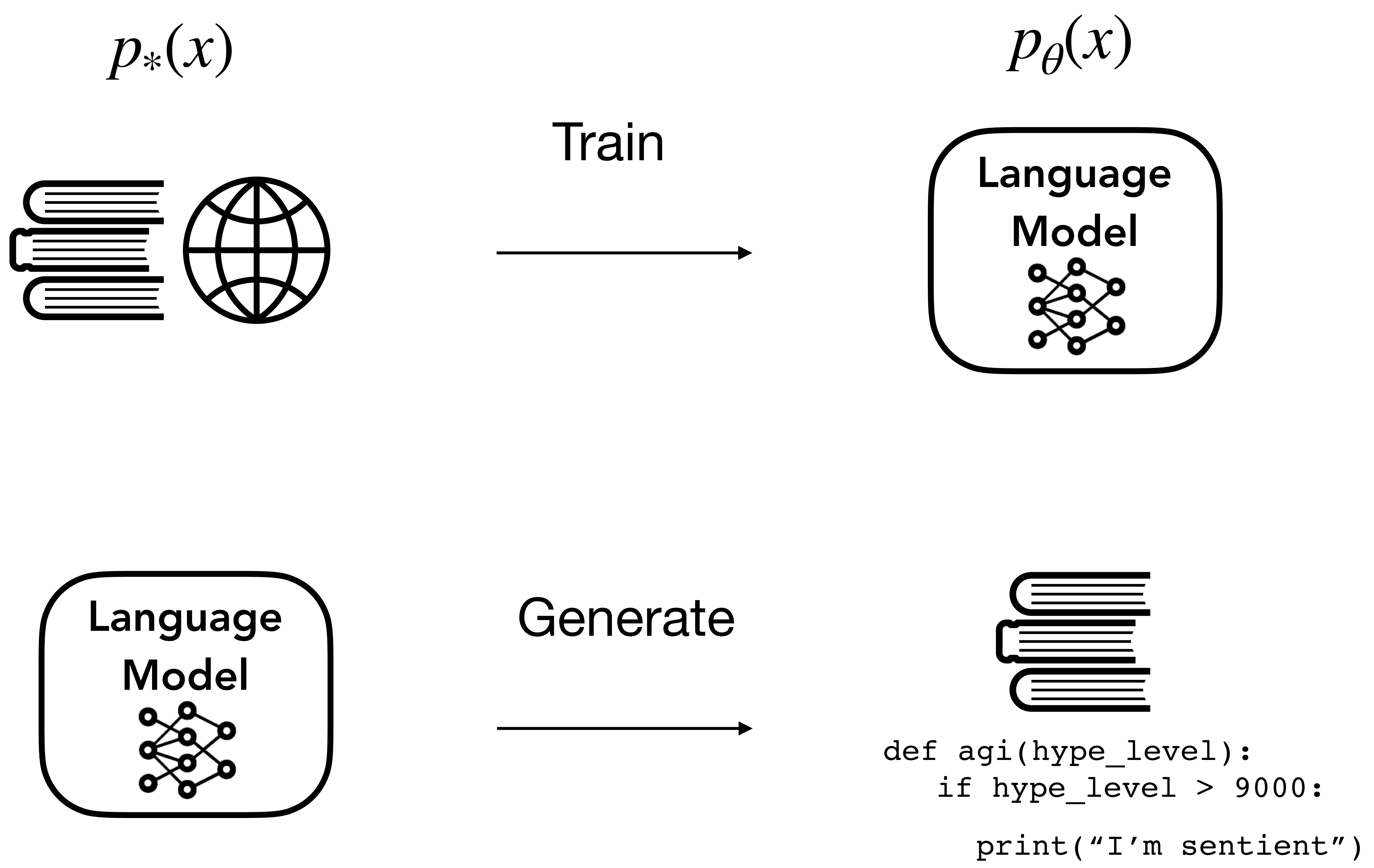
Denoising diffusion models

Sean Welleck

July 27 2022

Some content inspired or adapted from:
[MIT Lecture: Diffusion Probabilistic Models, Jascha Sohl-Dickstein]
[CVPR 2022 Tutorial]
[IRCAM diffusion notebooks]
[Lillian Weng's Blog]
[Yang Song's Blog]
Papers referenced in slides

Generative modeling

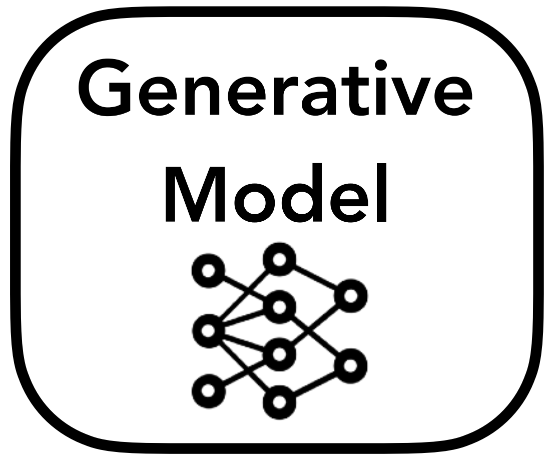


Generative modeling

$p_*(x)$

Train

$p_\theta(x)$



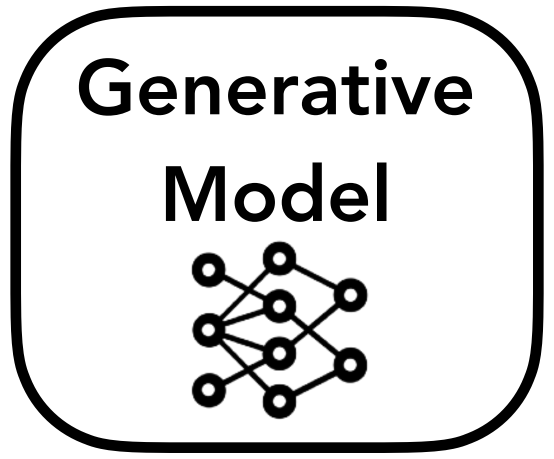
Autoregressive

Variational auto encoder (VAE)

Generative adversarial networks (GAN)

