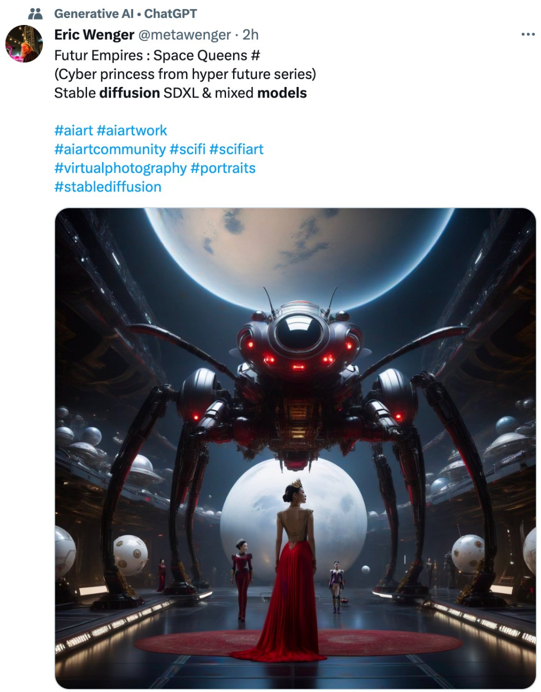
UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

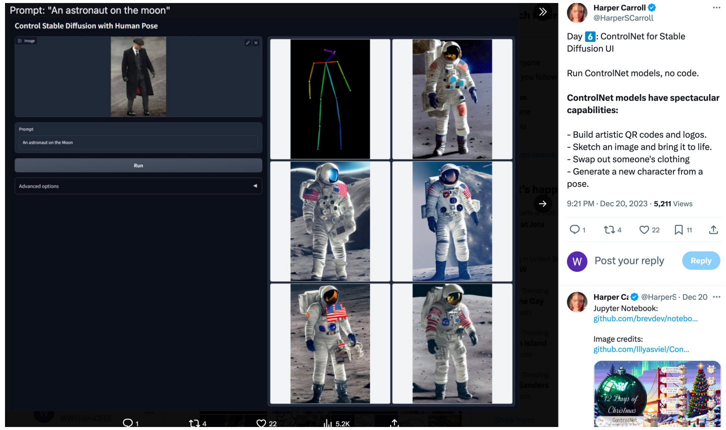
AIGC Tutorial

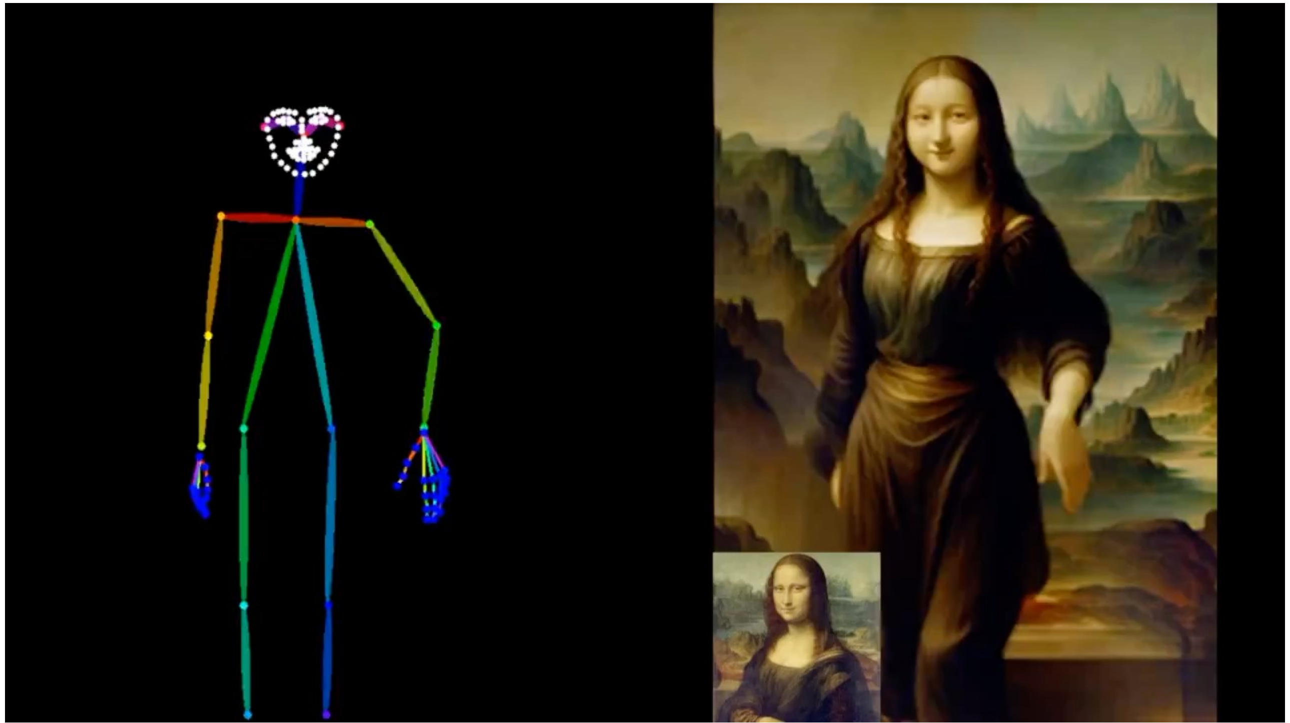
——A Introduction to Diffusion model

Weilong Chen
Electrical and Computer Engineering Department
University of Electronic Science and Technology of China
University of Houston, TX USA


Generative AI • ChatGPT
Eric Wenger @metawenger · 2h
Futur Empires : Space Queens #
(Cyber princess from hyper future series)
Stable diffusion SDXL & mixed models
#aiart #aiartwork
#aiartcommunity #scifi #scifiart
#virtualphotography #portraits
#stablediffusion


boat
teddy bear
castle
boots


Prompt: "An astronaut on the moon"
Control Stable Diffusion with Human Pose
Image
Prompt
An astronaut on the Moon
Run
Advanced options
Harper Carroll
@HarperSCarroll
Day 6: ControlNet for Stable Diffusion UI
Run ControlNet models, no code.
ControlNet models have spectacular capabilities:
- Build artistic QR codes and logos.
- Sketch an image and bring it to life.
- Swap out someone's clothing
- Generate a new character from a pose.
9:21 PM · Dec 20, 2023 · 5,211 Views
Post your reply
Reply
Harper C. @HarperS · Dec 20
Jupyter Notebook:
github.com/brevdev/notebo...
Image credits:
github.com/lllyasviel/Con...
12 Days of Christmas

