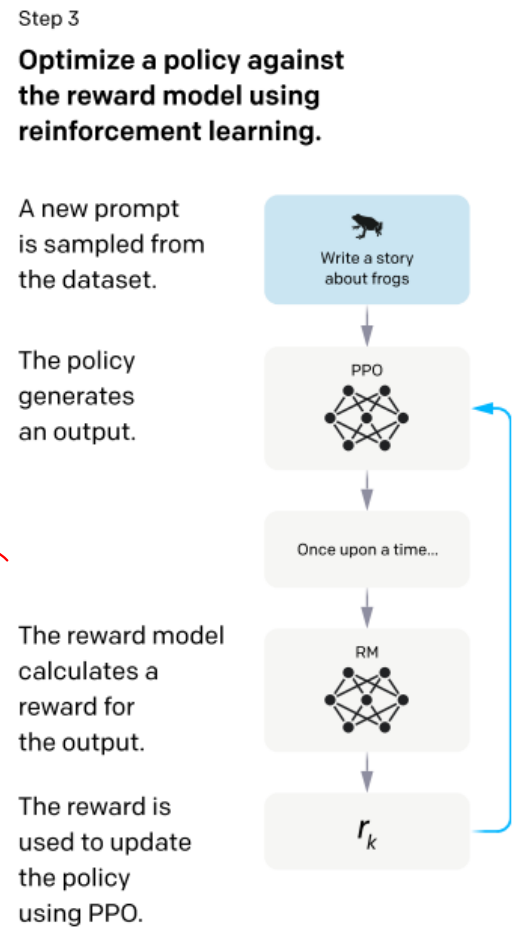
- There are two high-level modifications to get from REINFORCE to proximal policy optimization (PPO):
  2. Policy gradient methods are *on-policy*: the policy being optimized is also being used to generate the trajectories used in training
     - This can also lead to instability/poor convergence if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

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

Reinforcement Learning from Human Feedback: PPO

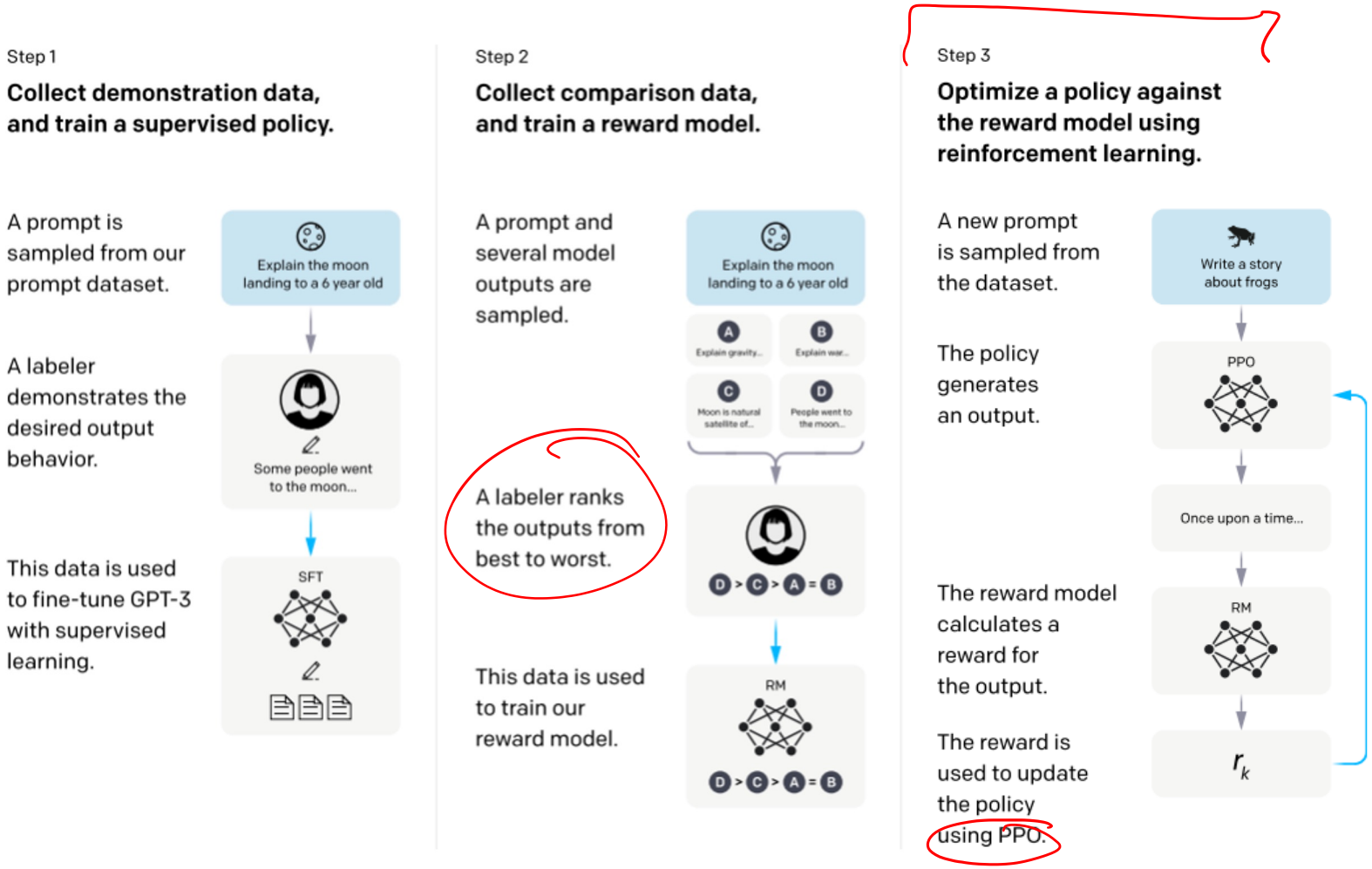
- Step 3 fine-tunes the LLM's parameters using the PPO objective *plus a pre-training loss* term:

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



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

Alright, so what does all of this get us?

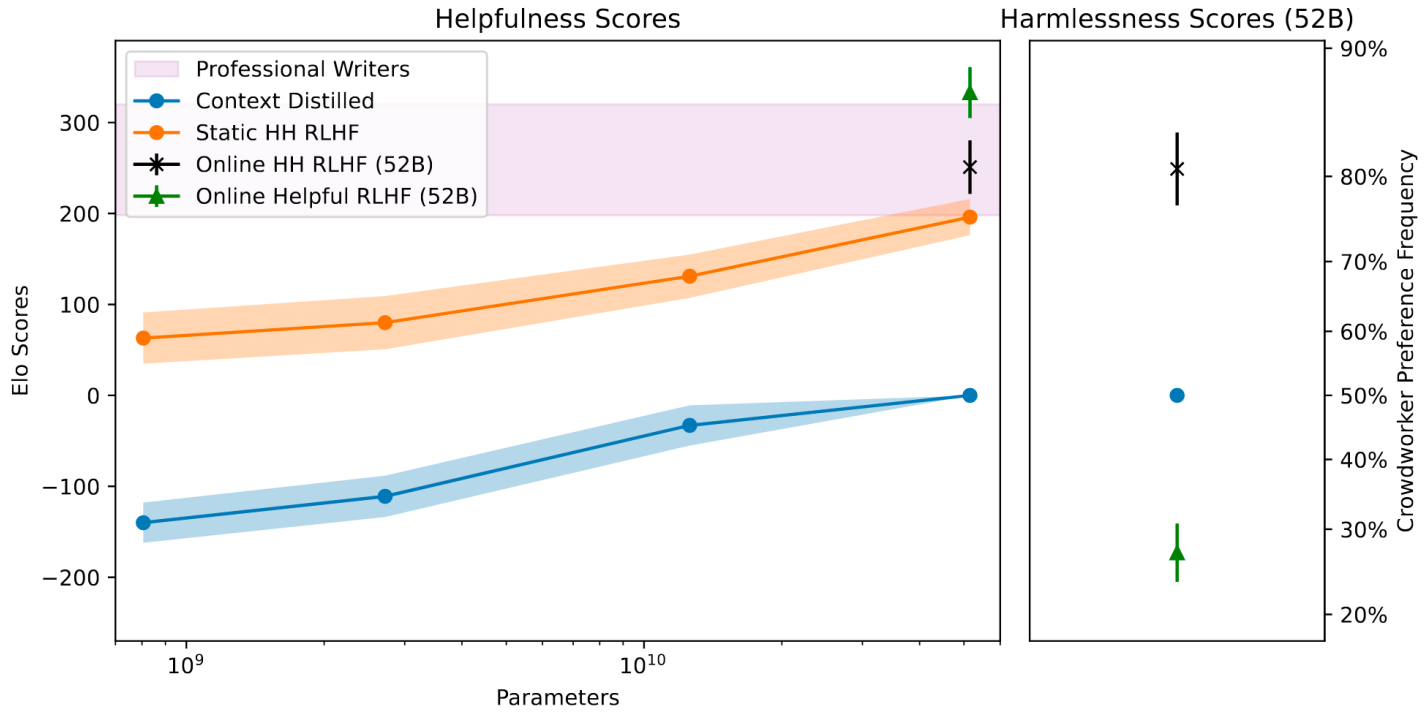


Source: https://arxiv.org/pdf/2203.02155

Reinforcement Learning from Human Feedback: Results

Recall…

- 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)

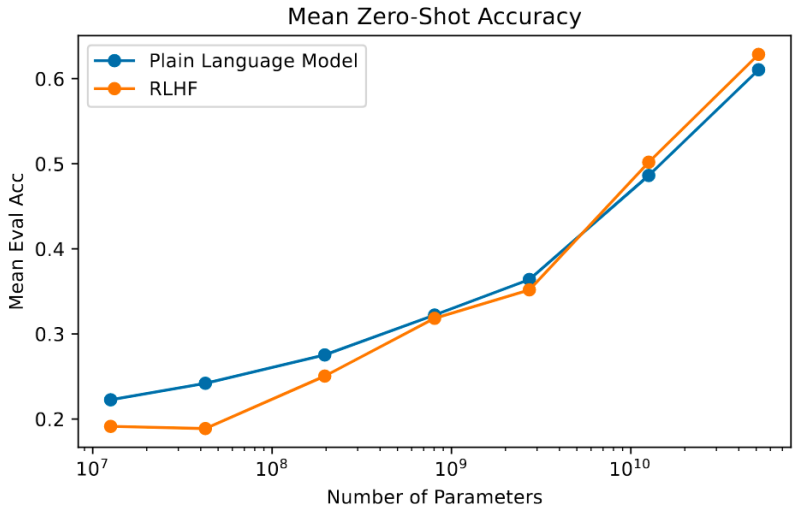


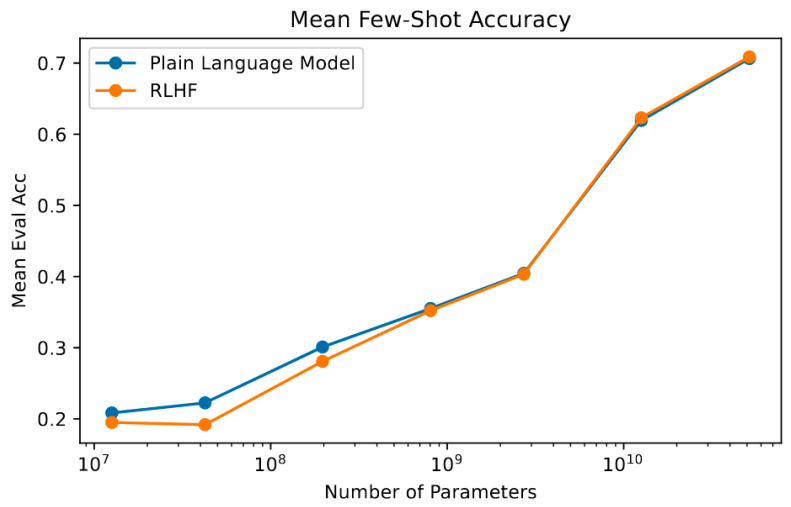
Source: http://arxiv.org/abs/2204.05862

Recall...