Denoising Diffusion models

$p_\theta(x)$



Sample

Diffusion models

Emerging as powerful generative models

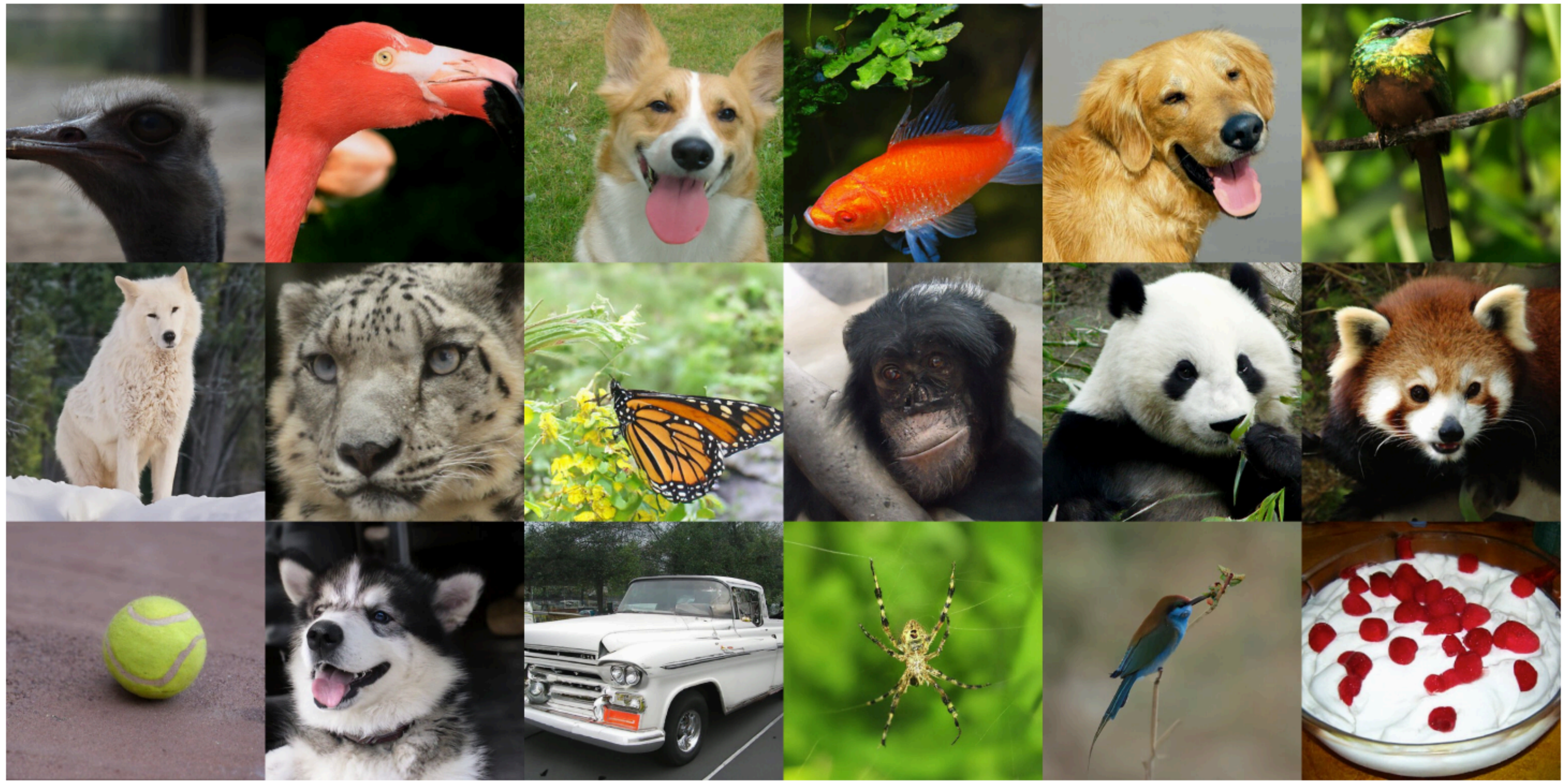
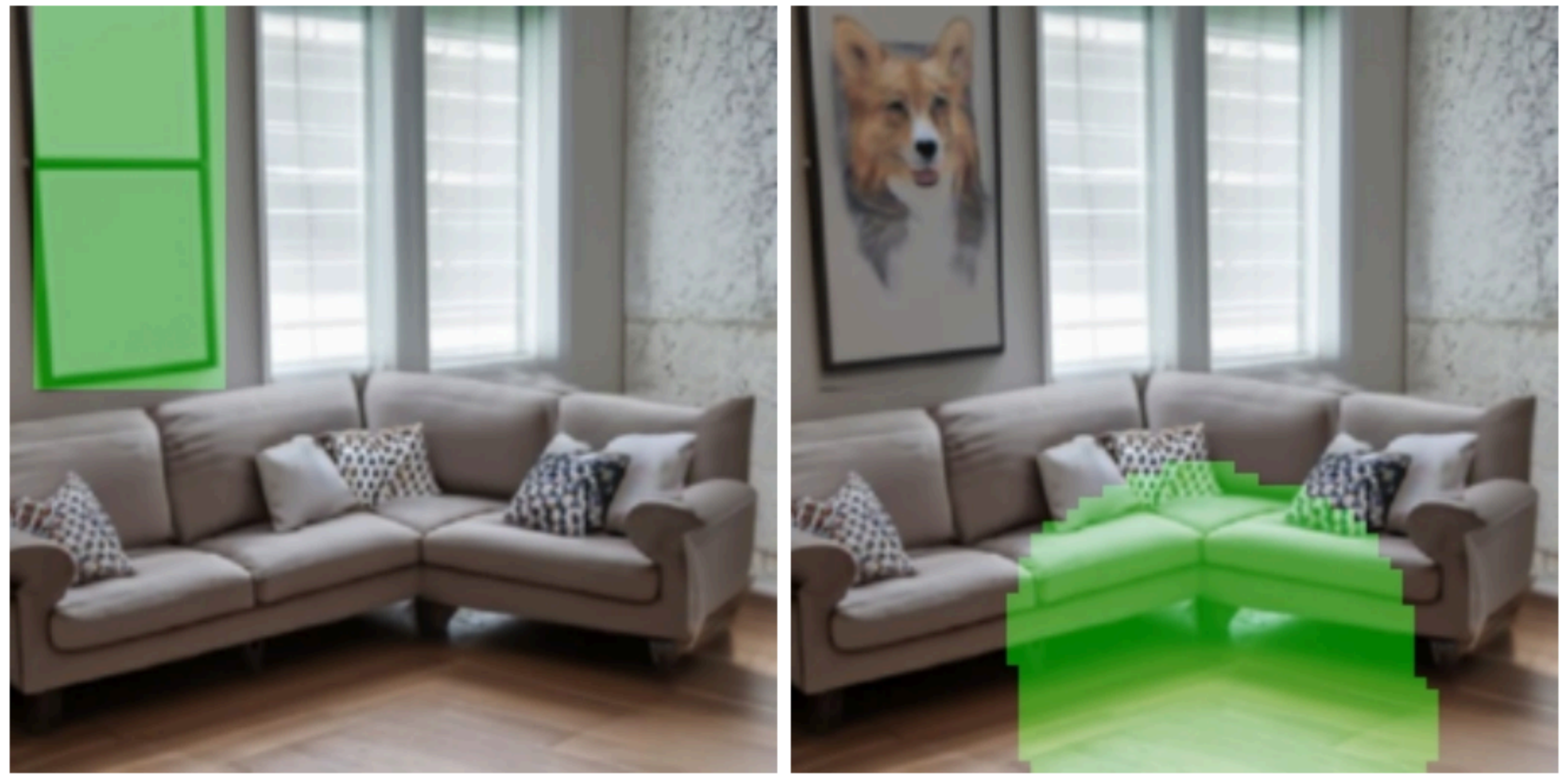


Figure 1: Selected samples from our best ImageNet 512×512 model (FID 3.85)

[Dhariwal & Nichol 2021, Diffusion Models Beat GANs on Image Synthesis, NeurIPS 2021]

Diffusion models



“a cozy living room” “a painting of a corgi on the wall above a couch”

Figure 3. Iteratively creating a complex scene using GLIDE.

A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.

[Nichol et al. 2022, GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models]
[Saharia et al. 2022, Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding]

Diffusion models

huggingface / diffusers Public

license Apache-2.0 | release v0.1.2 | Contributor Covenant 2.0

Diffusers provides pretrained diffusion models across multiple modalities, such as vision and audio, and serves as a modular toolbox for inference and training of diffusion models.

Diffusers: State-of-the-art diffusion models for image and audio generation in PyTorch

deep-learning · pytorch · image-generation · diffusion · score-based-generative-modeling

- Readme
- Apache-2.0 license
- 1.2k stars
- 42 watching
- 61 forks

Jack Hessel @jmhessel · Jun 11

"Oil painting of a group of confused people wondering what to do with the most complicated machine in the world" #dalle2

Outline

- History
- Method
    - In theory → in practice
    - “Diffusion as gradients”
- Controllable generation
- Examples: GLIDE, Dalle-2, Imagen

Diffusion probabilistic models
[Sohl-Dickstein et al, ICML 2015]

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein, Stanford University, JASCHA@STANFORD.EDU
Eric A. Weiss, University of California, Berkeley, EAWEISS@BERKELEY.EDU
Niru Maheswaranathan, Stanford University, NIRUM@STANFORD.EDU
Surya Ganguli, Stanford University, SGANGULI@STANFORD.EDU

Denoising diffusion probabilistic models (DDPM)
[Ho et al, Neurips 2020]

Denoising Diffusion Probabilistic Models

Jonathan Ho, UC Berkeley, jonathanho@berkeley.edu
Ajay Jain, UC Berkeley, ajayj@berkeley.edu
Pieter Abbeel, UC Berkeley, pabbeel@cs.berkeley.edu

Improvements & Applications, e.g.

Improved Denoising Diffusion Probabilistic Models

Alex Nichol [*1] Prafulla Dhariwal [*1]

ICML 2021

GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models

Alex Nichol [*] Prafulla Dhariwal [*] Aditya Ramesh [*] Pranav Shyam Pamela Mishkin Bob McGrew Ilya Sutskever Mark Chen

ICML 2022

Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Chitwan Saharia[*], William Chan[*], Saurabh Saxena[†], Lala Li[†], Jay Whang[†], Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho[†], David J Fleet[†], Mohammad Norouzi[*]

{sahariac,williamchan,mnorouzi}@google.com
{srbs,lala,jwhang,jonathanho,davidfleet}@google.com

Google Research, Brain Team
Toronto, Ontario, Canada

Arxiv May 2022

A Connection Between Score Matching and Denoising Autoencoders

Pascal Vincent
vincentp@iro.umontreal.ca
Dept. IRO, Université de Montréal,
CP 6128, Succ. Centre-Ville, Montréal (QC) H3C 3J7, Canada.

Denoising score matching
[Vincent, 2010]

Generative Modeling by Estimating Gradients of the Data Distribution

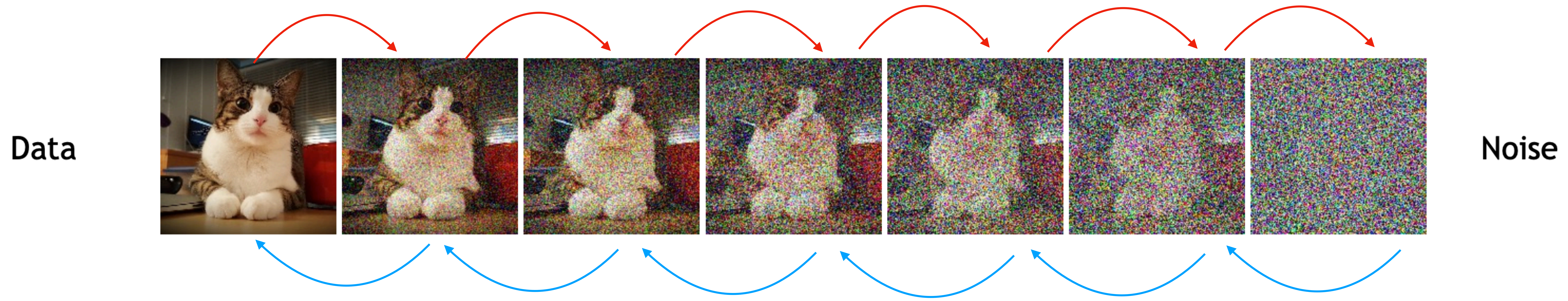
Yang Song, Stanford University, yangsong@cs.stanford.edu
Stefano Ermon, Stanford University, ermon@cs.stanford.edu

Noise-conditioned score networks
[Song & Ermon, Neurips 2019]

MANY more in [CVPR 2022 Tutorial]!!