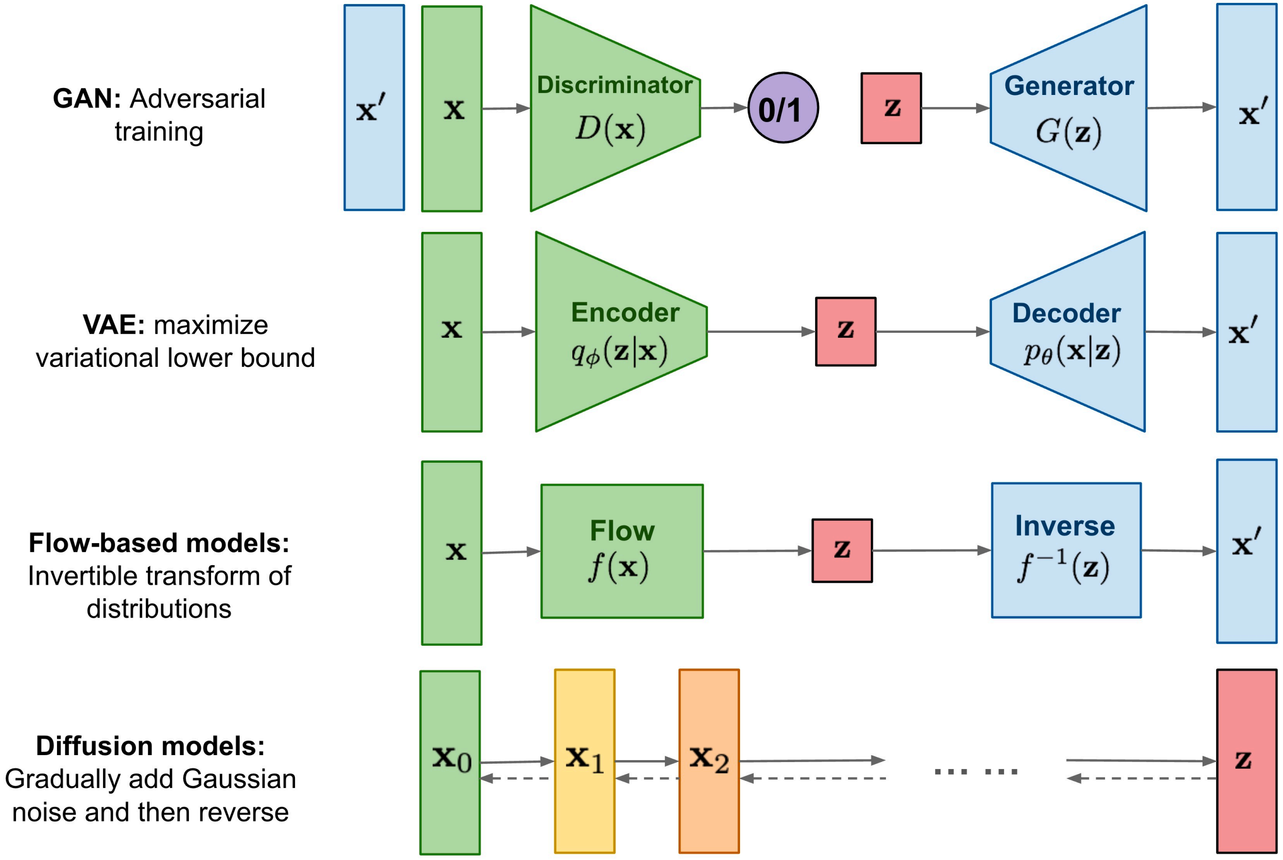


Outline

Diffusion Model

- DDPM, DDIM
- OpenAI help push the diffusion model, GLIDE, DALLE2
- Latent Diffusion Models and Latent Consistency Model
- How can we use it?
- Newest applications of the diffusion model

Overview

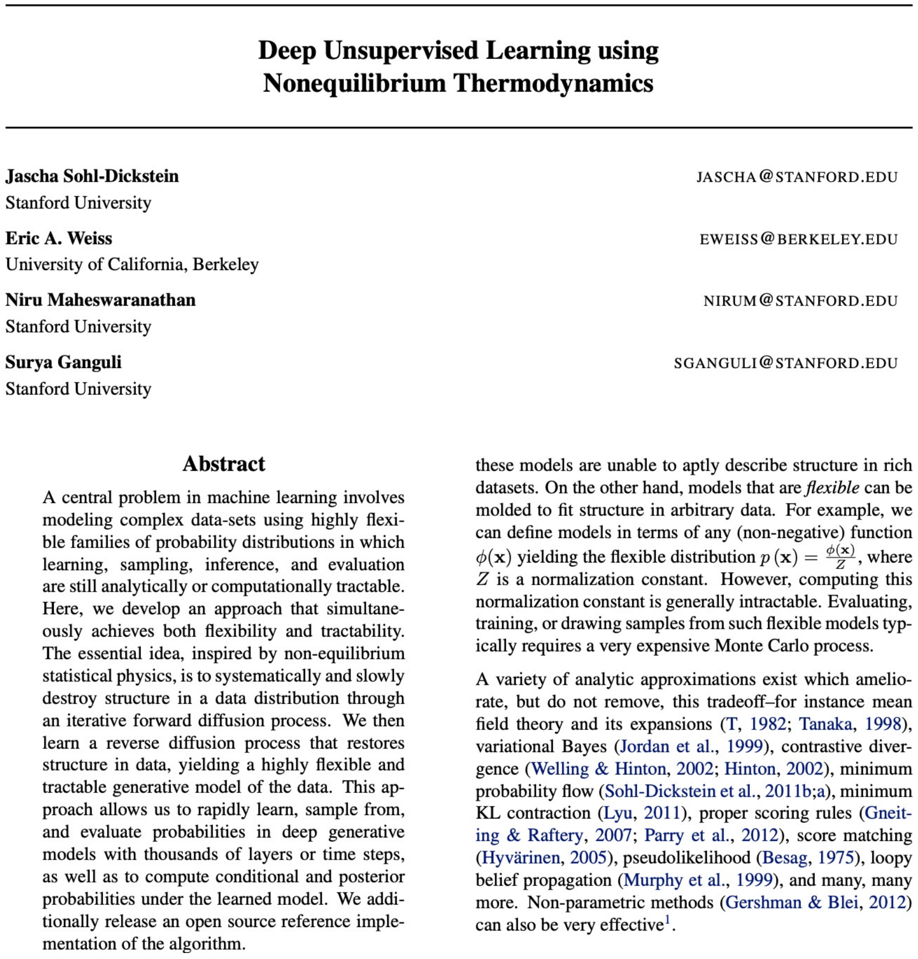


Diffusion models are inspired by non-equilibrium thermodynamics.

They define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise.

Unlike VAE or flow models, diffusion models are learned with a fixed procedure and the latent variable has high dimensionality (same as the original data).

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



Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University

Eric A. Weiss EWEISS@BERKELEY.EDU
University of California, Berkeley

Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University

Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

Abstract

A central problem in machine learning involves modeling complex data-sets using highly flexible families of probability distributions in which learning, sampling, inference, and evaluation are still analytically or computationally tractable. Here, we develop an approach that simultaneously achieves both flexibility and tractability. The essential idea, inspired by non-equilibrium statistical physics, is to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process. We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data. This approach allows us to rapidly learn, sample from, and evaluate probabilities in deep generative models with thousands of layers or time steps, as well as to compute conditional and posterior probabilities under the learned model. We additionally release an open source reference implementation of the algorithm.

these models are unable to aptly describe structure in rich datasets. On the other hand, models that are *flexible* can be molded to fit structure in arbitrary data. For example, we can define models in terms of any (non-negative) function $\phi(\mathbf{x})$ yielding the flexible distribution $p(\mathbf{x}) = \frac{\phi(\mathbf{x})}{Z}$, where Z is a normalization constant. However, computing this normalization constant is generally intractable. Evaluating, training, or drawing samples from such flexible models typically requires a very expensive Monte Carlo process.