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**



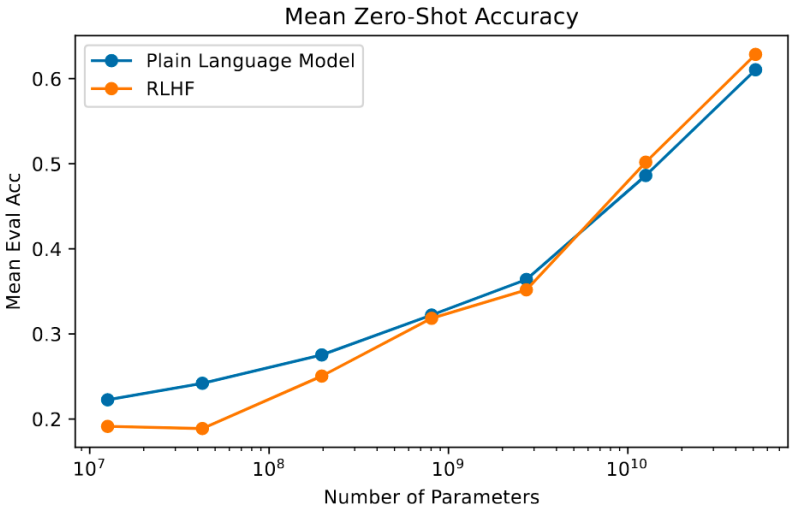


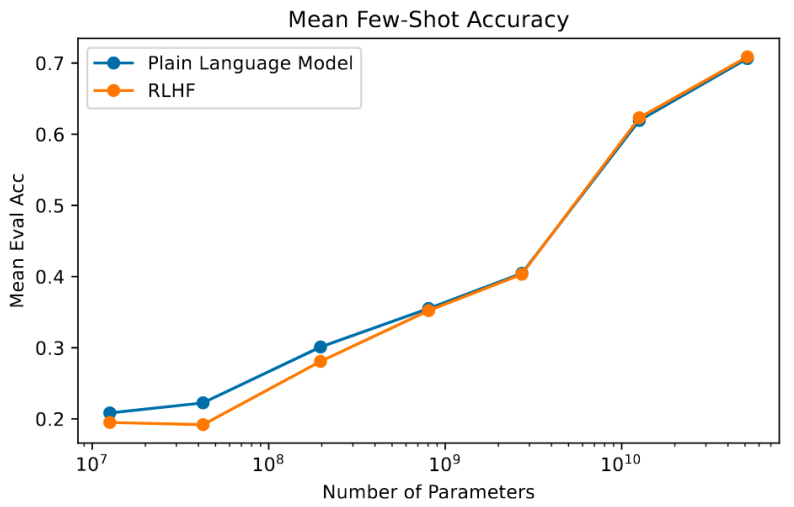
Source: http://arxiv.org/abs/2204.05862

Recall…

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

(Slides from Henry Chai)

DIRECT PREFERENCE OPTIMIZATION

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.

(y_w, y_e)

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

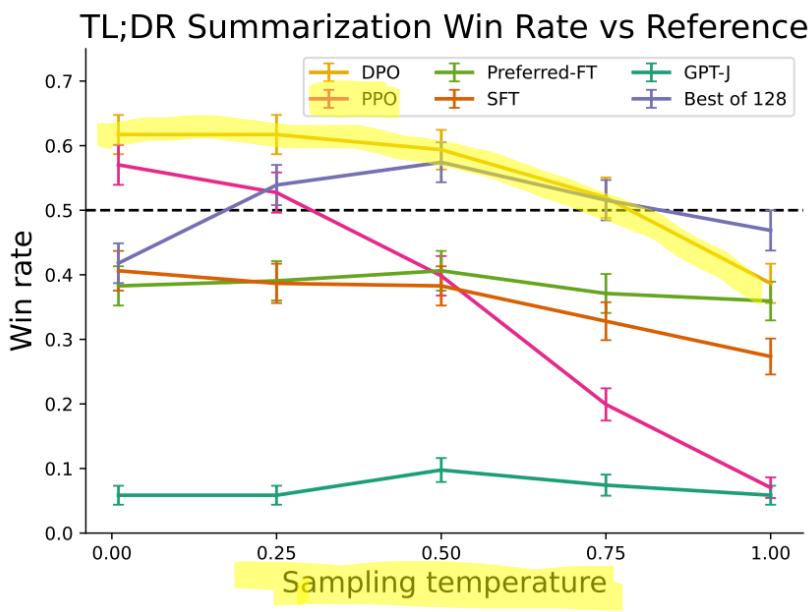
$$p(y_w \succ y_l \mid x) = \frac{1}{1 + \exp\left(\beta \log \frac{\pi_{\phi^*}(y_l|x)}{\pi^{SFT}(y_l|x)} - \beta \log \frac{\pi_{\phi^*}(y_w|x)}{\pi^{SFT}(y_w|x)}\right)}$$

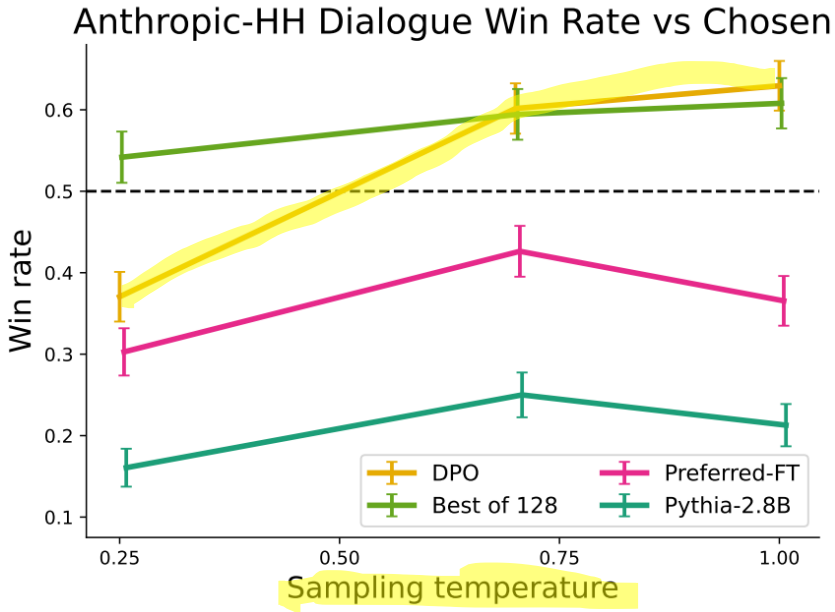
  - "Your language model is secretly a reward model"

- 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

CONDITIONAL IMAGE GENERATION

Image Generation

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

Figure from Razavi et al. (2019)

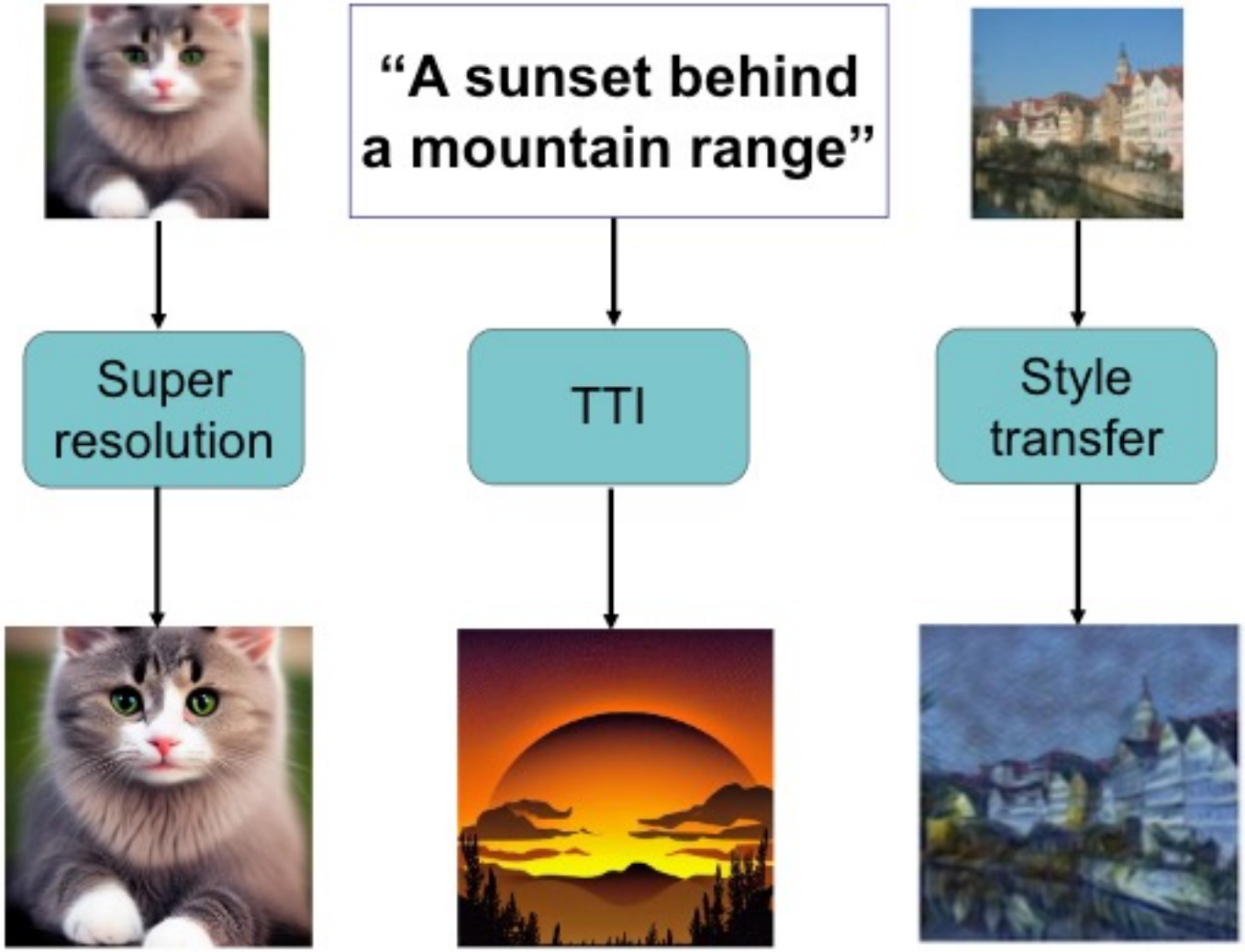


Figure from Bie et al. (2023)

Class Conditional Generation

- **Task**: Given a class label indicating the image type, sample a new image from the model with that type
- Image classification is the problem of taking in an image and predicting its label p(y|x)
- Class conditional generation is doing this in reverse p(x|y)

sea anemone

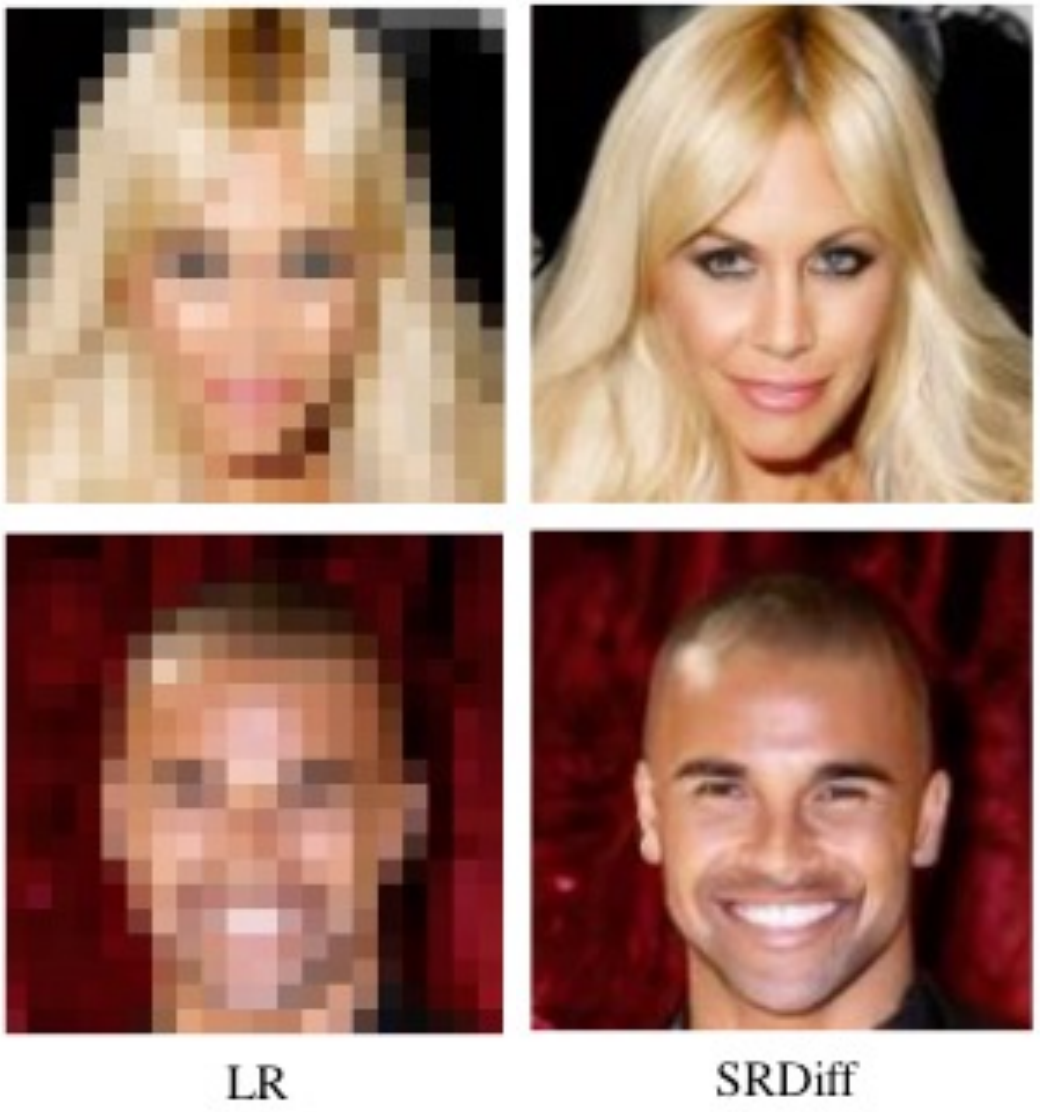
brain coral

slug

goldfinch

Figure from Razavi et al. (2019)

Super Resolution



- Given a low resolution image, generate a high resolution reconstruction of the image
- Compelling on low resolution inputs (see example to the left) but also effective on high resolution inputs