Denoising diffusion models

- Destroy structure in data by progressively adding noise (“**diffusion**”)

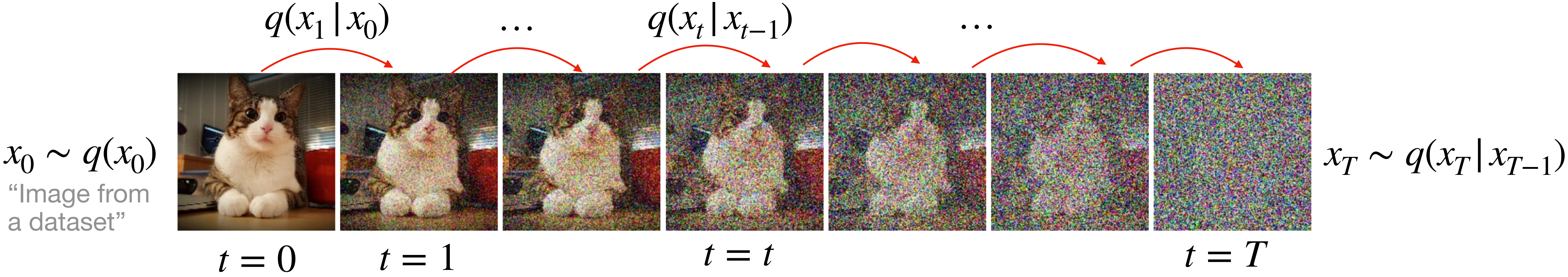


- Learn to reverse the diffusion process (“**denoising**”)
  - This gives us a model of the data & a way to generate
- *Intuition: modeling small changes is easier than directly modeling the data*

Image: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Forward diffusion process**: adds noise to data



- $q(x_t | x_{t-1}) = \mathcal{N}(x_t \; ; \; \text{mean}_t, \text{variance}_t)$
  - $\text{mean}_t = \sqrt{(1 - \beta_t)} x_{t-1}$
  - $\text{variance}_t = \beta_t I$

Forward process variances β_t, determined by a schedule $\beta_1, \ldots, \beta_T$, e.g. linear from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ in [Ho et al 2022].

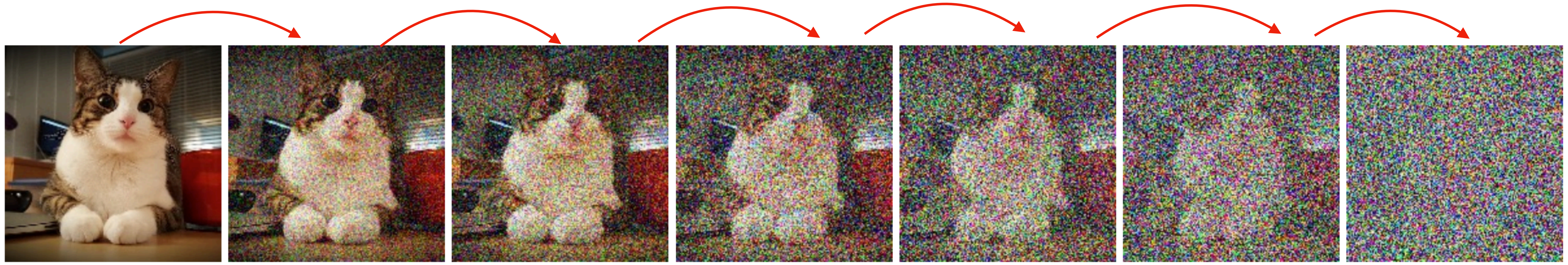
Content & image based on: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Forward diffusion process:** adds noise to data

$q(x_1 | x_0)$ $\dots$ $q(x_t | x_{t-1})$ $\dots$

$x_0 \sim q(x_0)$



$x_T \sim q(x_T | x_{T-1})$

$t = 0$ $t = 1$ $t = t$ $t = T$

- $q(x_{0:T}) = q(x_0) \prod_{t=1}^{T} q(x_t | x_{t-1})$

Content & image based on: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Forward diffusion process**: adds noise to data

$q(x_t | x_0)$

$x_0 \sim q(x_0)$

Noise

$t = 0$ $t = t$ $t = T$

- $q(x_t | x_0) = \mathcal{N}(x_t \; ; \; \sqrt{\bar{\alpha}_t} x_0, \sqrt{(1 - \bar{\alpha}_t)} I)$
- $\bar{\alpha}_t = \prod_{t'=1}^{t} (1 - \beta_{t'})$

- This lets us sample x_t given x_0 and noise schedule:

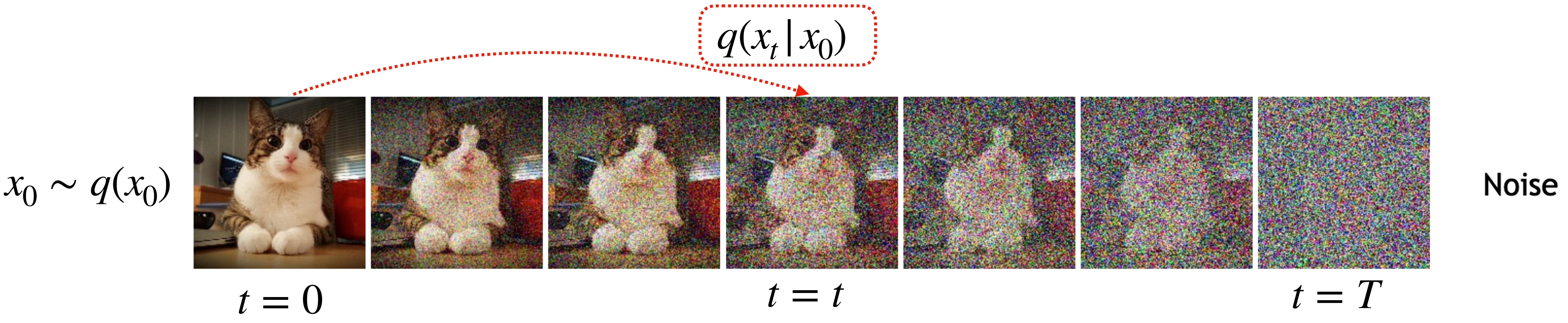
$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, I)$. "the re-parameterization trick"

Content & image based on: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Forward diffusion process**: adds noise to data



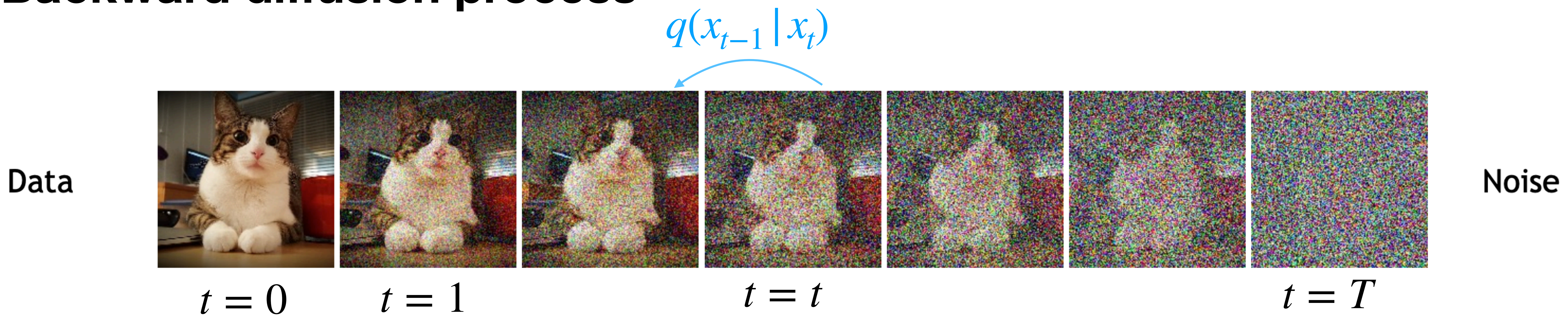
- $q(x_t | x_0) = \mathcal{N}(x_t \; ; \; \sqrt{\bar{\alpha}_t} x_0, \sqrt{(1 - \bar{\alpha}_t)} I)$
  - Noise schedule (choice of β_t, which determines $\bar{\alpha}_t$) designed so that:
    - $\sqrt{\bar{\alpha}_t}$ decreases towards zero
    - Final step is standard Gaussian noise, $q(x_T | x_0) \approx \mathcal{N}(x_T; 0, I)$

Example: linear from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ in [Ho et al 2022].