A variety of analytic approximations exist which ameliorate, but do not remove, this tradeoff–for instance mean field theory and its expansions (T, 1982; Tanaka, 1998), variational Bayes (Jordan et al., 1999), contrastive divergence (Welling & Hinton, 2002; Hinton, 2002), minimum probability flow (Sohl-Dickstein et al., 2011b;a), minimum KL contraction (Lyu, 2011), proper scoring rules (Gneiting & Raftery, 2007; Parry et al., 2012), score matching (Hyvärinen, 2005), pseudolikelihood (Besag, 1975), loopy belief propagation (Murphy et al., 1999), and many, many more. Non-parametric methods (Gershman & Blei, 2012) can also be very effective[1].

Motivation: Estimating small perturbations is more tractable than explicitly describing the full data distribution.

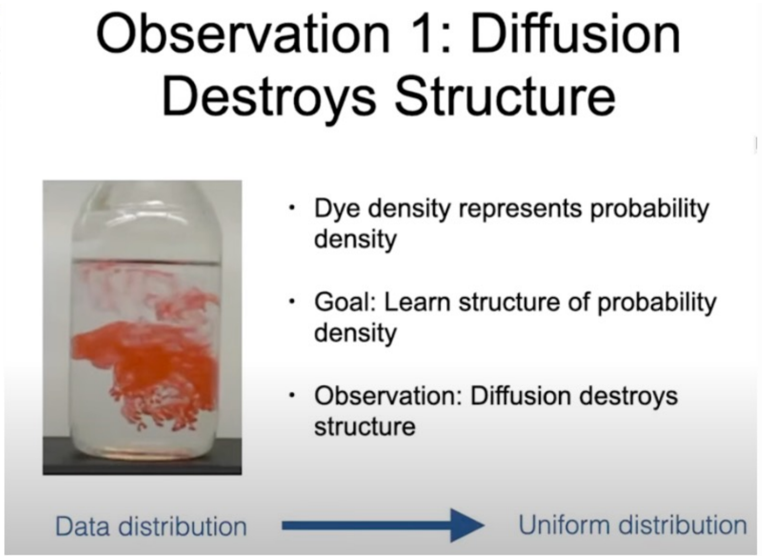
The essential idea is to:

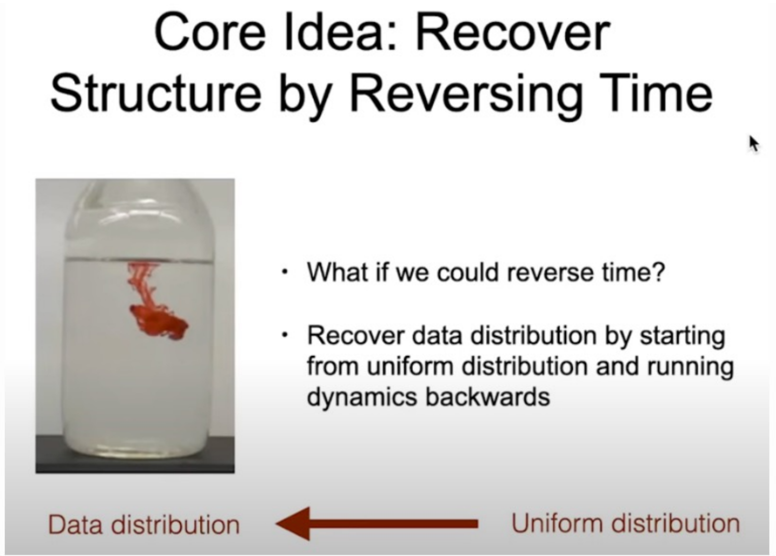
1. systematically and slowly destroy the structure in a data distribution through an iterative forward diffusion process.