Figure from Li et al. (2021)

Image Editing

A variety of tasks involve automatic editing of an image:

- **Inpainting** fills in the (pre-specified) missing pixels
- **Colorization** restores color to a greyscale image
- **Uncropping** creates a photo-realistic reconstruction of a missing side of an image

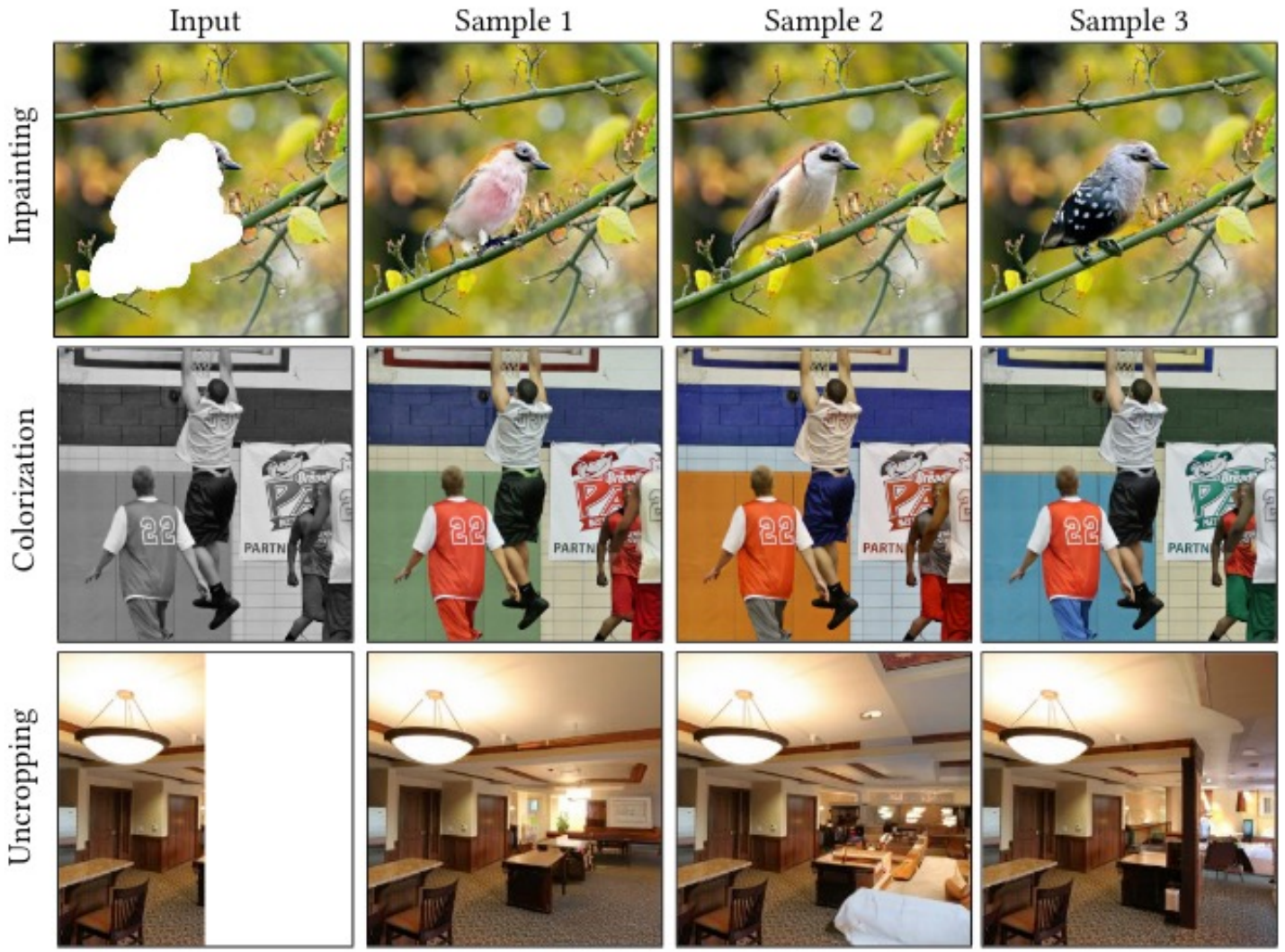


Figure from Saharia et al. (2022)

Style Transfer

- The goal of style transfer is to blend two images
- Yet, the blend should retain the semantic content of the source image presented in the style of another image

Figure from Gatys et al. (2016)

Figure 3. Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork. The original photograph depicting the Neckarfront in Tübingen, Germany, is shown in **A** (Photo: Andreas Praefcke). The painting that provided the style for the respective generated image is shown in the bottom left corner of each panel. **B** *The Shipwreck of the Minotaur* by J.M.W. Turner, 1805. **C** *The Starry Night* by Vincent van Gogh, 1889. **D** *Der Schrei* by Edvard Munch, 1893. **E** *Femme nue assise* by Pablo Picasso, 1910. **F** *Composition VII* by Wassily Kandinsky, 1913.

Text-to-Image Generation

- Given a text description, sample an image that depicts the prompt
- The following images are samples from SDXL with refinement

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

Figure from Podell et al. (2023)

Timeline: Text-to-Image Generation

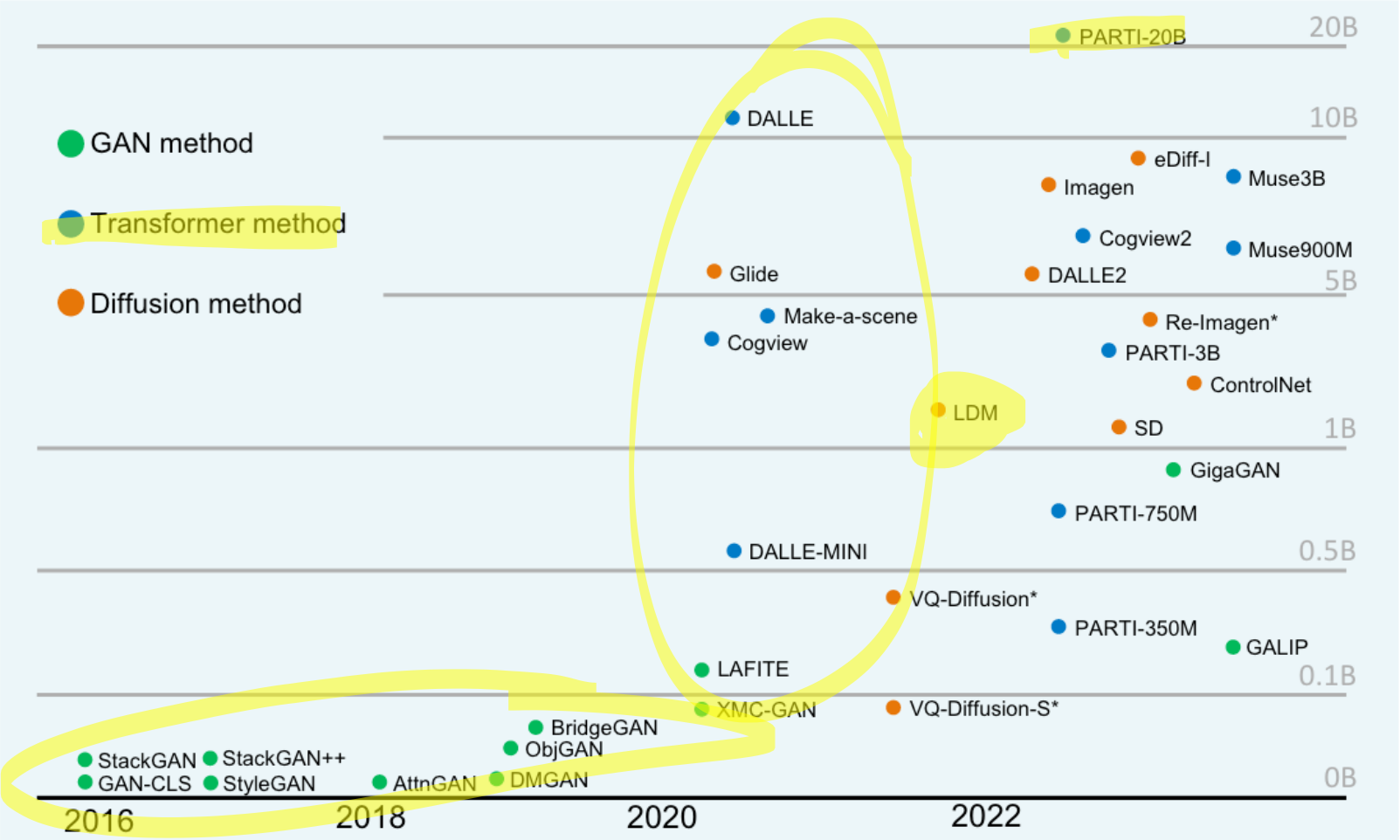


Fig. 5. Timeline of TTI model development, where green dots are GAN TTI models, blue dots are autoregressive Transformers and orange dots are Diffusion TTI models. Models are separated by their parameter, which are in general counted for all their components. Models with asterisk are calculated without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

Figure from Bie et al. (2023) http://arxiv.org/abs/2309.00810

Timeline: Text-to-Image Generation

A comparison of the text to image methods discussed highlighting their date published, model configuration and evaluation results. For the model type, green dot refers to the GAN model TTI, blue dot refers to the autoregressive TTI and orange dot indicates the Diffusion TTI. For evaluation metrics, IS and FID score are provided under the evaluation of MSCOCO dataset in a zero-shot fashion. The last column provides the specific model size in scale of Million(M) or Billion(B); ⋆ : no zero-shot results found, use standard results instead.

Method	Date	Model Type	Data Size	Open Source	IS evaluation	FID evaluation	Model size
AttnGAN [33]	11/2017	🟢	120K	✗	20.80	35.49 ⋆	13M
StyleGAN [34]	11/2017	🟢	120K	✗	20.80	35.49 ⋆	-
Obj-GAN [220]	09/2019	🟢	120K	✓	24.09	36.52 ⋆	34M
Control-GAN [221]	09/2019	🟢	120K	✓	23.61	33.10 ⋆	-
DM-GAN [35]	04/2019	🟢	120K	✓	32.32	27.34 ⋆	21M
XMC-GAN [165]	01/2021	🟢	120K	✗	30.45	9.33 ⋆	90M
LAFITE [44]	11/2021	🟢	-	✓	26.02	26.94	150M
Retreival-GAN [208]	08/2022	🟢	120K	✗	29.33	9.13 ⋆	25M
GigaGAN [46]	01/2023	🟢	-	✗	-	10.24	650M
GALIP [45]	03/2023	🟢	3M-12M	✓	-	12.54	240M
DALLE [39]	02/2021	🔵	250M	✗	-	27.5	12B
Cogview [189]	06/2021	🔵	300M	✓	-	27.1	4B
Make-A-Scene	03/2022	🔵	35M	✗	-	11.84	4B
Cogview2 [43]	05/2022	🔵	300M	✓	-	24.0	6B
PARTI-350M [5]	06/2022	🔵	∼1000M	✗	-	14.10	350M
PARTI-20B [5]	06/2022	🔵	∼1000M	✗	-	7.23	20B
DALLE-mini [187]	07/2021	🔵	250M	✗	-	-	∼500M
MUSE-3B [31]	03/2023	🔵	∼1000M	✗	-	7.88	7.6B
GLIDE [40]	12/2021	🟠	250M	✓	-	12.24	5B
VQ-diffusion-F [68]	11/2021	🟠	>7M	✓	-	13.86 ⋆	370M
DALLE-2 [4]	04/2022	🟠	250M	✗	-	10.39	5.2B
Imagen [30]	05/2022	🟠	∼860M	✗	-	7.27	7.6B
LDM [3]	08/2022	🟠	400M	✓	30.29	12.63	1.45B
eDiff-I [197]	11/2022	🟠	1000M	✗	-	6.95	9B
Shift Diffusion[158]	08/2022	🟠	900M	✓	-	10.88	-
Re-Imagen[203]	09/2022	🟠	50M	✗	-	6.88	∼8B
ControlNet [159]	03/2023	🟠	-	✓	-	-	∼2.2B