Content & image based on: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Backward diffusion process**

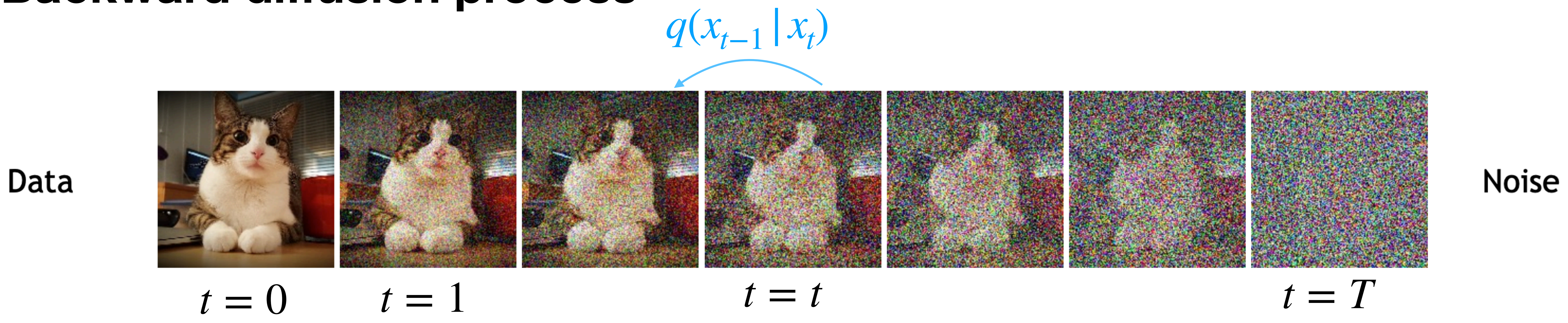


- If we knew how to "reverse time", we could generate data:
  - $x_T \sim N(0,I)$
  - Iteratively sample $x_{t-1} \sim q(x_{t-1} | x_t)$

Image: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Backward diffusion process**

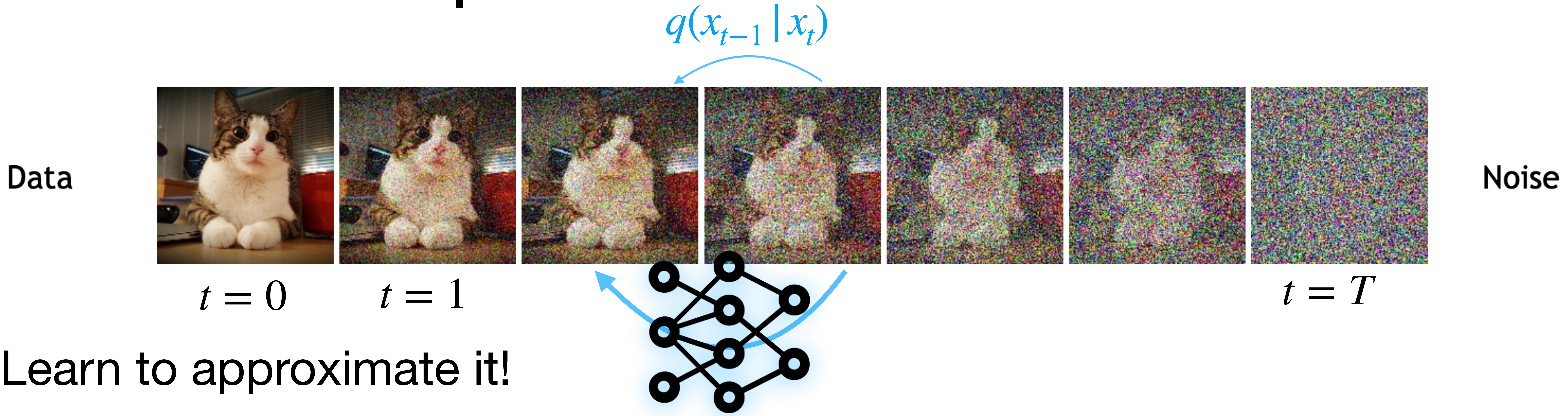


- Adding noise is easy, but we don't know how to "reverse time"
  - $q(x_{t-1} | x_t) \propto q(x_{t-1}) q(x_t | x_{t-1})$

Image: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Backward diffusion process**

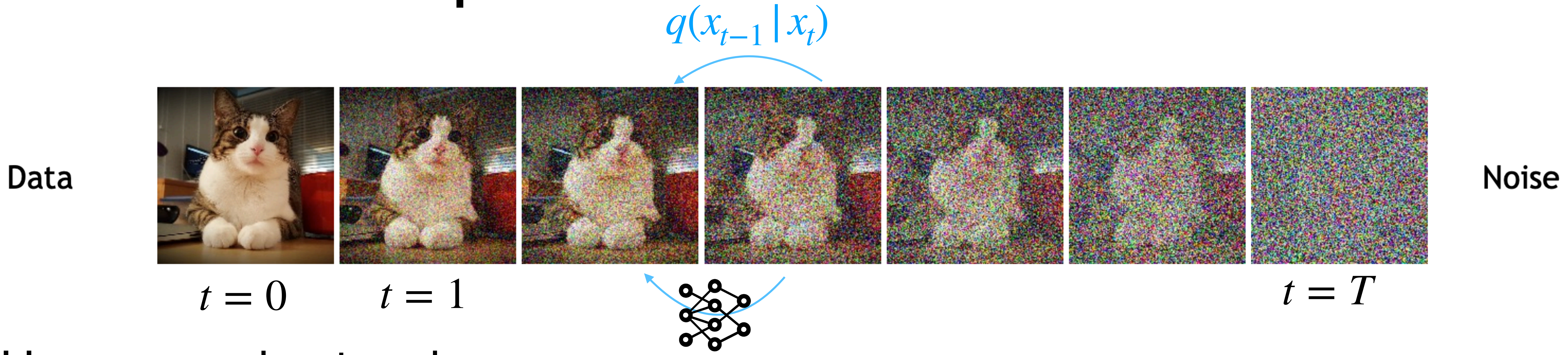


- Learn to approximate it!
    - $q(x_{t-1} | x_t) \approx p_\theta(x_{t-1} | x_t)$

Image: [CVPR 2022 Tutorial]

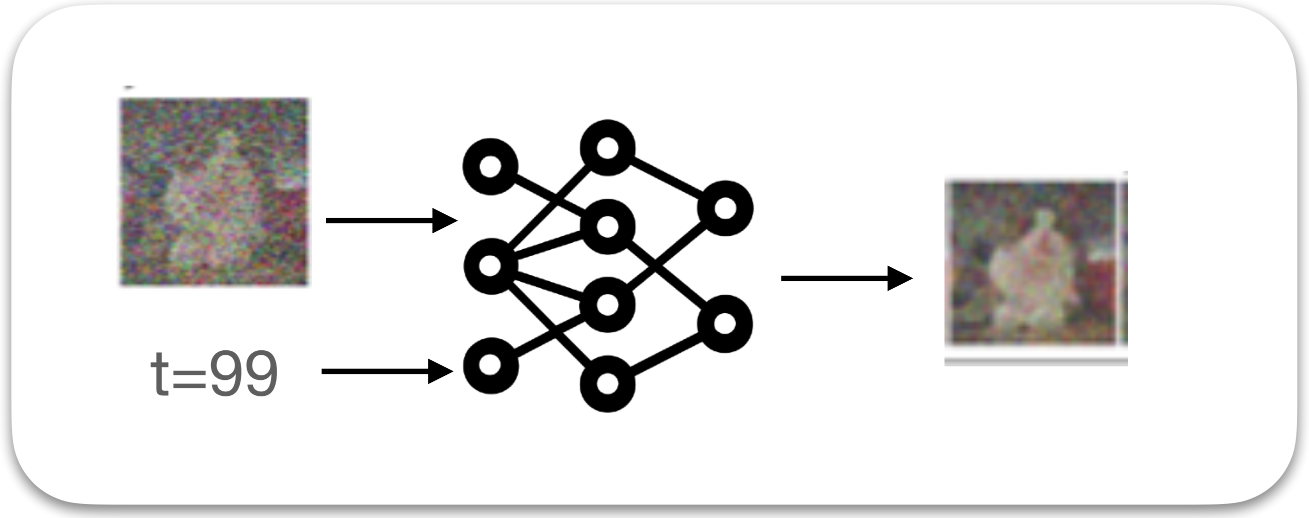
Denoising diffusion models

- **Backward diffusion process**



- Use a neural network:
  - $p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_{t-1} ; \mu_\theta(x_t, t), \sigma_t^2 I)$

Reverse process variance ("reverse noise schedule")
E.g set to β_t [Ho et al 2020], learned [Nichol & Dhariwal 2021]

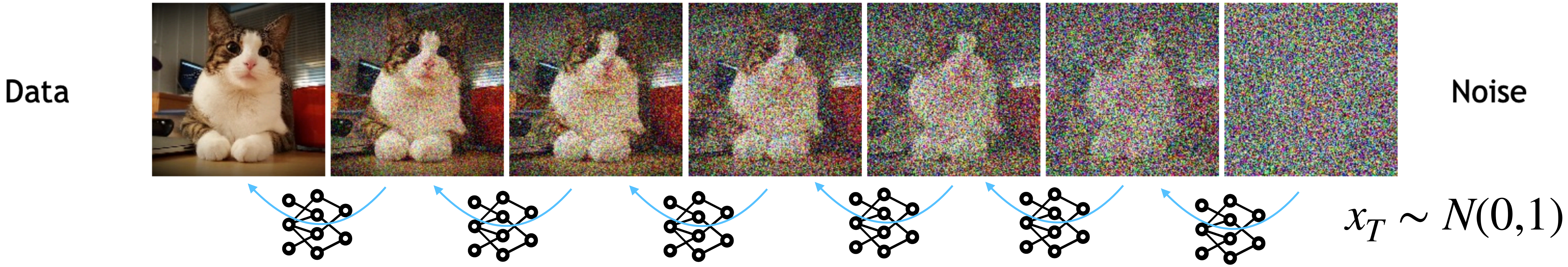


often a U-Net architecture

Image: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Backward diffusion process**



- Iteratively sample $x_{t-1} \sim p_\theta(x_{t-1} | x_t)$
  - $\rightarrow x_0$ is a cat image