2. learn a reverse diffusion process that restores data structure, yielding a highly flexible and tractable regenerative model.

https://proceedings.mlr.press/v37/sohl-dickstein15.html

https://www.youtube.com/watch?v=XCUlnHP1TNM&ab_channel=NickAliJahanian





DDPM

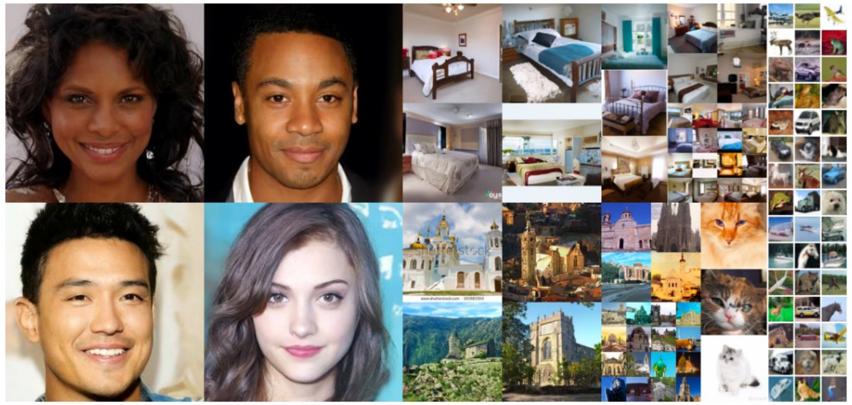
Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley

Ajay Jain
UC Berkeley

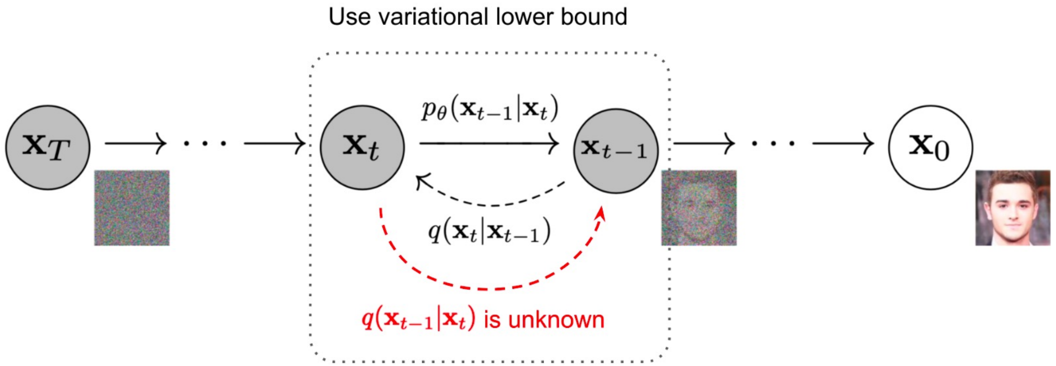
Pieter Abbeel
UC Berkeley

Paper (high-res, 98 MB) | Paper (arXiv, 10 MB) | GitHub



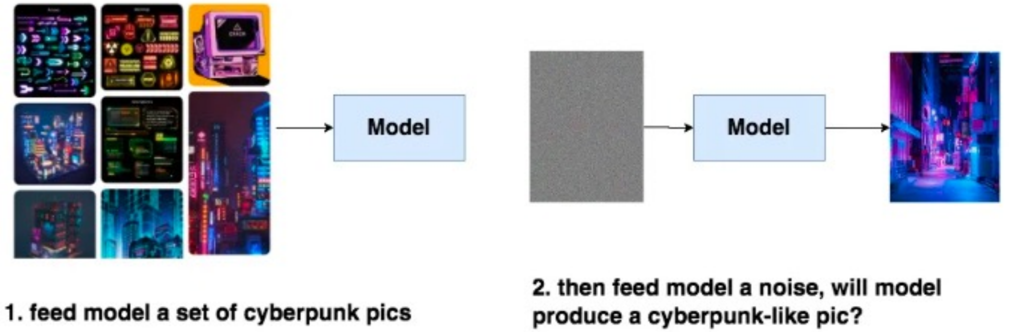
Images generated unconditionally by our probabilistic model.
These are not real people, places, animals or objects.

Demonstrate that diffusion models are capable of generating high quality samples.



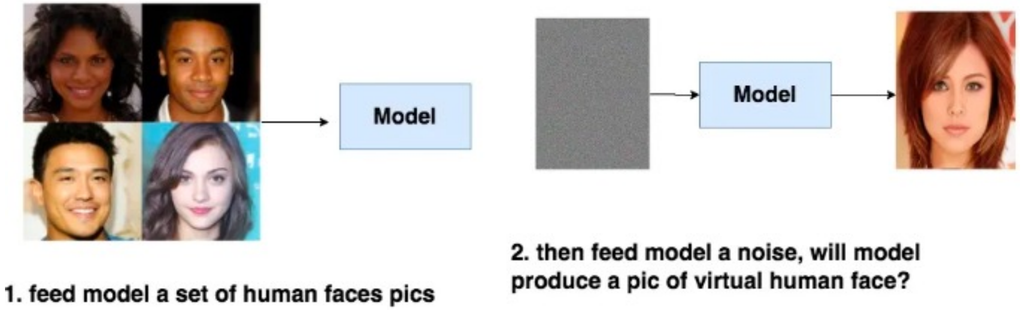
https://hojonathanho.github.io/diffusion/
https://zhuanlan.zhihu.com/p/637815071

DDPM



If you feed the model a bunch of cyberpunk-style pictures, let the model learn cyberpunk-style distribution information.