Figure from Bie et al. (2023) http://arxiv.org/abs/2309.00810

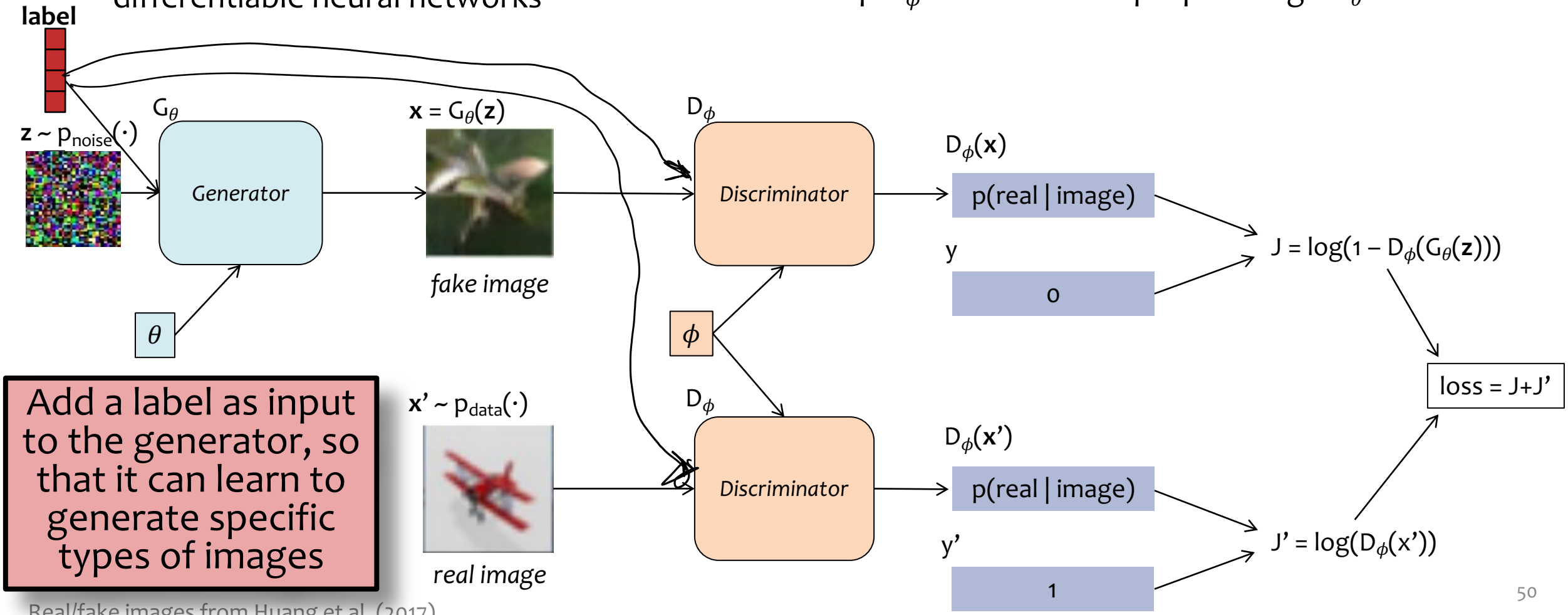
TEXT-TO-IMAGE: GANS

Class-conditional GANs

- Objective function is a simple differentiable function
- We chose G and D to be differentiable neural networks

Training alternates between:

- Keep G_θ fixed and backprop through D_ϕ
- Keep D_ϕ fixed and backprop through G_θ



Add a label as input to the generator, so that it can learn to generate specific types of images

Real/fake images from Huang et al. (2017)

Generative adversarial text to image synthesis

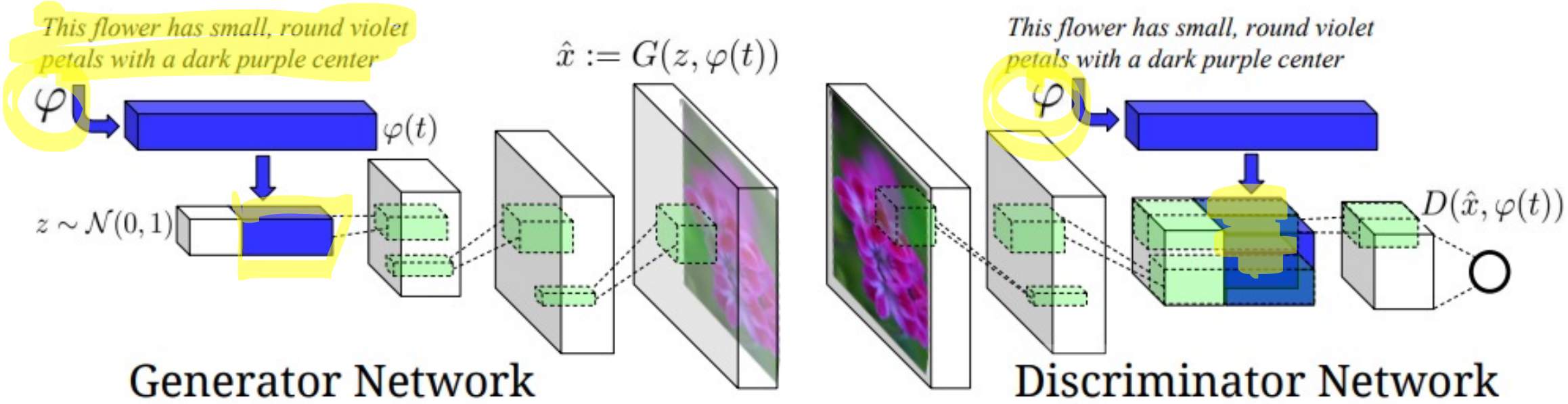


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Figure from Reed et al. (2016)

TEXT-TO-IMAGE: AUTOREGRESSIVE MODELS

Parti

The Pathways Autoregressive Text-to-Image (Parti) model:

- Step 1: Image tokenization (ViT-VQGAN)
  - pre-train a model to convert images into image tokens (discrete set of embeddings)
- Step 2: Training
  - treat image generation as a sequence-to-sequence problem
  - text prompt is input to encoder (pretrained BERT)
  - sequence of image tokens is output of decoder
- Step 3: Generation
  - ViT-VQGAN takes in the image tokens and generates a high-quality image

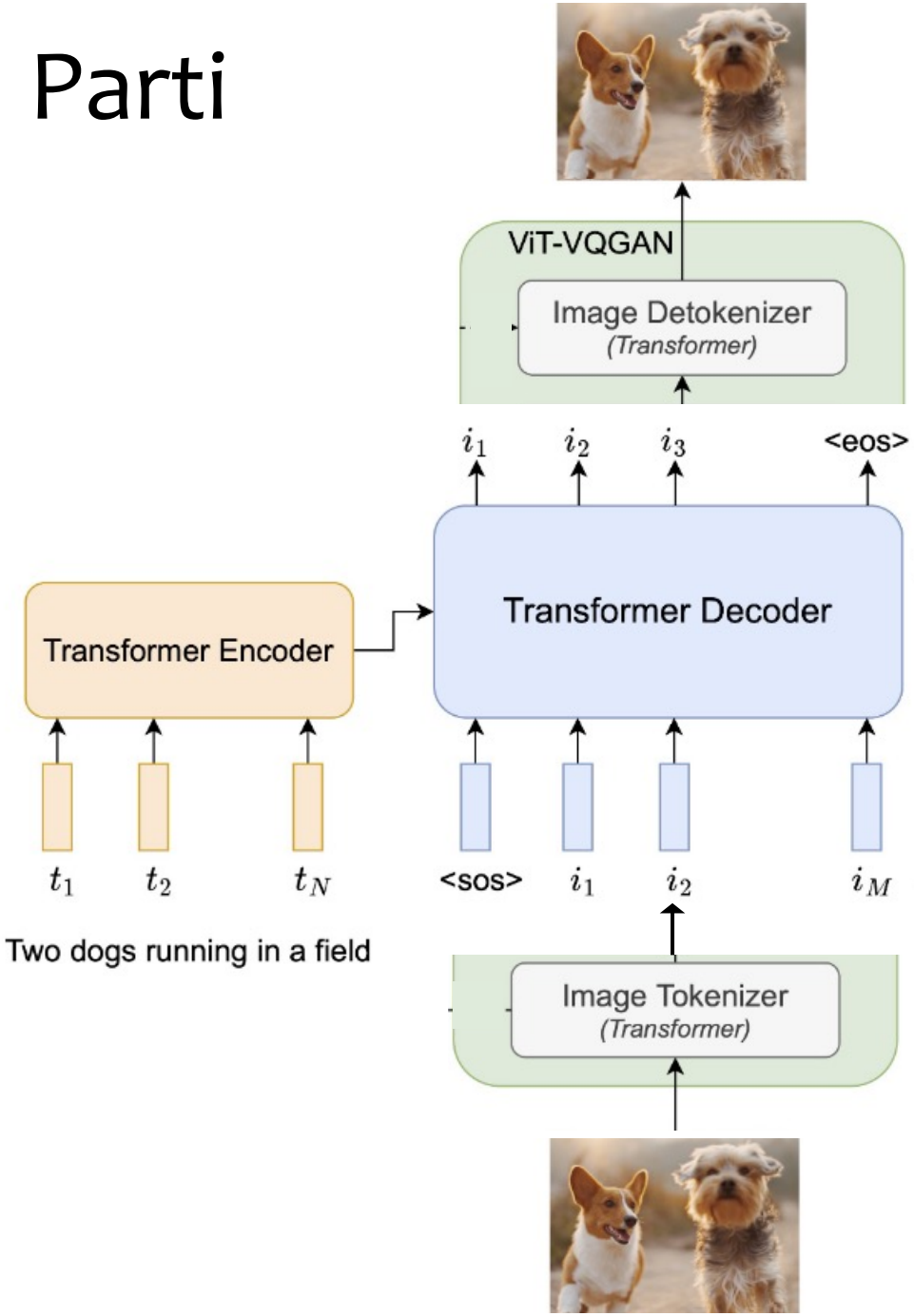
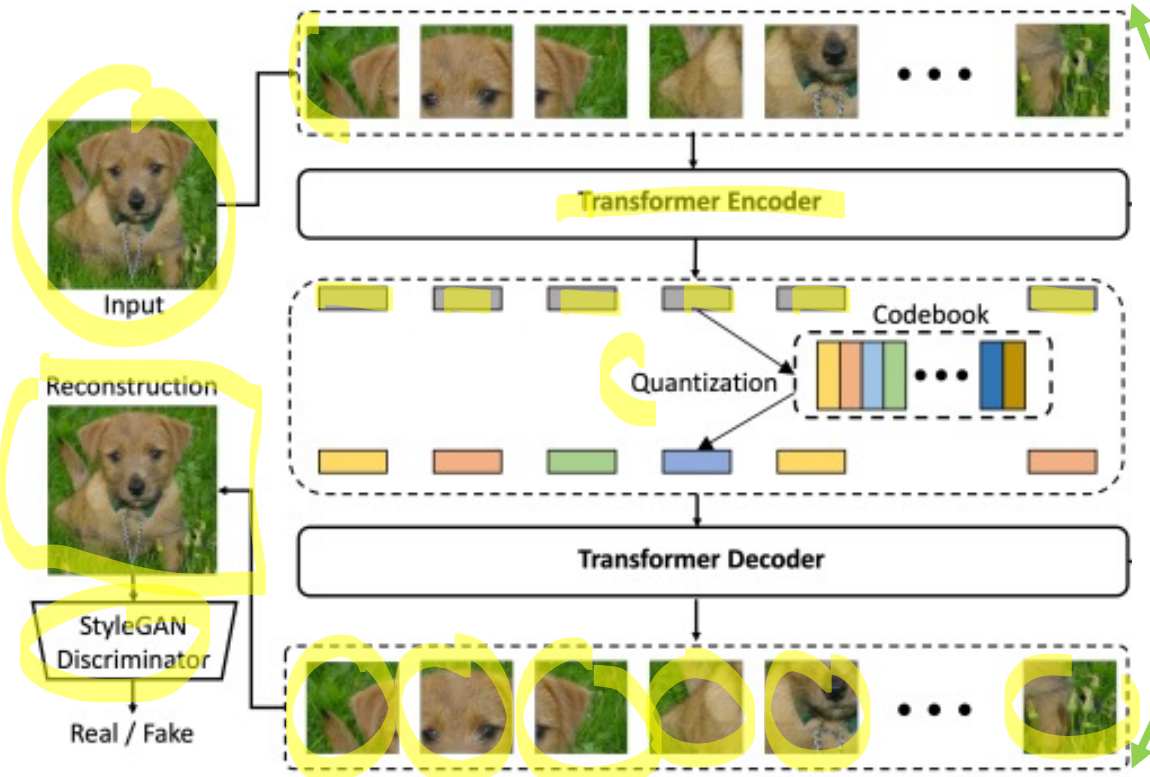


Figure adapted from https://arxiv.org/pdf/2206.10789

Parti

The Pathways Autoregressive Text-to-Image (Parti) model:

- **Step 1: Image tokenization (ViT-VQGAN)**
  - pre-train a model to convert images into image tokens (discrete set of embeddings)
- Step 2: Training
  - treat image generation as a sequence-to-sequence problem
  - text prompt is input to encoder (pretrained BERT)
  - sequence of image tokens is output of decoder
- Step 3: Generation
  - ViT-VQGAN takes in the image tokens and generates a high-quality image



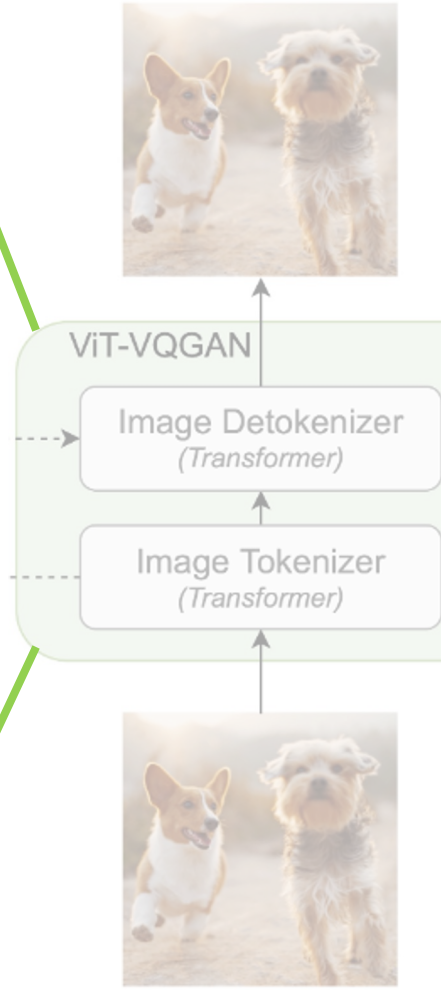


Figure adapted from https://arxiv.org/pdf/2206.10789 and https://arxiv.org/pdf/2110.04627