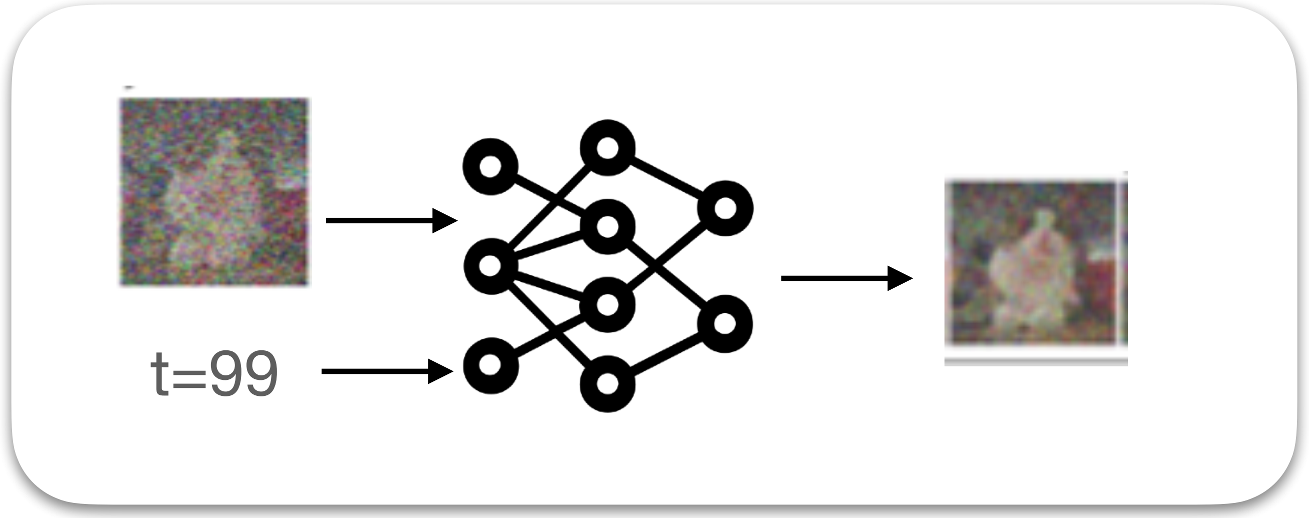
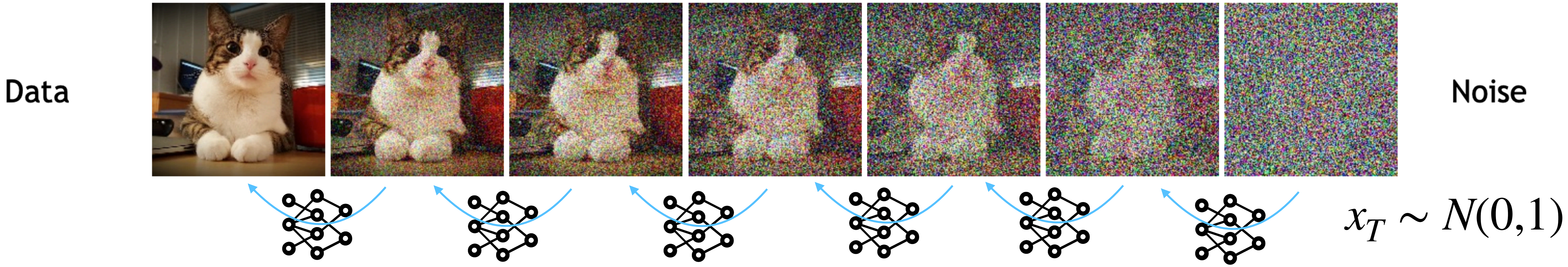


Image: [CVPR 2022 Tutorial]

Denoising diffusion models

- **Backward diffusion process**



- Gives us a model of the process:

- $$p_\theta(x_{0:N}) = p(x_N)\prod_{t=1}^{N} p_\theta(x_{t-1} \mid x_t)$$

Image: [CVPR 2022 Tutorial]

How do we learn such a network?

- Maximize marginal log-likelihood:
  - $\mathbb{E}_{x_0 \sim q(x_0)} - \log p_\theta(x_0) \leq \mathbb{E}_{q(x_0)q(x_{1:T}|x_0)} \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right]$

Variational bound (similar to VAEs)

  - … derivation …

See Appendix A of [Ho et al 2020]

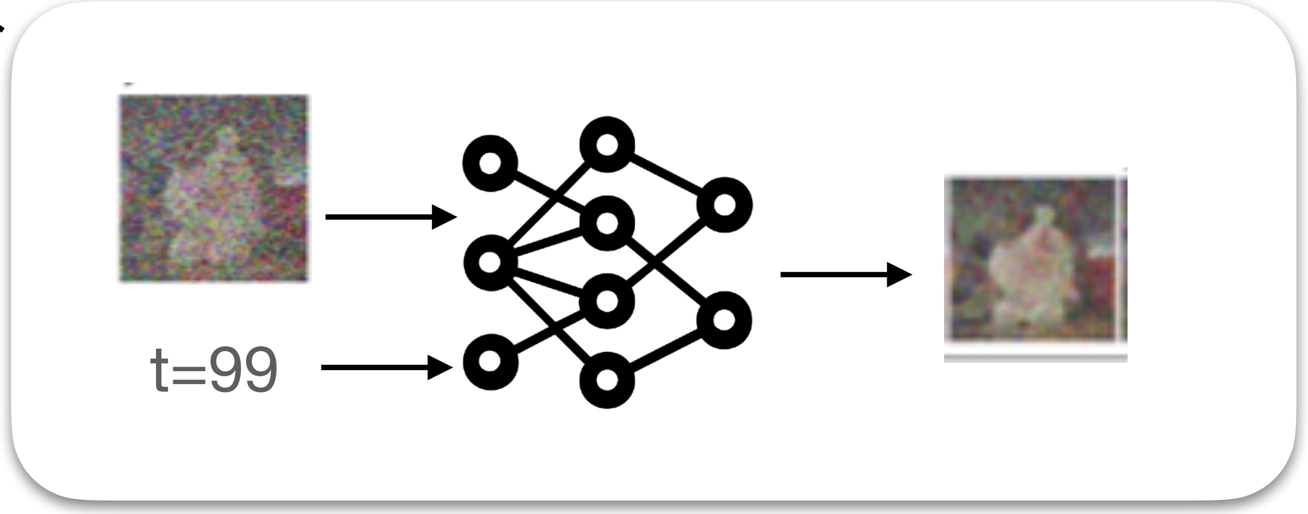
  - $\implies L = \mathbb{E}\left[\mathrm{KL}(q(x_{t-1} \mid x_t, x_0) \| p_\theta(x_{t-1} \mid x_t) \right]$
  - $\propto \mathbb{E}_t \left[\| \mu_t(x_t, x_0) - \mu_\theta(x_t, t) \|^2 \right]$

Gaussian → MSE Loss!!

The "target" mean can be computed given x_0, x_t and noise schedule:

$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\mu_t(x_t, x_0), \tilde{\beta}_t I)$, where

$$\mu_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \qquad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$



How do we learn such a network?

In practice

- $L = \mathbb{E}\left[||\mu_t(x_t, x_0) - \mu_\theta(x_t, t)||^2\right]$. First, [Ho et al 2020] observe that μ_t can be rewritten as:

See eqn. (10) in [Ho et al 2020]

$$\mu_t(x_t, x_0) = \underbrace{\frac{1}{\sqrt{\alpha_t}}}_{a_t}\left(x_t - \underbrace{\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}}_{b_t}\epsilon\right).$$

- Since x_t, a_t, b_t are given, [Ho et al 2020] *predict noise* $\epsilon \approx e_\theta(x_t, t)$:
    - $\mu_\theta(x_t, t) = a_t(x_t - b_t\epsilon_\theta(x_t, t))$
- The objective simplifies to:
    - $\tilde{L} = \mathbb{E}\left[\lambda_t||\epsilon - \epsilon_\theta(x_t, t)||^2\right]$, where $\lambda_t = \dfrac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}$

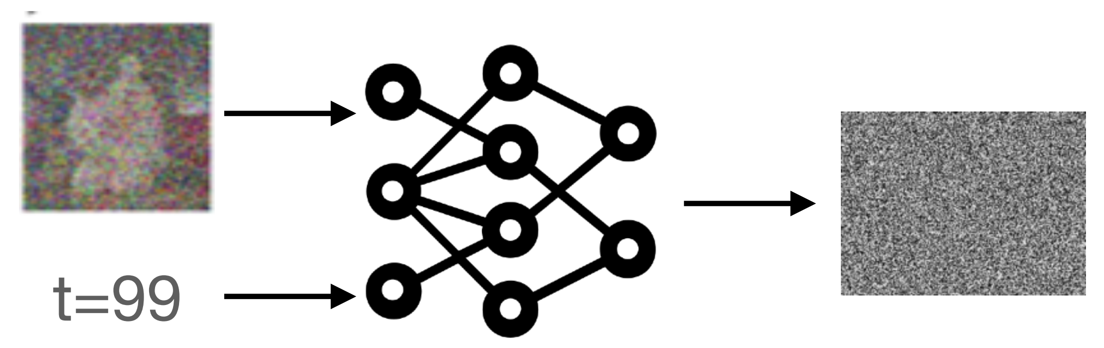
eqn. (12) in [Ho et al 2020]

- Finally, [Ho et al 2020] drop the weighting term λ_t:
    - $\mathscr{L}_{\text{simple}} = \mathbb{E}_{t, x_0, \epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right]$

eqn. (14) in [Ho et al 2020]

Recall that $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon,$

Intuitively, the network only predicts the “unknown” part



We will see later that there is a connection between predicting the noise and predicting a gradient

Algorithm summary

Training

Algorithm 1 Training

repeat

$\mathbf{x}_0 \sim q(\mathbf{x}_0)$

$t \sim \text{Uniform}(\{1, \ldots, T\})$

$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

until converged

MSE loss on
(actual noise, predicted noise)

$$x_t \sim q(x_t | x_0)$$

Recall that:

$$q(x_t | x_0) = \mathcal{N}(x_t \,;\, \sqrt{\bar{\alpha}_t} x_0, \sqrt{(1 - \bar{\alpha}_t)} I)$$

This lets us sample x_t using the re-parameterization trick:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, I)$$

Algorithm summary

Show me the code!

huggingface / diffusers Public

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged

```
# Sample noise that we'll add to the images
noise = torch.randn(clean_images.shape).to(clean_images.device)
bsz = clean_images.shape[0]
```

[source]

Algorithm summary

Show me the code!