Then feed the model a random noise, you can make the model produce a realistic cyberpunk photo.



The essential role of DDPM is to learn the distribution of training data and produce real pictures that match the distribution of training data as much as possible.

https://zhuanlan.zhihu.com/p/637815071

DDPM

Diffusion process: adds Gaussian noise added for each step $q(x_t|x_{t-1})$

Denoise process: reverse the picture from noise $q(x_{t-1}|x_t)$

https://zhuanlan.zhihu.com/p/622800350

DDPM

Diffusion process: adds Gaussian noise added for each step $q(x_t|x_{t-1})$

$$x_t = x_{t-1} + \epsilon = x_0 + \epsilon_0 + \epsilon_1 + \ldots + \epsilon \longrightarrow x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_t$$

Given x_0, how to directly obtain x_t without sampling many times? $q(x_t|x_0)$

As the number of steps increases, the less original information content in the picture and the more noise there is. We can give the original picture and noise a weight to calculate:

$$x_t = \sqrt{\bar{a}_t} x_0 + \sqrt{1 - \bar{a}_t} \epsilon$$

$$\alpha_t = 1 - \beta_t$$

$$\bar{\alpha}_t = \alpha_1 \alpha_2 \ldots \alpha_t$$

β_t is a constant hyperparameter. As T increases, they become larger and larger.

https://zhuanlan.zhihu.com/p/622800350
https://zhuanlan.zhihu.com/p/637815071

DDPM

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Denoise process: reverse the picture from noise

$$q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t)$$

Denoise Process

https://zhuanlan.zhihu.com/p/637815071
https://www.zhangzhenhu.com/aigc/%E6%89%A9%E6%95%A3%E6%A6%82%E7%8E%87%E6%A8%A1%E5%9E%8B.html

DDPM

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

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon$$

The noise by the sample at the t-th time $\epsilon \sim \mathcal{N}(0, I)$ is our noise ground truth.

The predicted noise is: $\epsilon_\theta(\sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon, t)$

Regardless of any input data or any step, the model is to predict a noise from a Gaussian distribution.

Algorithm 2 Sampling

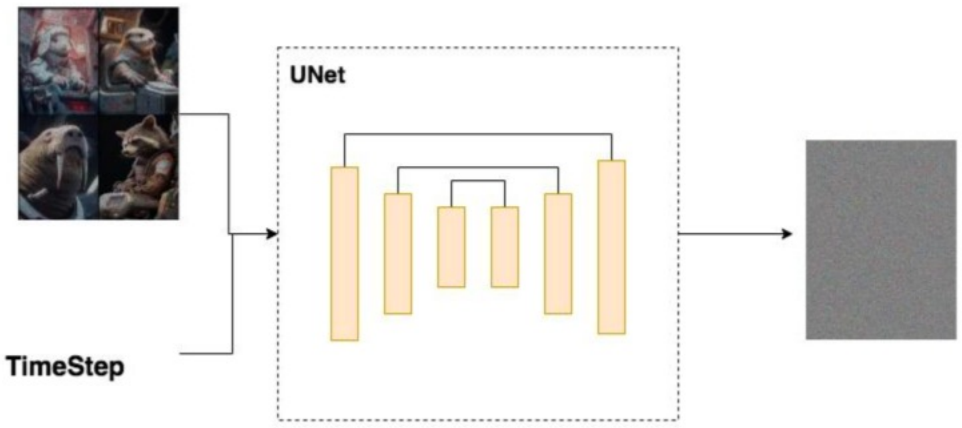
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\boxed{\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,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