Parti

The Pathways Autoregressive Text-to-Image (Parti) model:

- Step 1: Image tokenization (ViT-VQGAN)
  - pre-train a model to convert images into image tokens (discrete set of embeddings)
- **Step 2: Training**
  - treat image generation as a sequence-to-sequence problem
  - text prompt is input to encoder (pretrained BERT)
  - sequence of image tokens is output of decoder
- Step 3: Generation
  - ViT-VQGAN takes in the image tokens and generates a high-quality image

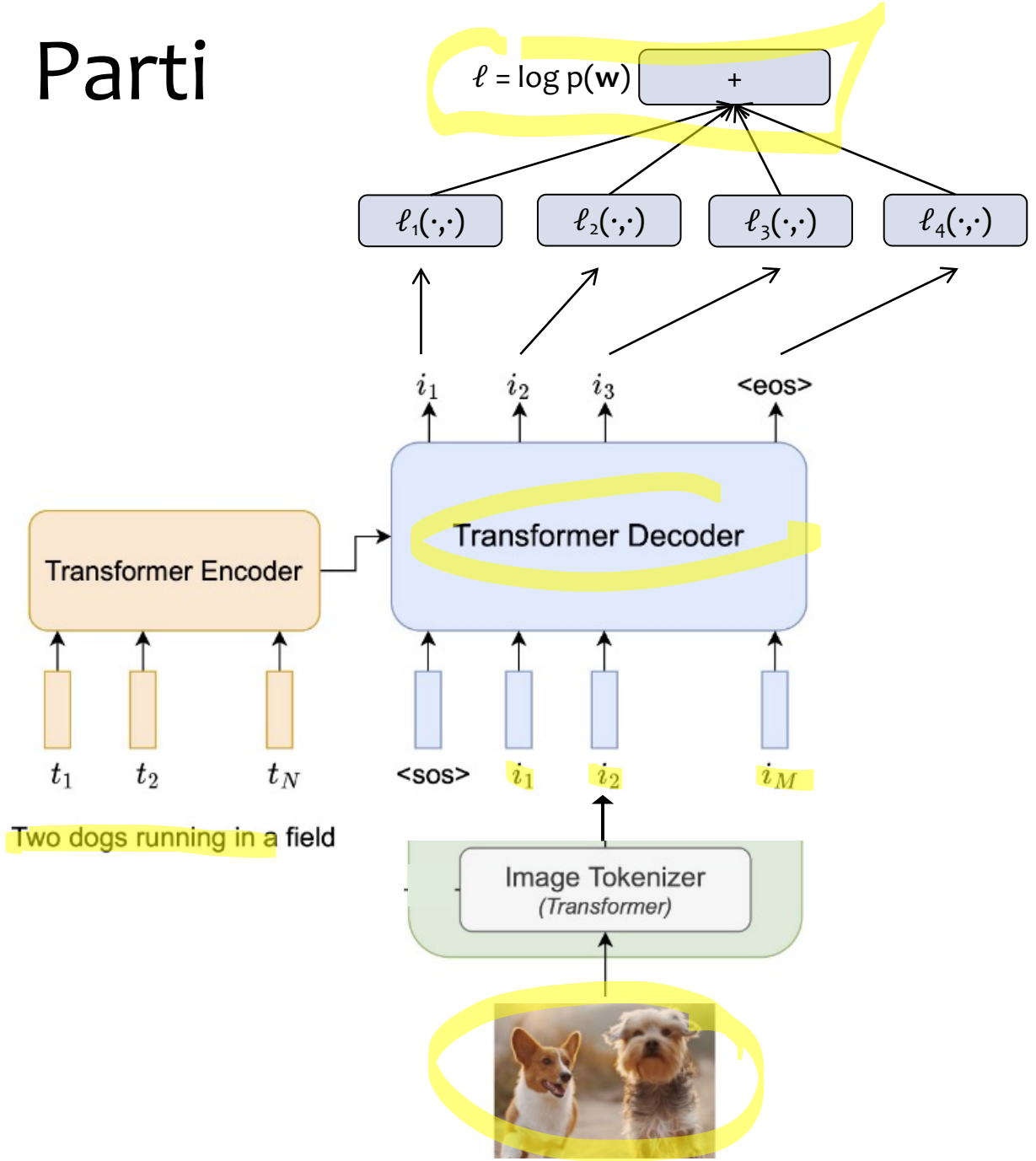


Figure adapted from https://arxiv.org/pdf/2206.10789

Parti

The Pathways Autoregressive Text-to-Image (Parti) model:

- Step 1: Image tokenization (ViT-VQGAN)
  - pre-train a model to convert images into image tokens (discrete set of embeddings)
- Step 2: Training
  - treat image generation as a sequence-to-sequence problem
  - text prompt is input to encoder (pretrained BERT)
  - sequence of image tokens is output of decoder
- **Step 3: Generation**
  - ViT-VQGAN takes in the image tokens and generates a high-quality image

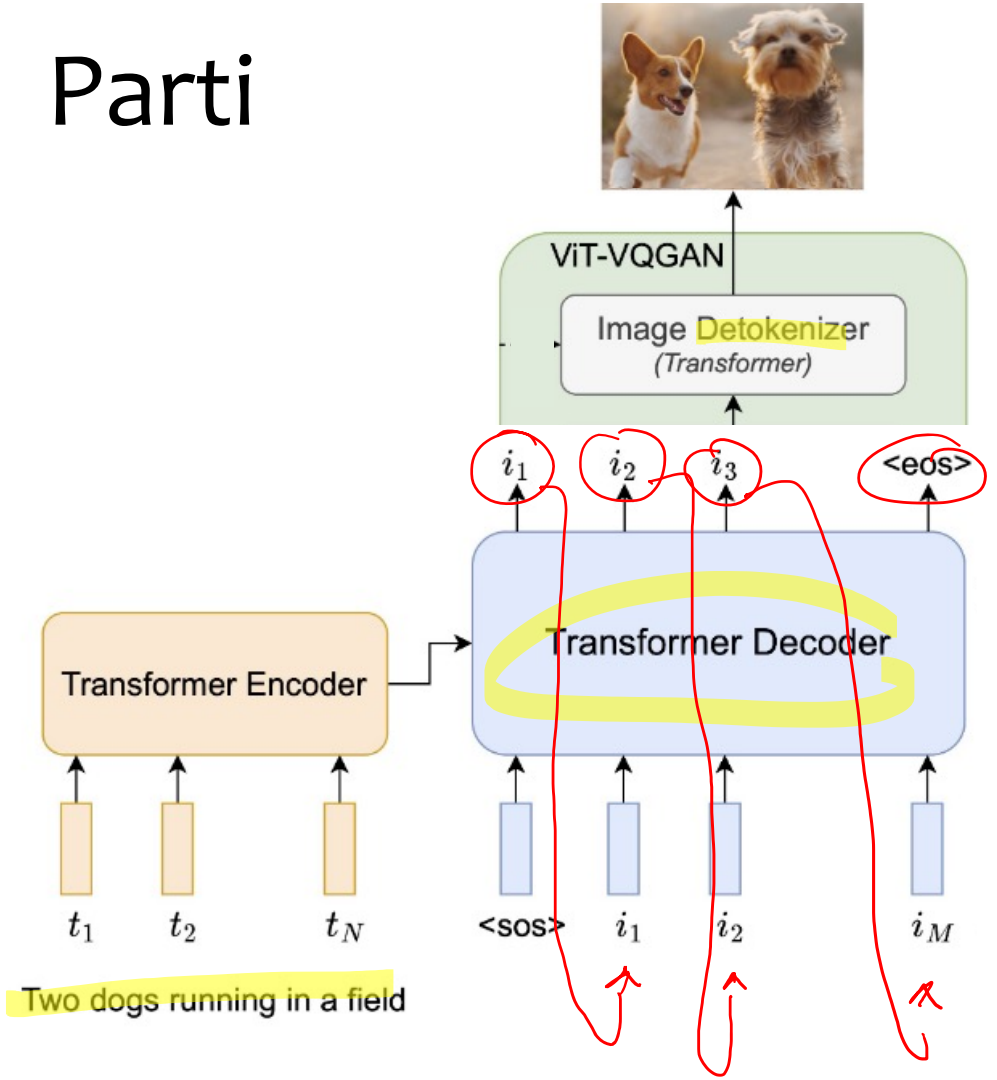


Figure adapted from https://arxiv.org/pdf/2206.10789

TEXT-TO-IMAGE: DIFFUSION MODELS

CLIP (background for Dall-E 2)

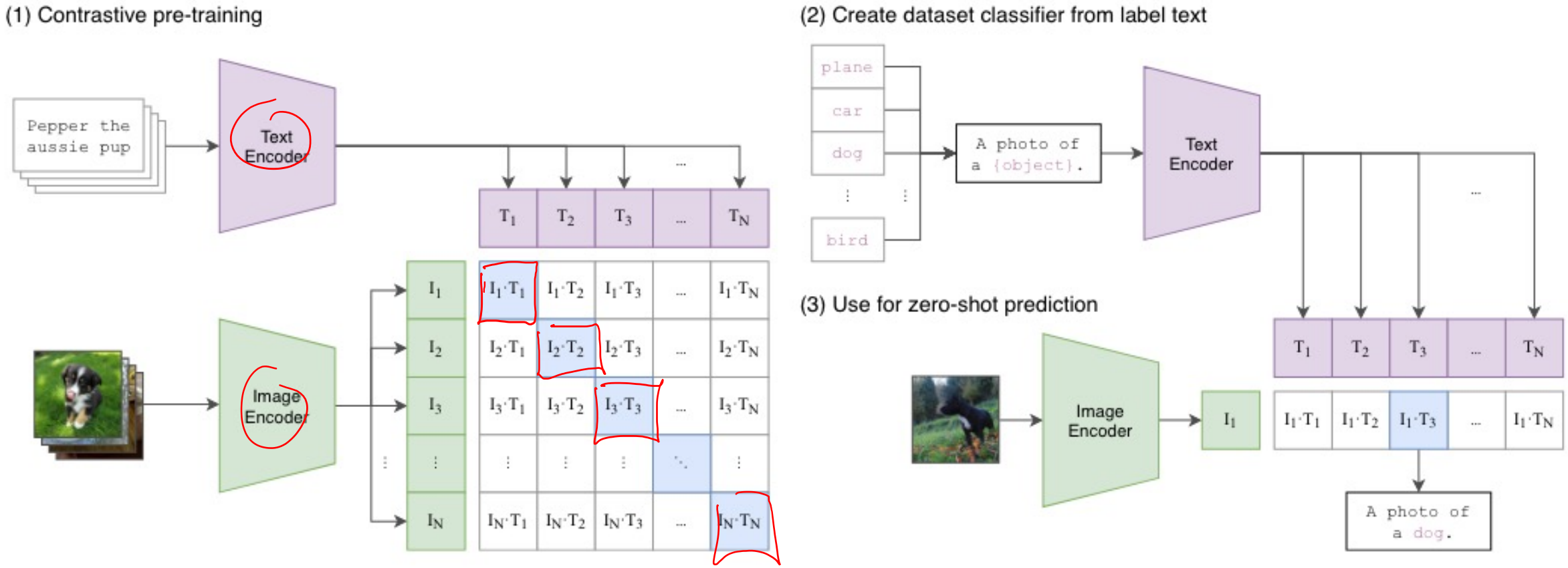


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

Figure from Radford et al. (2021)

Dall-E 2

- First pre-train a CLIP model
  - **text encoder (trained then frozen):** encode text as embedding using CLIP
  - **img encoder (trained then discarded):** though used to create CLIP image embeddings
- Second, train the prior/decoder of the Dall-E 2 model:
  - **prior (trained):** train a Gaussian diffusion model to generate CLIP image embeddings, conditioned on CLIP text embedding
  - **decoder (trained):** train an image diffusion model generate an image, conditioned on CLIP image embedding