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

```
# Sample noise that we'll add to the images
noise = torch.randn(clean_images.shape).to(clean_images.device)
bsz = clean_images.shape[0]
# Sample a random timestep for each image
timesteps = torch.randint(
    0, noise_scheduler.num_train_timesteps, (bsz,), device=clean_images.device
).long()
```

Algorithm summary

Show me the code!

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

$$x_t \sim q(x_t | x_0)$$

```python
# Sample noise that we'll add to the images
noise = torch.randn(clean_images.shape).to(clean_images.device)
bsz = clean_images.shape[0]
# Sample a random timestep for each image
timesteps = torch.randint(
    0, noise_scheduler.num_train_timesteps, (bsz,), device=clean_images.device
).long()

# Add noise to the clean images according to the noise magnitude at each timestep
# (this is the forward diffusion process)
noisy_images = noise_scheduler.add_noise(clean_images, noise, timesteps)
```

Algorithm summary

Show me the code!

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

```
# Sample noise that we'll add to the images
noise = torch.randn(clean_images.shape).to(clean_images.device)
bsz = clean_images.shape[0]
# Sample a random timestep for each image
timesteps = torch.randint(
    0, noise_scheduler.num_train_timesteps, (bsz,), device=clean_images.device
).long()

# Add noise to the clean images according to the noise magnitude at each timestep
# (this is the forward diffusion process)
noisy_images = noise_scheduler.add_noise(clean_images, noise, timesteps)
```

```
# Predict the noise residual
noise_pred = model(noisy_images, timesteps)["sample"]
loss = F.mse_loss(noise_pred, noise)
accelerator.backward(loss)
```

Algorithm summary

Show me the code!

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

```
# Sample noise that we'll add to the images
noise = torch.randn(clean_images.shape).to(clean_images.device)
bsz = clean_images.shape[0]
# Sample a random timestep for each image
timesteps = torch.randint(
    0, noise_scheduler.num_train_timesteps, (bsz,), device=clean_images.device
).long()

# Add noise to the clean images according to the noise magnitude at each timestep
# (this is the forward diffusion process)
noisy_images = noise_scheduler.add_noise(clean_images, noise, timesteps)
```

```
# Predict the noise residual
noise_pred = model(noisy_images, timesteps)["sample"]
loss = F.mse_loss(noise_pred, noise)
accelerator.backward(loss)
```

Algorithm summary

Sampling

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

$$\mu_\theta(x_t, x_0) = a_t(x_t - b_t \epsilon_\theta(x_t, x_0))$$

Diffusion ≈ gradients

Diffusion probabilistic models
[Sohl-Dickstein et al, ICML 2015]

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University
Eric A. Weiss EAWEISS@BERKELEY.EDU
University of California, Berkeley
Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University
Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

Denoising diffusion probabilistic models (DDPM)
[Ho et al, Neurips 2020]

Denoising Diffusion Probabilistic Models

Jonathan Ho UC Berkeley jonathanho@berkeley.edu
Ajay Jain UC Berkeley ajayj@berkeley.edu
Pieter Abbeel UC Berkeley pabbeel@cs.berkeley.edu

A Connection Between Score Matching and Denoising Autoencoders

Pascal Vincent
vincentp@iro.umontreal.ca
Dept. IRO, Université de Montréal,
CP 6128, Succ. Centre-Ville, Montréal (QC) H3C 3J7, Canada.

Denoising score matching
[Vincent, 2010]

Generative Modeling by Estimating Gradients of the Data Distribution

Yang Song Stanford University yangsong@cs.stanford.edu
Stefano Ermon Stanford University ermon@cs.stanford.edu

Noise-conditioned score networks
[Song & Ermon, Neurips 2019]

Diffusion ≈ gradients

"Score" : $\nabla_x \log p(x)$

p_t: perturb data with noise-level t

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

Denoising Diffusion Probabilistic Model

$$\mathcal{L}_{\text{score}} = \mathbb{E}_{t,x}\left[\|\nabla_x \log p_t(x) - s_\theta(x, t)\|^2\right]$$

Noise-Conditional Score-networks

[Ho et al 2020] show: $\nabla_{x_t} \log p_t(x_t) \propto e_\theta(x_t, t)$

(also see 2.2 of [Song et al 2021])

- Noise ≈ gradient of log-prob ("score")
- Noise predictor ≈ score predictor
- Intuition: both things learn to predict small changes to noisy data

For a great overview from the author, see Yang Song's blog: https://yang-song.github.io/blog/2021/score

Diffusion sampling ≈ Langevin dynamics

$$x_t = x_{t-1} - \frac{\lambda}{2}\nabla_x E_\theta(x_{t-1}) + \sigma z$$

Langevin Dynamics [Welling & Teh 2011]

Denoising Diffusion Probabilistic Model

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$$

NCSN Annealed Langevin Dynamics

$$\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$$

- Diffusion sampling ≈ annealed Langevin dynamics with learned gradient

https://yang-song.github.io/blog/2021/score/#introduction

Method Recap

- Diffusion models: learn to iteratively denoise data, $p_\theta(x_{t-1} | x_t)$
  - Original formulation: variational inference
  - Practical formulation: similar to denoising score matching
- Intuitively, both learn small changes to noisy data at various noise scales

Guided generation

- Goal: conditional generation

  e.g. $p(x \mid y)$ where y is a class label (e.g. cat) or a text prompt

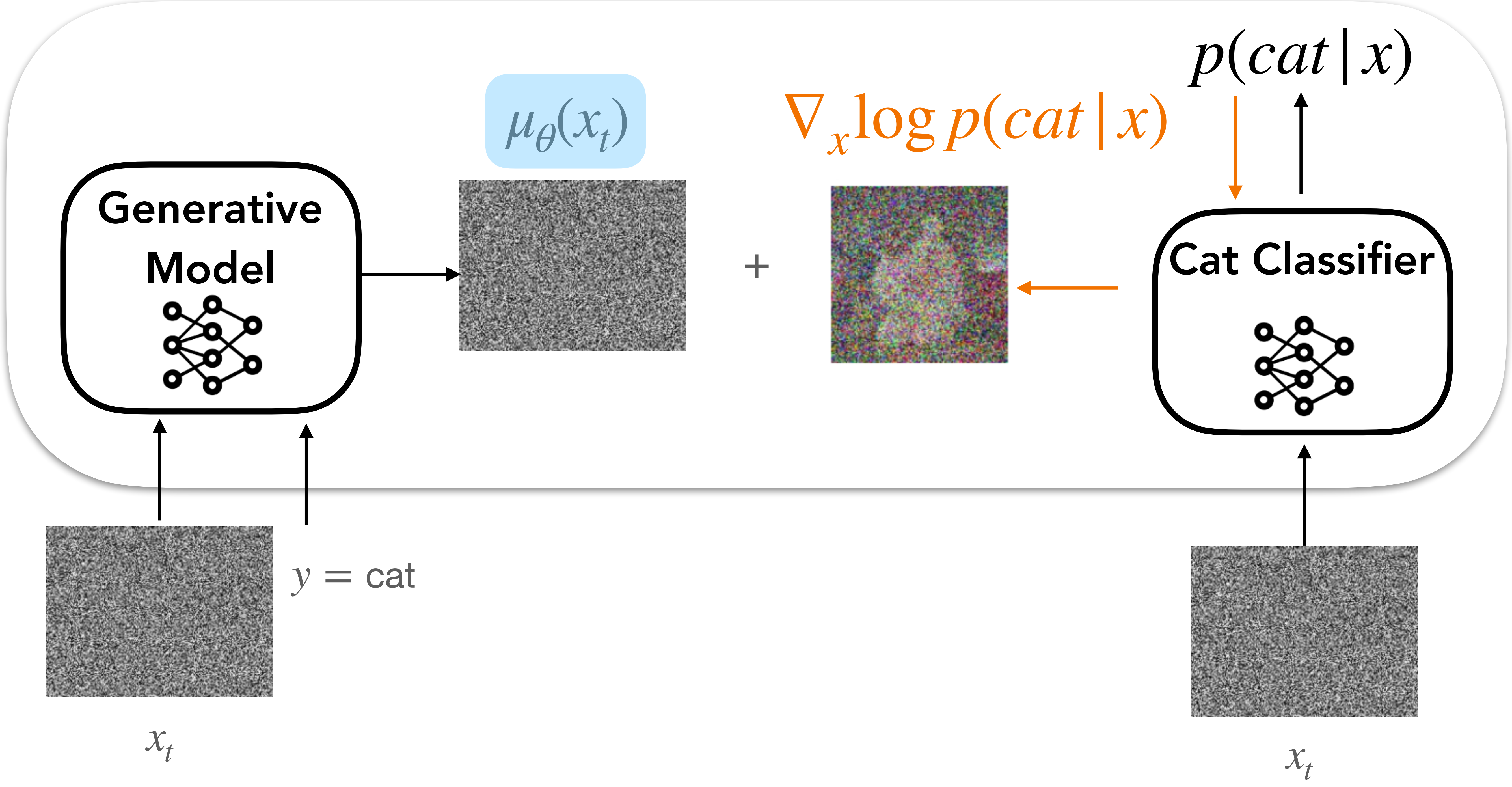
Classifier guidance

At each step of sampling:

$$x_{t-1} \leftarrow \text{sample from } \mathcal{N}(\mu_\theta(x_t) + s \cdot \nabla_{x_t} \log p_\phi(y \mid x_t), \sigma_t^2)$$