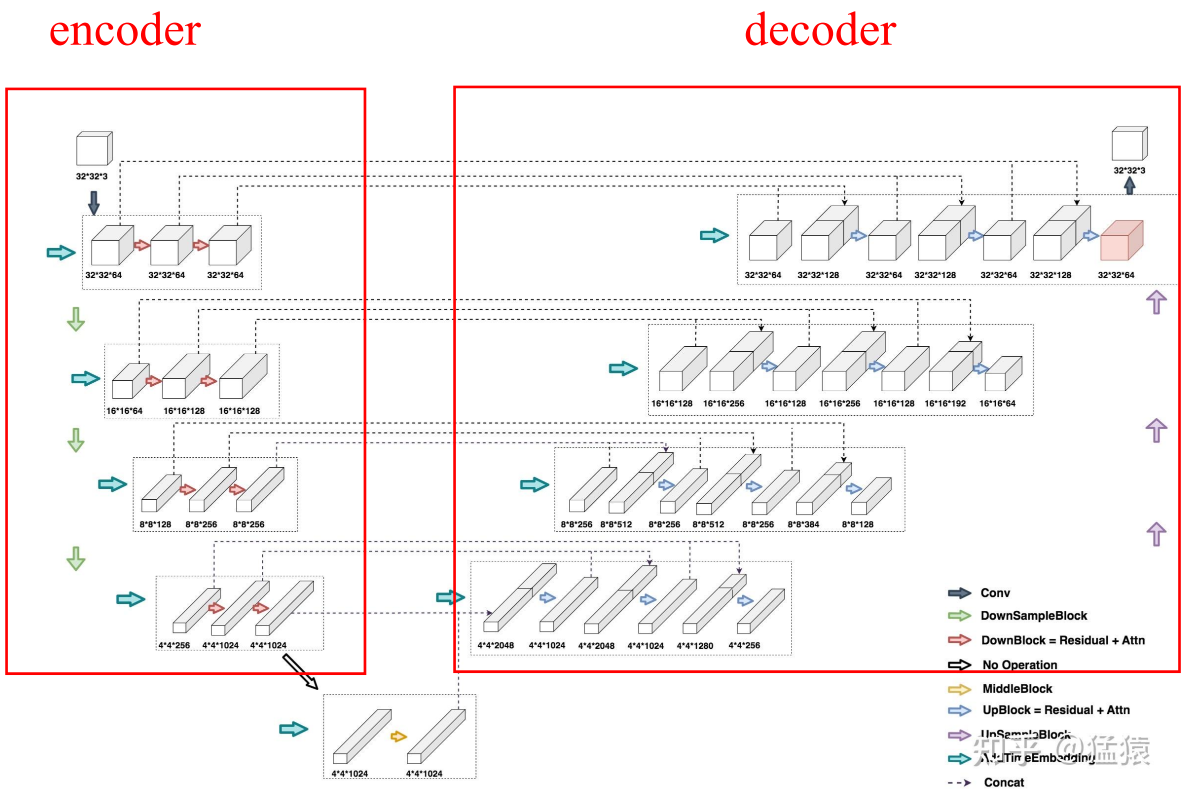
For the trained model, starting from T, we pass in a noise (or a picture with noise added) and gradually remove the noise. according to $x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon$, we can get the relationship between x_t and x_{t-1}.

The $\sigma_t \boldsymbol{z}$ is an additional term added to increase the randomness.

https://zhuanlan.zhihu.com/p/650394311

UNet


UNet
TimeStep


encoder
decoder
Conv
DownSampleBlock
DownBlock = Residual + Attn
No Operation
MiddleBlock
UpBlock = Residual + Attn
Concat

DDIM

Sampling process can be much faster with non-Markovian diffusion process.

Do not need to retrain the DDPM.

DDPM

T = 100					
x_t	100	99	...	2	1
x_{t-1}	99	98	...	1	0

DDIM

T = 100					
x_t	100	80	60	40	20
x_{t-1}	80	60	40	20	0

DDPM

```
for time_step in tqdm(reversed(range(self.T)), desc="Inference"):
    t = x_t.new_ones([x_T.shape[0], ], dtype=torch.long) * time_step
    mean, log_var = self.p_mean_variance(x_t, t)
    if time_step > 0:
        noise = torch.randn_like(x_t)
    else:
        noise = 0
    x_t = mean + torch.exp(0.5 * log_var) * noise
x_0 = x_t
```

DDIM

```
for i, j in tqdm(zip(reversed(list(t_seq)), reversed(list(t_prev_seq))), desc='Inference'):
    t = x_t.new_ones([x_T.shape[0], ], dtype=torch.long) * i
    prev_t = x_t.new_ones([x_T.shape[0], ], dtype=torch.long) * j

    alpha_cumprod_t = extract(self.alphas_bar_prev_whole, t, x_t.shape)
    alpha_cumprod_t_prev = extract(self.alphas_bar_prev_whole, prev_t, x_t.shape)

    # 根据训练好的ddpm算eps
    eps = self.model(x_t, t - 1) # 采用t-1是因为原本的ddpm的0位置元素代表t=1时刻,差了一个1
    # 计算x_0,用于第一项
    x_0 = self.predict_xstart_from_eps(x_t, t - 1, eps)
    if self.clip_denoised:
        x_0 = torch.clamp(x_0, min=-1., max=1.) # 裁剪梯度

    # 计算sigma,用于第三项
    sigma_t = self.ddim_eta * torch.sqrt(
        (1 - alpha_cumprod_t_prev) / (1 - alpha_cumprod_t) * (1 - alpha_cumprod_t / alpha_cumprod_t_prev))

    # 用于第二项
    pred_dir_xt = torch.sqrt(1 - alpha_cumprod_t_prev - sigma_t ** 2) * eps

    x_prev = torch.sqrt(alpha_cumprod_t_prev) * x_0 + pred_dir_xt + sigma_t ** 2 * torch.randn_like(x_t)
    x_t = x_prev
```

https://www.zhangzhenhu.com/aigc/ddim.html
https://zhuanlan.zhihu.com/p/666552214