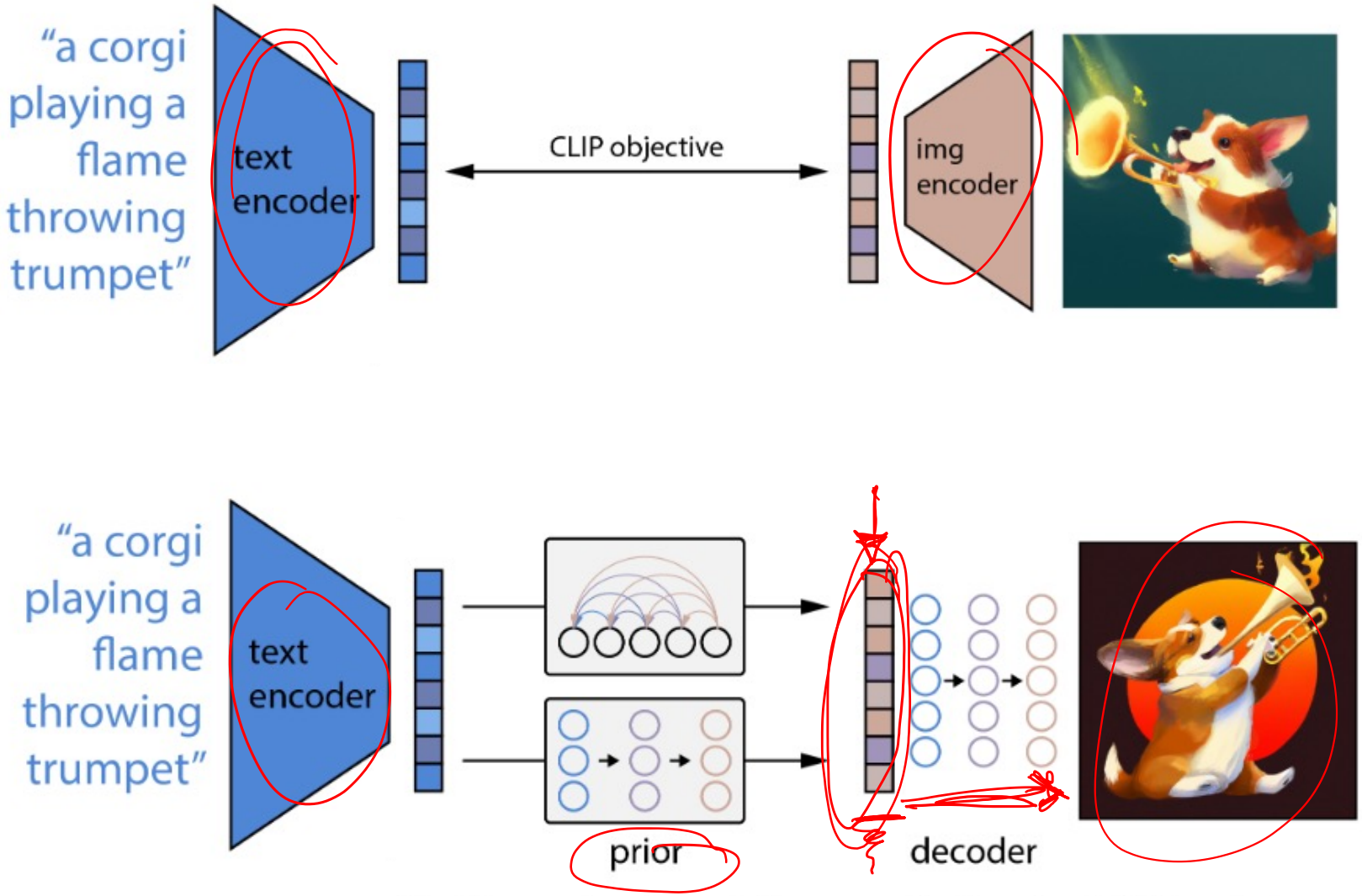


Figure adapted from Ramesh et al. (2022)

Imagen

- Imagen uses a text-to-image diffusion model coupled with a super-resolution diffusion model
- All the models operate in pixel space
- While effective, the compute requirements are very high

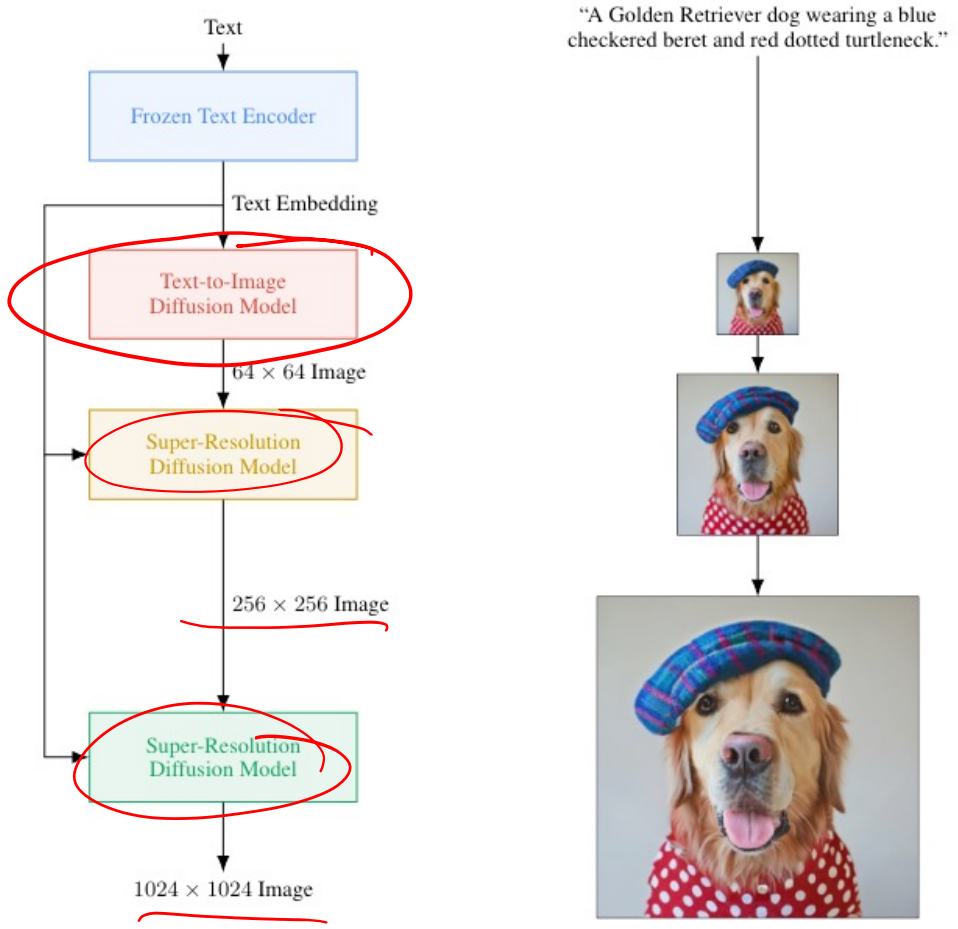


Figure A.4: Visualization of Imagen. Imagen uses a frozen text encoder to encode the input text into text embeddings. A conditional diffusion model maps the text embedding into a 64×64 image. Imagen further utilizes text-conditional super-resolution diffusion models to upsample the image, first $64 \times 64 \rightarrow 256 \times 256$, and then $256 \times 256 \rightarrow 1024 \times 1024$.

LATENT DIFFUSION MODEL (LDM)

Latent Diffusion Model

Motivation:

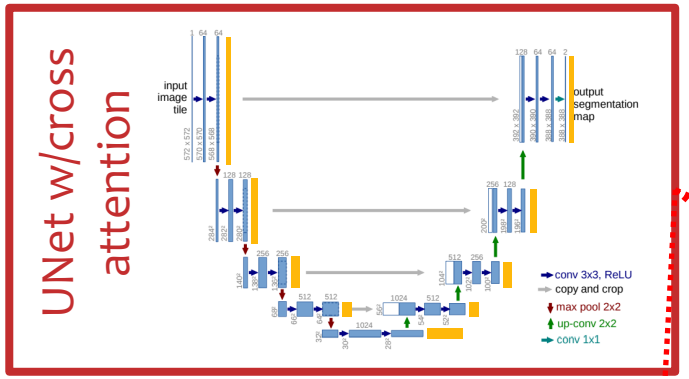
- diffusion models typically operate in pixel space
- yet, training typically takes hundreds of GPU days
  - 150 – 1000 V100 days [Guided Diffusion] (Dhariwal & Nichol, 2021)
  - 256 TPU-v4s for 4 days = 1000 TPU days [Imagen] (Sharia et al., 2022)
- inference is also slow
  - 50k samples in 5 days on A100 GPU [Guided Diffusion] (Dhariwal & Nichol, 2021)
  - 15 seconds per image

Key Idea:

- train an autoencoder (i.e. encoder-decoder model) that learns an efficient latent space that is perceptually equivalent to the data space
- keeping the autoencoder fixed, train a diffusion model on the latent representations of real images z_0 = encoder(x)
  - forward model: latent representation z_0 → noise z_T
  - reverse model: noise z_T → latent representation z_0
- to generate an image:
  - sample noise z_T
  - apply reverse diffusion model to obtain a latent representation z_0
  - decode the latent representation to an image x
- condition on prompt via cross attention in latent space

Latent Diffusion Model

Latent Diffusion Model (LDM)