Classifier is trained on noisy images x_t

Approximates $\propto p(x_t \mid y) p(y \mid x_t)^s$



[Dhariwal & Nichol 2021, Diffusion Models Beat GANs on Image Synthesis, NeurIPS 2021]

Classifier guidance

At each step of sampling:

Classifier is trained on noisy images x_t

$$x_{t-1} \leftarrow \text{sample from } \mathcal{N}(\mu_\theta(x_t) + s \cdot \nabla_{x_t} \log p_\phi(y \mid x_t), \sigma_t^2)$$

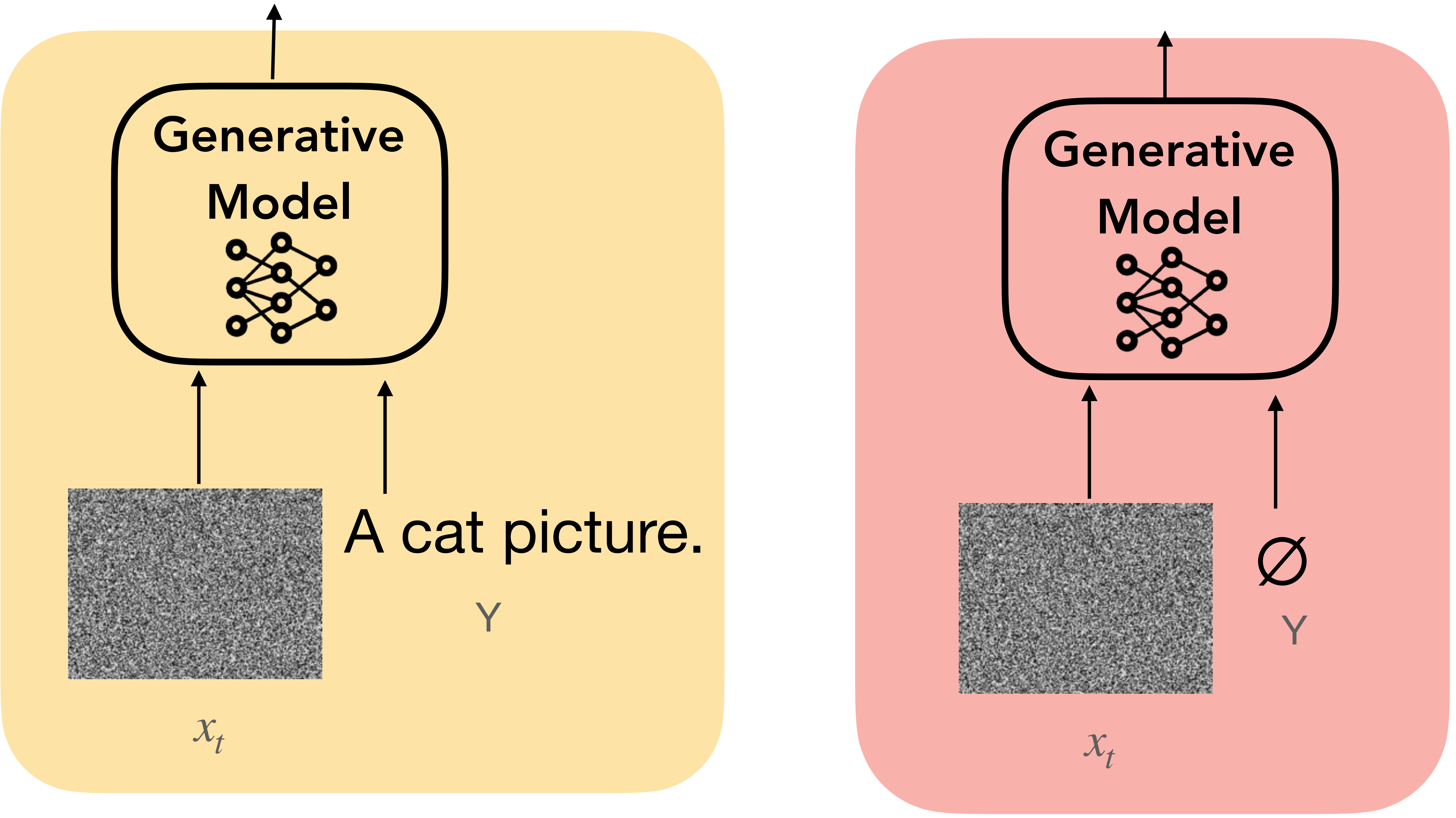
[Dhariwal & Nichol 2021, Diffusion Models Beat GANs on Image Synthesis, NeurIPS 2021]

Classifier-free guidance

At each step of sampling use:

$$\epsilon_\theta(x_t) + s \cdot \underbrace{(\epsilon_\theta(x_t \mid y) - \epsilon_\theta(x_t))}_{\text{'class relevant direction'}}$$

Generative model trained with & without conditioning info



[Ho & Salimans 2021, Classifier-Free Diffusion Guidance]

Classifier-free guidance

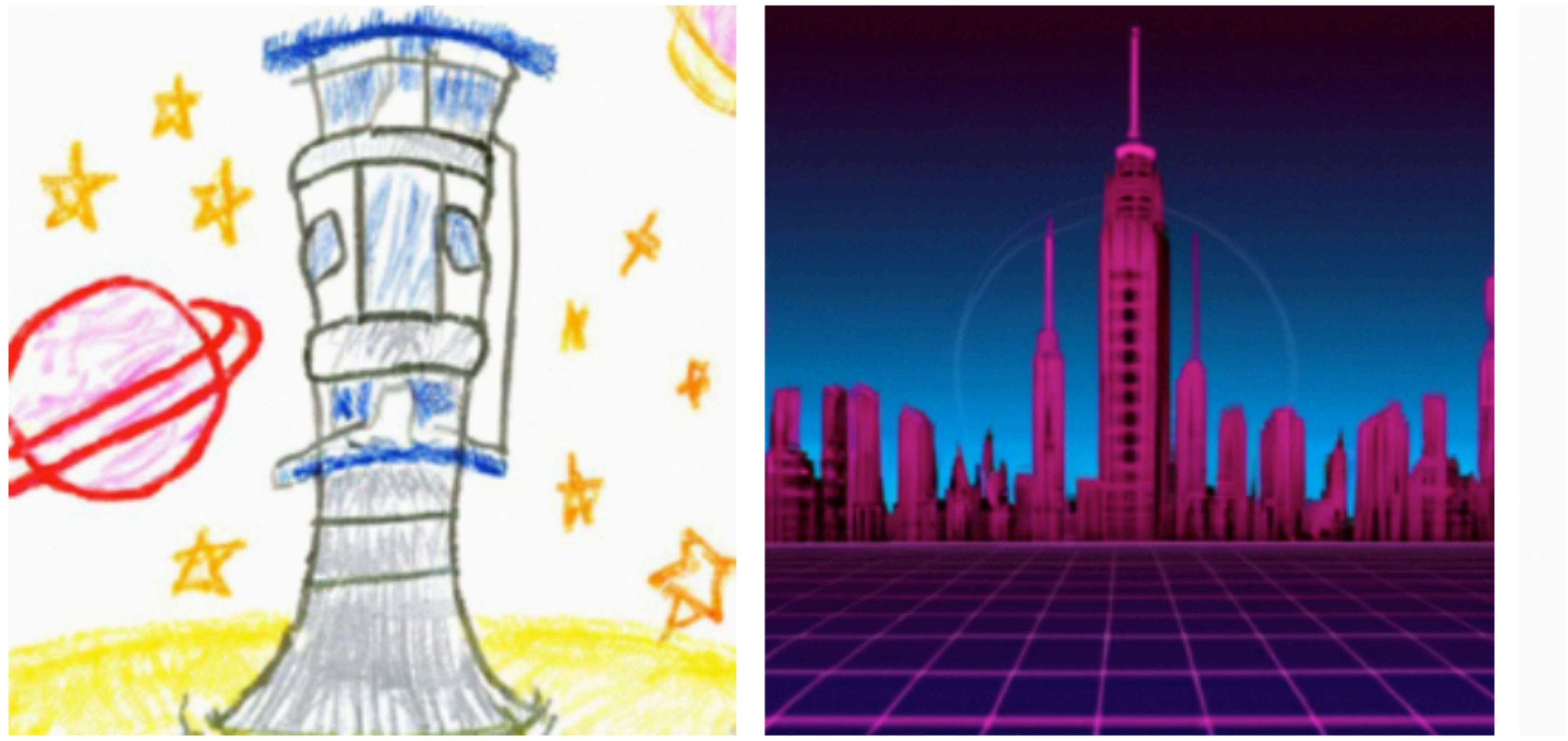
Figure 1: Unconditional guidance on the malamute class for a 64x64 ImageNet diffusion model. Left to right: increasing amounts of unconditional guidance, starting from non-guided samples on the left.

[Ho & Salimans 2021, Classifier-Free Diffusion Guidance]

Case study: GLIDE

- 3.5B text-conditional diffusion model
- Trained on DALL-E dataset

Guidance	Photorealism	Caption
Unguided	-88.6	-106.2
CLIP guidance	-73.2	29.3
Classifier-free guidance	**82.7**	**110.9**



"a crayon drawing of a space elevator" "a futuristic city in synthwave style"

Figure 1. Selected samples from GLIDE using classifier-free guidance.