prompt space

latent space

pixel space

$\widehat{\mathbf{y}}$

τ_θ

LLM

$\mathbf{y}$

orange cat

$p_\theta(\mathbf{z}_T)$

$p_\theta(\mathbf{z}_{T-1} \mid \mathbf{z}_T, \widehat{\mathbf{y}})$

$p_\theta(\mathbf{z}_t \mid \mathbf{z}_{t+1}, \widehat{\mathbf{y}})$

$p_\theta(\mathbf{z}_0 \mid \mathbf{z}_1, \widehat{\mathbf{y}})$

z_T z_{T-1} ... z_{t+1} z_t ... z_1 z

$\mathcal{D}$

Decoder

$\widetilde{\mathbf{x}}$

$\mathcal{E}$

Encoder

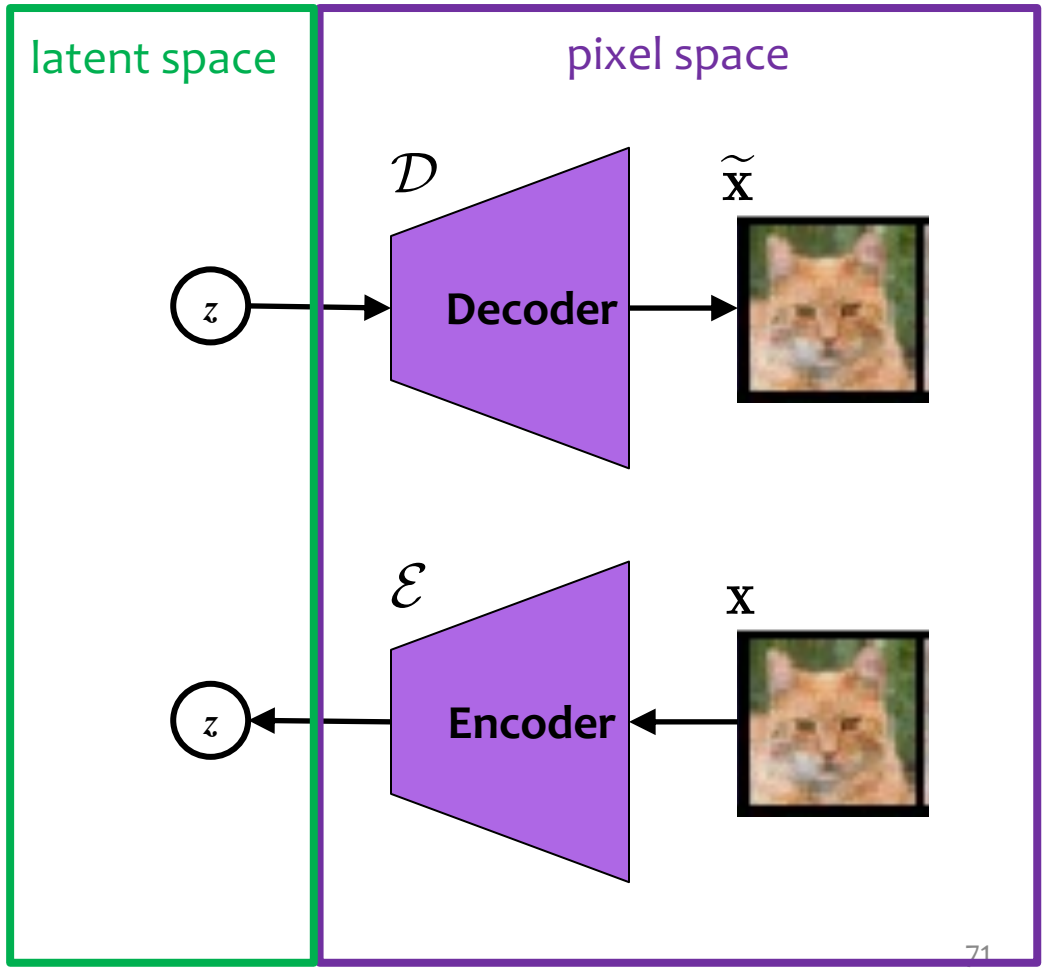
$\mathbf{x}$

$q_\phi(\mathbf{z}_T \mid \mathbf{z}_{T-1})$

$q_\phi(\mathbf{z}_{t+1} \mid \mathbf{z}_t)$

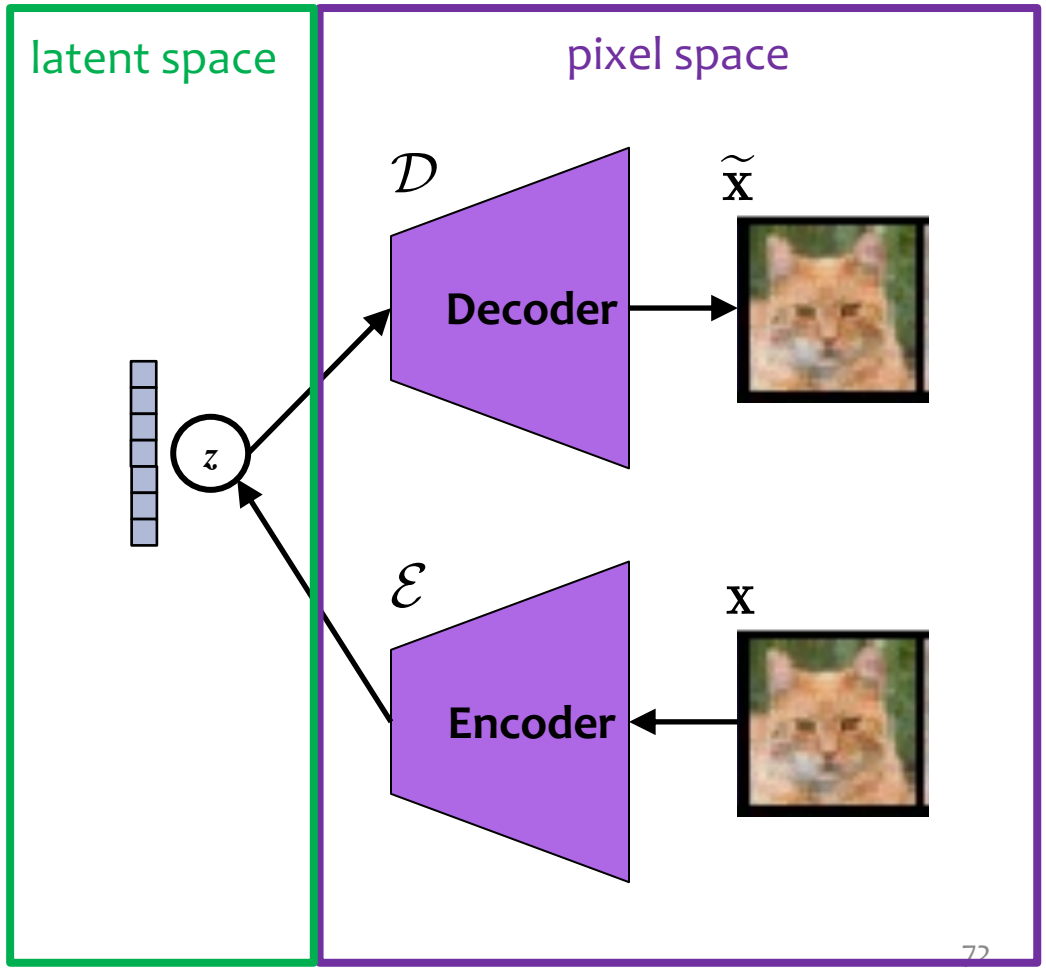
$q_\phi(\mathbf{z}_1 \mid \mathbf{z}_0)$

LDM: Autoencoder



LDM: Autoencoder

- The autoencoder is chosen so that it can project high dimensional images (e.g. 1024x1024) down to low dimensional **latent space** and **faithfully** project back up to **pixel space**
- The original LDM paper considers two options:
  1. a VAE-like model (regularizes the noise towards a Gaussian)
  2. a VQGAN (performs vector quantization in the decoder; i.e., it uses a discrete codebook)
- This model is trained ahead of time just on raw images (no text prompts) and then frozen
- The frozen encoder-decoder can be reused for all subsequent LDM training

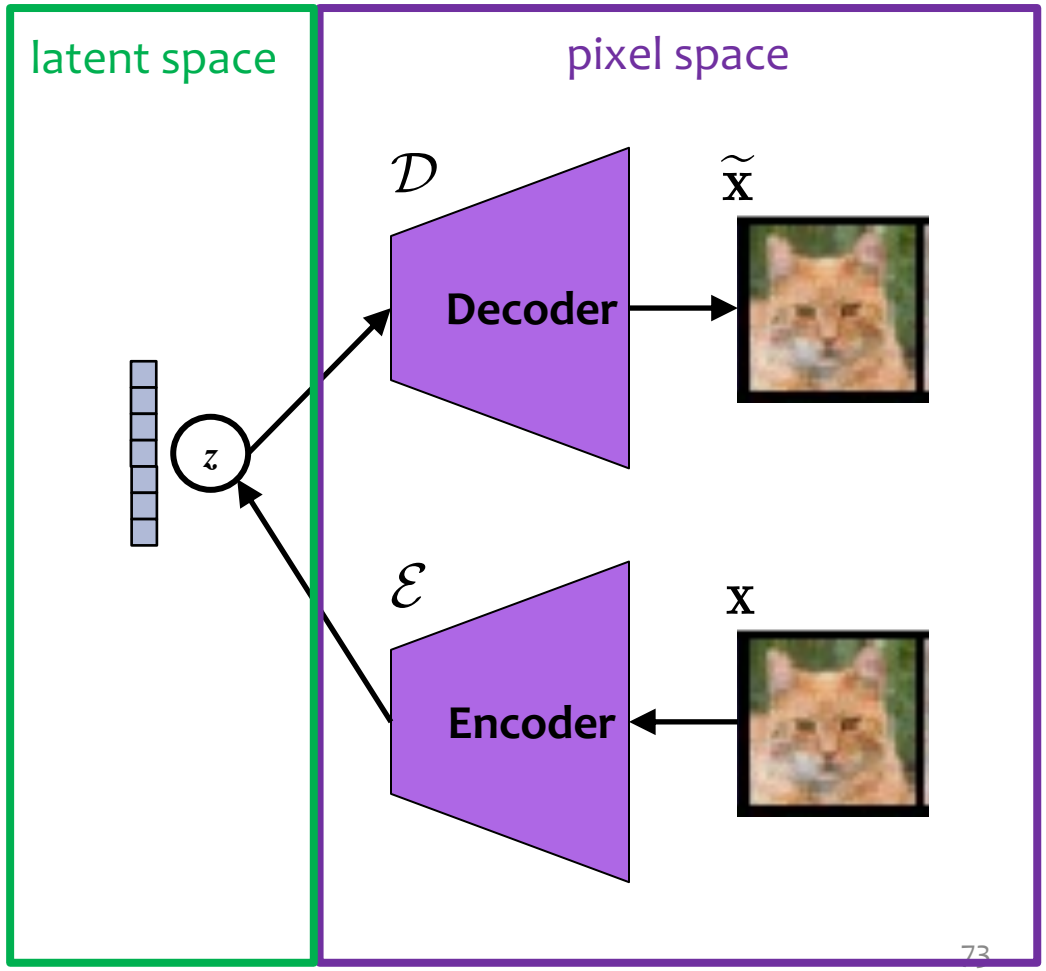


LDM: Autoencoder

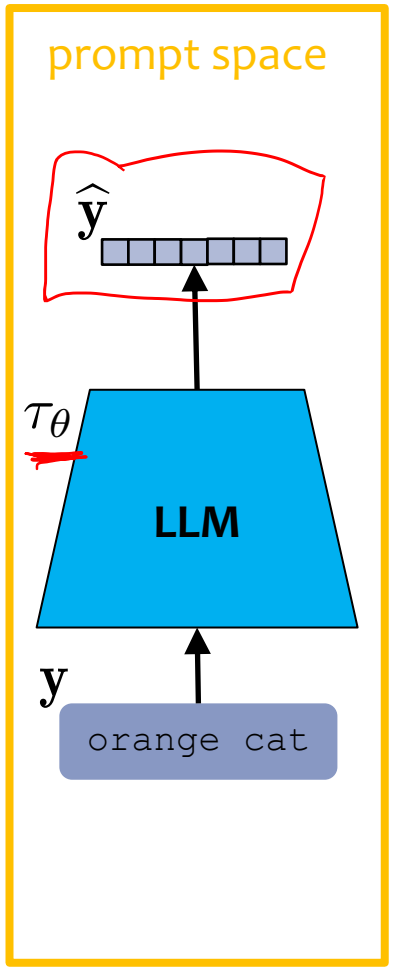
- After trying a zoo of autoencoder options, the original paper picked one that offered a good level of compression without much loss of information

f	$\lvert\mathcal{Z}\rvert$	c	R-FID ↓	R-IS ↑	PSNR ↑	PSIM ↓	SSIM ↑
16 *VQGAN* [23]	16384	256	4.98	–	19.9 ±3.4	1.83 ±0.42	0.51 ±0.18
16 *VQGAN* [23]	1024	256	7.94	–	19.4 ±3.3	1.98 ±0.43	0.50 ±0.18
8 *DALL-E* [66]	8192	-	32.01	–	22.8 ±2.1	1.95 ±0.51	0.73 ±0.13
32	16384	16	31.83	40.40 ±1.07	17.45 ±2.90	2.58 ±0.48	0.41 ±0.18
16	16384	8	5.15	144.55 ±3.74	20.83 ±3.61	1.73 ±0.43	0.54 ±0.18
8	16384	4	1.14	201.92 ±3.97	23.07 ±3.99	1.17 ±0.36	0.65 ±0.16
8	256	4	1.49	194.20 ±3.87	22.35 ±3.81	1.26 ±0.37	0.62 ±0.16
4	8192	3	0.58	224.78 ±5.35	27.43 ±4.26	0.53 ±0.21	0.82 ±0.10
4†	8192	3	1.06	221.94 ±4.58	25.21 ±4.17	0.72 ±0.26	0.76 ±0.12
4	256	3	0.47	223.81 ±4.58	26.43 ±4.22	0.62 ±0.24	0.80 ±0.11
2	2048	2	0.16	232.75 ±5.09	30.85 ±4.12	0.27 ±0.12	0.91 ±0.05
2	64	2	0.40	226.62 ±4.83	29.13 ±3.46	0.38 ±0.13	0.90 ±0.05
32	KL	64	2.04	189.53 ±3.68	22.27 ±3.93	1.41 ±0.40	0.61 ±0.17
32	KL	16	7.3	132.75 ±2.71	20.38 ±3.56	1.88 ±0.45	0.53 ±0.18
16	KL	16	0.87	210.31 ±3.97	24.08 ±4.22	1.07 ±0.36	0.68 ±0.15
16	KL	8	2.63	178.68 ±4.08	21.94 ±3.92	1.49 ±0.42	0.59 ±0.17
8	KL	4	0.90	209.90 ±4.92	24.19 ±4.19	1.02 ±0.35	0.69 ±0.15
4	KL	3	0.27	227.57 ±4.89	27.53 ±4.54	0.55 ±0.24	0.82 ±0.11
2	KL	2	0.086	232.66 ±5.16	32.47 ±4.19	0.20 ±0.09	0.93 ±0.04

Table 8. Complete autoencoder zoo trained on OpenImages, evaluated on ImageNet-Val. † denotes an attention-free autoencoder.

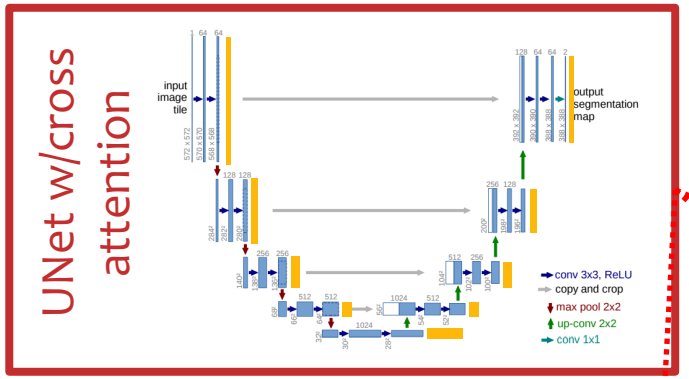


LDM: the Prompt Model



- The prompt model is just a Transformer LM
- We learn its parameters alongside the diffusion model
- The goal is to build up good representations of the text prompts such that they inform the latent diffusion process

LDM: with DDPM



prompt space

latent space

$\widehat{\mathbf{y}}$

τ_θ

LLM

$\mathbf{y}$

orange cat

$p_\theta(\mathbf{z}_T)$

$p_\theta(\mathbf{z}_{T-1} \mid \mathbf{z}_T, \widehat{\mathbf{y}})$

$p_\theta(\mathbf{z}_t \mid \mathbf{z}_{t+1}, \widehat{\mathbf{y}})$

$p_\theta(\mathbf{z}_0 \mid \mathbf{z}_1, \widehat{\mathbf{y}})$

z_T z_{T-1} ... z_{t+1} z_t ... z_1 z

$q_\phi(\mathbf{z}_T \mid \mathbf{z}_{T-1})$

$q_\phi(\mathbf{z}_{t+1} \mid \mathbf{z}_t)$

$q_\phi(\mathbf{z}_1 \mid \mathbf{z}_0)$

LDM: with DDPM

Noise schedule:

We choose α_t to follow a fixed schedule s.t. $q_\phi(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, just like $p_\theta(\mathbf{x}_T)$.

Here we let $\mathbf{z}_0 = \mathbf{z}$, the output of the encoder from our autoencoder

Forward Process:

$$q_\phi(\mathbf{z}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^{T} q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1})$$

$$q(\mathbf{z}_0) = \text{data distribution}$$

$$q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \sim \mathcal{N}(\sqrt{\alpha_t}\mathbf{z}_{t-1}, (1-\alpha_t)\mathbf{I})$$

(Learned) Reverse Process:

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

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

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

LDM: with DDPM

Noise schedule:

We choose α_t to follow a fixed schedule s.t. $q_\phi(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, just like $p_\theta(\mathbf{x}_T)$.