"a cozy living room" "a painting of a corgi on the wall above a couch" "a round coffee table in front of a couch" "a vase of flowers on a coffee table" "a couch in the corner of a room"

Figure 3. Iteratively creating a complex scene using GLIDE. First, we generate an image for the prompt "a cozy living room", then use the shown inpainting masks and follow-up text prompts to add a painting to the wall, a coffee table, and a vase of flowers on the coffee table, and finally to move the wall up to the couch.

Nichol et al 2022, GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models

Case study: Dall-e 2

- Classifier-free guidance
- Prior: diffusion
- Decoder: diffusion
  - + 2 diffusion upsamplers

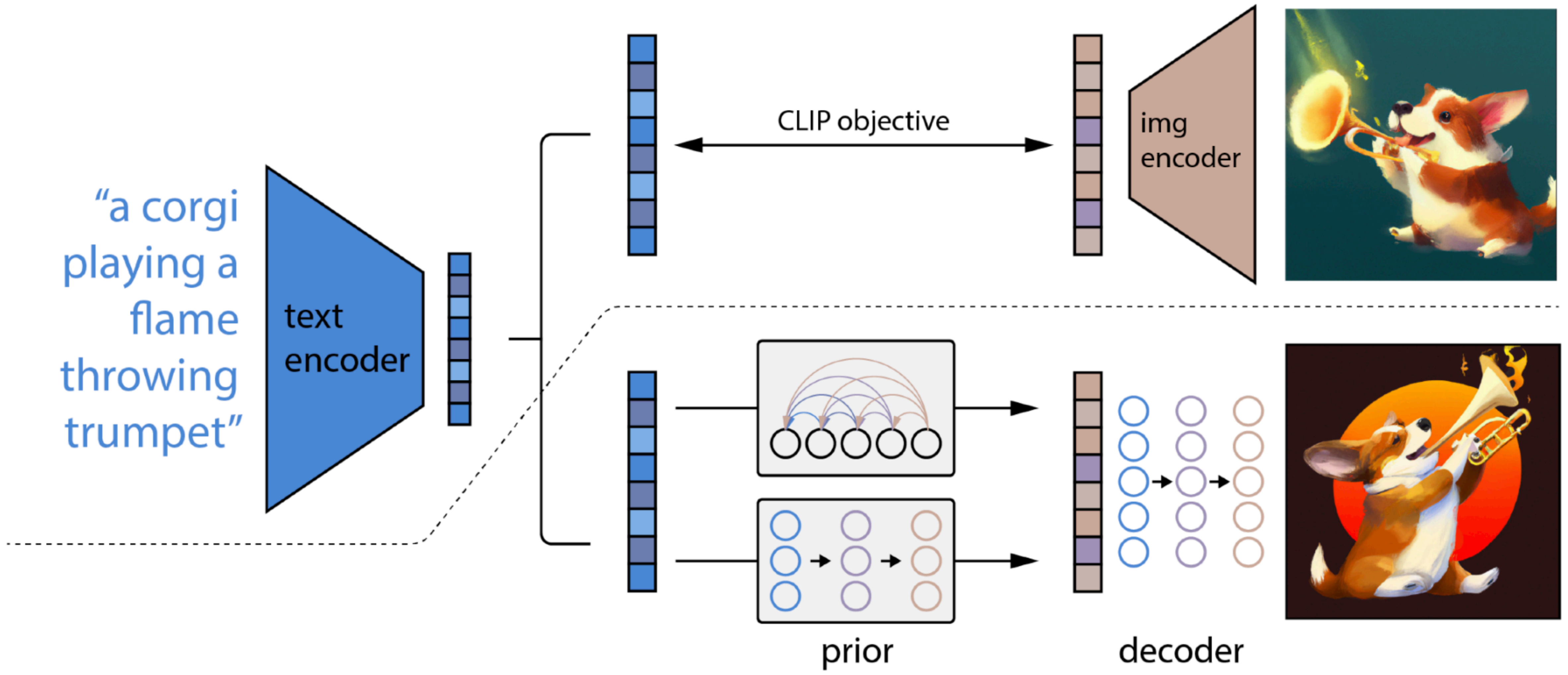


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

Ramesh et al 2022, Hierarchical Text-Conditional Image Generation with CLIP Latents

Case study: Dall-e 2

an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

Ramesh et al 2022, Hierarchical Text-Conditional Image Generation with CLIP Latents

Case study: Imagen

- Classifier-free guidance
- Frozen T5-XXL text encoder
- Cascaded diffusion
- 2B model
- ~460M internal image-text pairs
- ~400M public image-text pairs

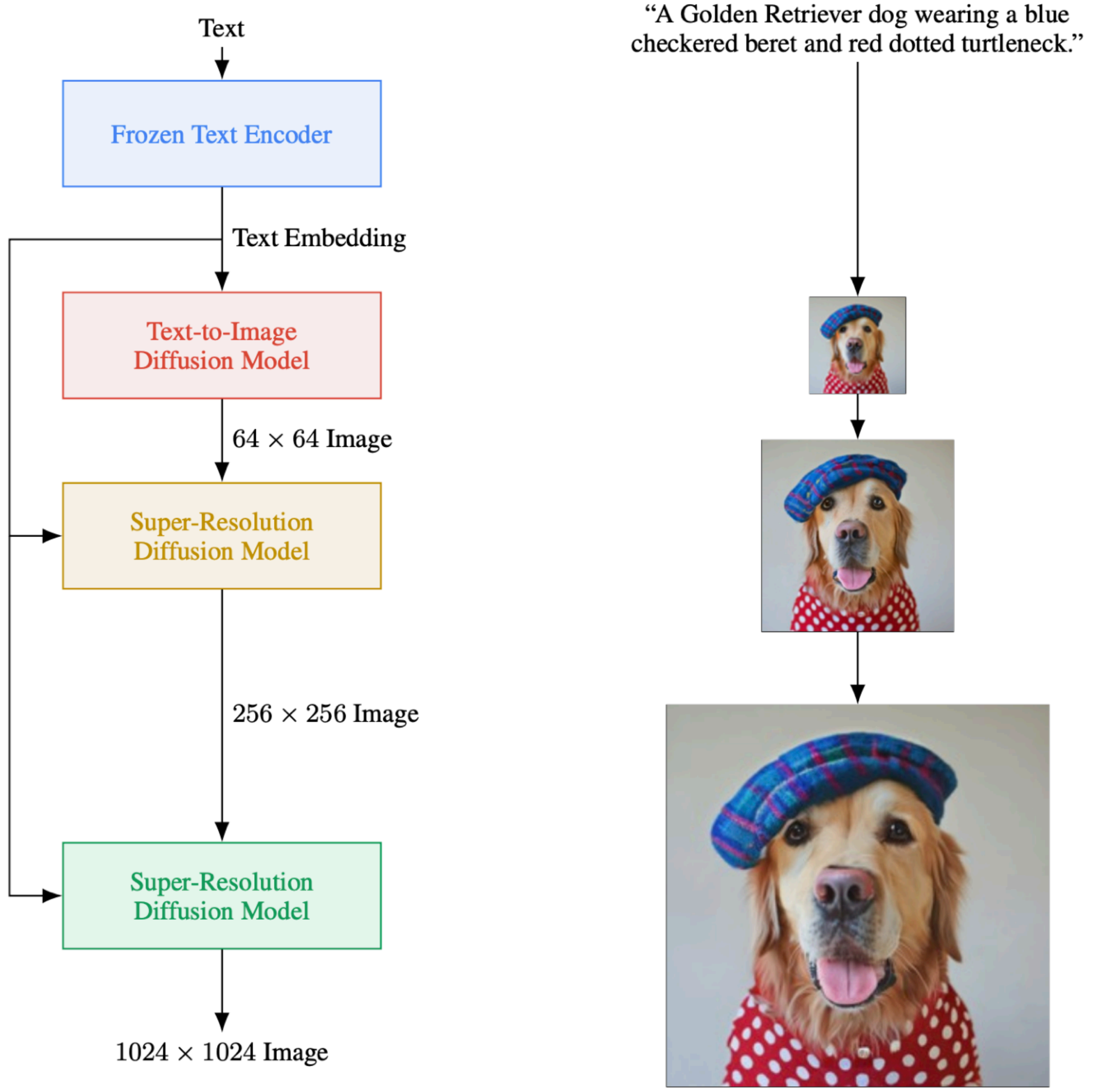


Figure A.4: Visualization of Imagen. Imagen uses a frozen text encoder to encode the input text into text embeddings. A conditional diffusion model maps the text embedding into a 64×64 image. Imagen further utilizes text-conditional super-resolution diffusion models to upsample the image, first $64 \times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

Saharia et al 2022, Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Case study: Imagen

A wall in a royal castle. There are two paintings on the wall. The one on the left a detailed oil painting of the royal raccoon king. The one on the right a detailed oil painting of the royal raccoon queen.

A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

A chrome-plated duck with a golden beak arguing with an angry turtle in a forest.

Saharia et al 2022, Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Case study: Imagen

Model	COCO FID ↓
Trained on COCO	
AttnGAN (Xu et al., 2017)	35.49
DM-GAN (Zhu et al., 2019)	32.64
DF-GAN (Tao et al., 2020)	21.42
DM-GAN + CL (Ye et al., 2021)	20.79
XMC-GAN (Zhang et al., 2021)	9.33
LAFITE (Zhou et al., 2021)	8.12
Make-A-Scene (Gafni et al., 2022)	7.55
Not trained on COCO	
DALL-E (Ramesh et al., 2021)	17.89
GLIDE (Nichol et al., 2021)	12.24
DALL-E 2 (Ramesh et al., 2022)	10.39
Imagen (Our Work)	7.27

Imagen attains a new state-of-the-art COCO FID.

Saharia et al 2022, Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

The end