Here we let $\mathbf{z}_0 = \mathbf{z}$, the output of the encoder from our autoencoder

Forward Process:

$$q_\phi(\mathbf{z}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^{T} q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1})$$

$$q(\mathbf{z}_0) = \text{data}$$

$$q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \sim \mathcal{N}(\sqrt{$$

Question: How do we define the mean to condition on the prompt representation?

(Learned) Reverse Process:

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

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

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

Properties of forward and *exact* reverse process

Recall...

Property #1:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

$$\text{where } \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

$\Rightarrow$ we can sample $\mathbf{x}_t$ from $\mathbf{x}_0$ at any timestep t efficiently in closed form

$\Rightarrow$ $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Property #2: Estimating $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is intractable because of its dependence on $q(\mathbf{x}_0)$. However, conditioning on $\mathbf{x}_0$ we can efficiently work with:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2 \mathbf{I})$$

$$\begin{aligned}\text{where } \tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\bar{\alpha}_t}(1-\alpha_t)}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}{1-\bar{\alpha}_t}\mathbf{x}_t \\ &= \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t \\ \sigma_t^2 &= \frac{(1-\bar{\alpha}_{t-1})(1-\alpha_t)}{1-\bar{\alpha}_t}\end{aligned}$$

Property #3: Combining the two previous properties, we can obtain a different parameterization of $\tilde{\mu}_q$ which has been shown empirically to help in learning p_θ.

Rearranging $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}$ we have that:

$$\mathbf{x}_0 = \left(\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}\right) / \sqrt{\bar{\alpha}_t}$$

Substituting this definition of $\mathbf{x}_0$ into property #2's definition of $\tilde{\mu}_q$ gives:

$$\begin{aligned}\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) &= \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t \\ &= \alpha_t^{(0)}\left(\left(\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}\right)/\sqrt{\bar{\alpha}_t}\right) + \alpha_t^{(t)}\mathbf{x}_t \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{(1-\alpha_t)}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right)\end{aligned}$$

Parameterizing the *learned* reverse process

Recall...

Recall: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Idea #1: Rather than learn $\Sigma_\theta(\mathbf{x}_t, t)$ just use what we know about $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \sim \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2 \mathbf{I})$:

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

Idea #2: Choose μ_θ based on $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, i.e. we want $\mu_\theta(\mathbf{x}_t, t)$ to be close to $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$. Here are three ways we could parameterize this:

Option C: Learn a network that approximates the $\boldsymbol{\epsilon}$ that gave rise to $\mathbf{x}_t$ from $\mathbf{x}_0$ in the forward process from $\mathbf{x}_t$ and t:

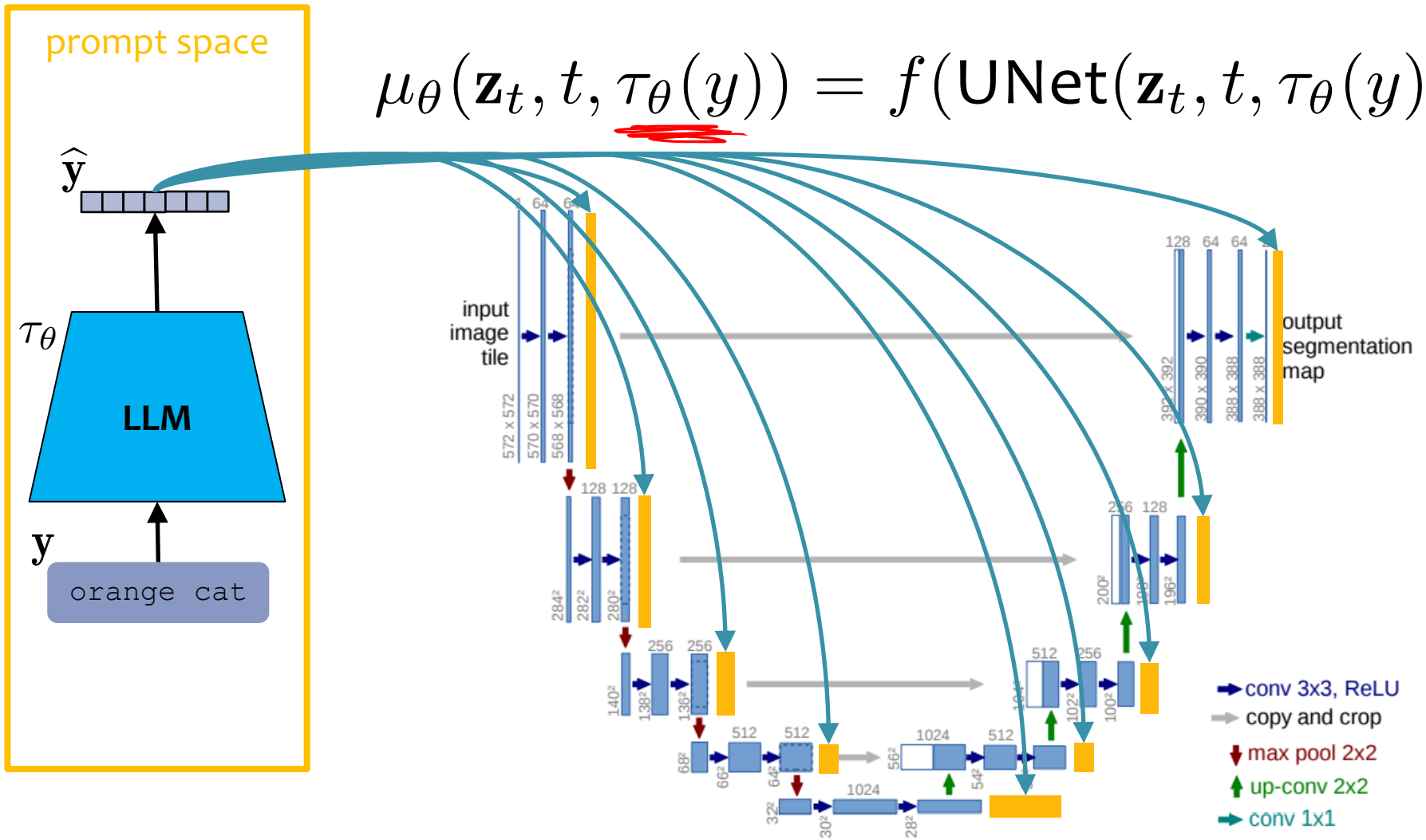
$$\mu_\theta(\mathbf{x}_t, t) = \alpha_t^{(0)} \mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) + \alpha_t^{(t)} \mathbf{x}_t$$

where $\mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) = \left(\mathbf{x}_0 + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) / \sqrt{\bar{\alpha}_t}$

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

LDM: Noise Model

$$\mu_\theta(\mathbf{z}_t, t, \tau_\theta(y)) = f(\text{UNet}(\mathbf{z}_t, t, \tau_\theta(y)))$$



- The noise model includes **cross attention** (yellow boxes) to 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

LDM: Cross-Attention in Noise Model

- The cross-attention is placed within a larger Transformer layer

Transformer Layer inside UNet

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	$\mathbb{R}^{h \times w \times c}$

- The cross-attention modifies the keys and values to be the prompt representation
- The queries are the current layer of UNet

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

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

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]. See Fig. 3 for a visual depiction.

Figure from http://arxiv.org/abs/2112.10752

LDM: Learning the Diffusion Model + LLM

Given a training sample $\mathbf{z}_0$, we want

$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y))$$

to be as close as possible to

$$q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{z}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{z}_0$ it should subtract it away exactly as *exact* reverse process would have.

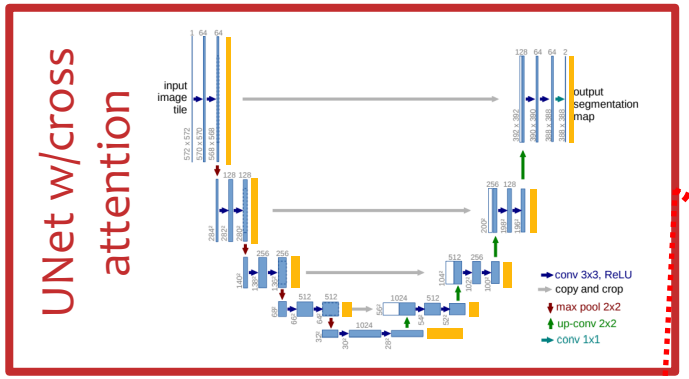
Objective Function:

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\right]$$

Algorithm 1 Training

1: initialize θ
2: **for** $e \in \{1, \ldots, E\}$ **do**
3: **for** $x_0, y \in \mathcal{D}$ **do**
4: $t \sim \text{Uniform}(1, \ldots, T)$
5: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$
7: $\ell_t(\theta) \leftarrow \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \tau_\theta(\mathbf{y}))\|^2$
8: $\theta \leftarrow \theta - \nabla_\theta \ell_t(\theta)$

Latent Diffusion Model (LDM)



prompt space

latent space

pixel space

$\widehat{\mathbf{y}}$

τ_θ

LLM

$\mathbf{y}$

orange cat

$p_\theta(\mathbf{z}_T)$

$p_\theta(\mathbf{z}_{T-1} \mid \mathbf{z}_T, \widehat{\mathbf{y}})$

$p_\theta(\mathbf{z}_t \mid \mathbf{z}_{t+1}, \widehat{\mathbf{y}})$

$p_\theta(\mathbf{z}_0 \mid \mathbf{z}_1, \widehat{\mathbf{y}})$

z_T z_{T-1} ... z_{t+1} z_t ... z_1 z

$\mathcal{D}$

Decoder

$\widetilde{\mathbf{x}}$

$\mathcal{E}$

Encoder

$\mathbf{x}$

$q_\phi(\mathbf{z}_T \mid \mathbf{z}_{T-1})$

$q_\phi(\mathbf{z}_{t+1} \mid \mathbf{z}_t)$

$q_\phi(\mathbf{z}_1 \mid \mathbf{z}_0)$

LDM Results

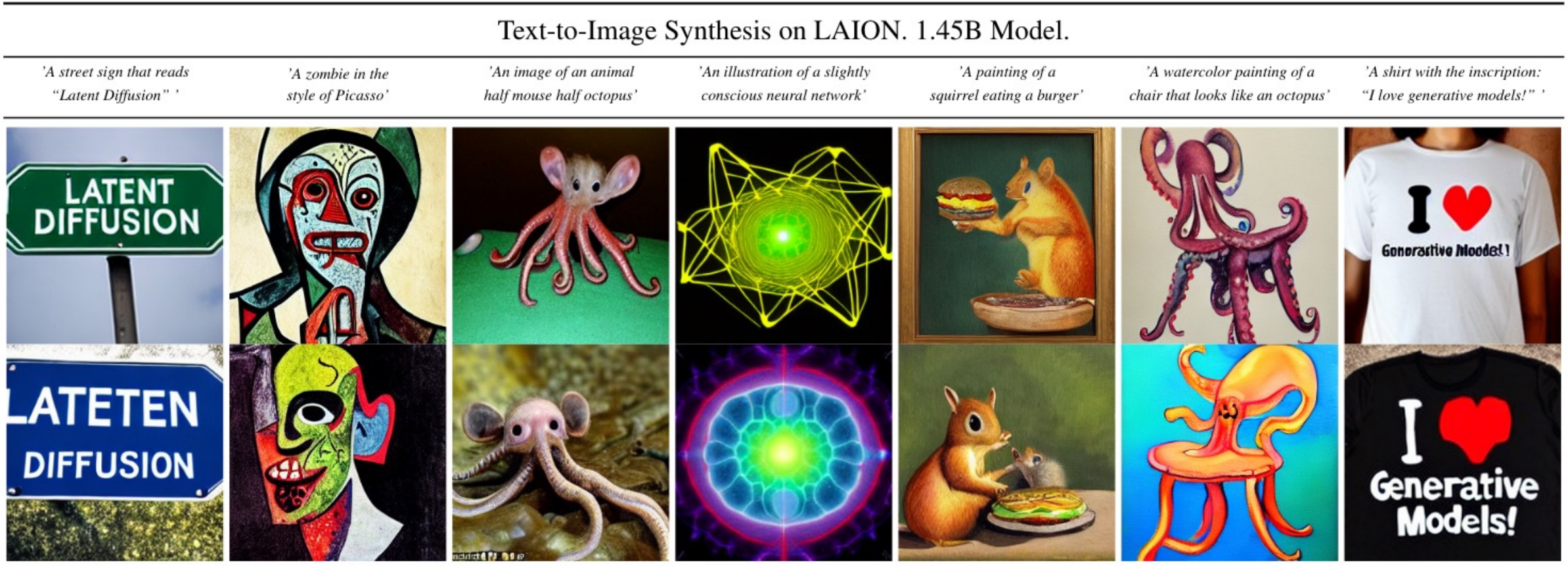


Figure 5. Samples for user-defined text prompts from our model for text-to-image synthesis, *LDM-8 (KL)*, which was trained on the LAION [78] database. Samples generated with 200 DDIM steps and $\eta = 1.0$. We use unconditional guidance [32] with $s = 10.0$.

Figure from http://arxiv.org/abs/2112.10752

LDM Results

- The result models obtain very high quality FID / IS scores with many fewer parameters than competing models
- The models are much more efficient than vanilla diffusion 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	$\mathbf{30.29}_{\pm \mathbf{0.42}}$	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]

Figure from http://arxiv.org/abs/2112.10752