Outline

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Diffusion Model

- DDPM, DDIM
- OpenAI help push the diffusion model
- Latent Diffusion Models and Latent Consistency Model
- How can we use it?
- Newest applications of the diffusion model

Diffusion Beats GAN

Diffusion Models Beat GANs on Image Synthesis

Improve the UNet

Channels	Depth	Heads	Attention resolutions	BigGAN up/downsample	Rescale resblock	FID 700K	FID 1200K
160	2	1	16	✗	✗	15.33	13.21
128	4					-0.21	-0.48
		4				-0.54	-0.82
			32,16,8			-0.72	-0.66
				✓		-1.20	-1.21
					✓	0.16	0.25
160	2	4	32,16,8	✓	✗	**-3.14**	**-3.00**

Number of heads	Channels per head	FID
1		14.08
2		-0.50
4		-0.97
8		-1.17
	32	-1.36
	64	-1.03
	128	-1.08

- Increasing depth versus width, holding model size relatively constant.
- Increasing the number of attention heads.
- Using attention at 32 x32, 16x 16, and 8x8 resolutions rather than only at 16x 16.
- Using the BigGAN residual block for upsampling and downsampling the activations.
- Rescaling residual connections with $1/\sqrt{2}$

https://sunlin-ai.github.io/2022/05/30/guided-diffusion.html#fn:1
https://arxiv.org/pdf/2105.05233.pdf

Diffusion Beats GAN

Diffusion Models Beat GANs on Image Synthesis

Classifier guided

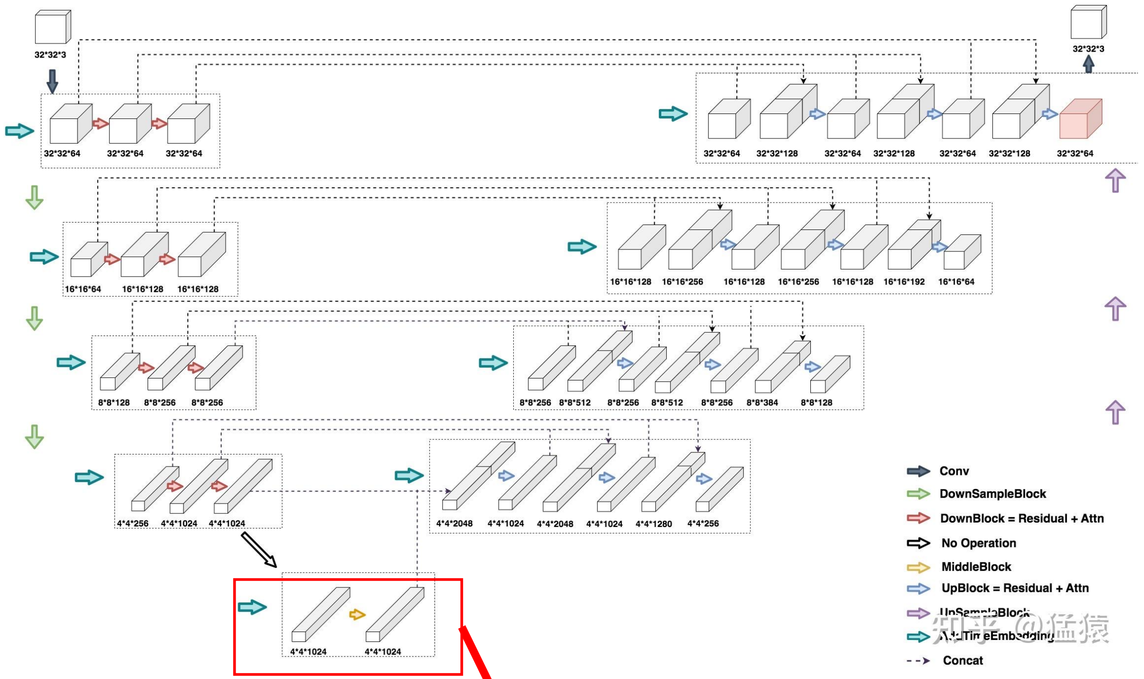
Modify after training the diffusion model

Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma\,\nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

Algorithm 2 Classifier guided DDIM sampling, given a diffusion model $\epsilon_\theta(x_t)$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(x_t) - \sqrt{1-\bar{\alpha}_t}\,\nabla_{x_t} \log p_\phi(y|x_t)$
  $x_{t-1} \leftarrow \sqrt{\bar{\alpha}_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}}\hat{\epsilon}$
end for
return x_0




```python
def cond_fn(x, t, y=None):
    assert y is not None
    with th.enable_grad():
        x_in = x.detach().requires_grad_(True)
        logits = classifier(x_in, t)
        log_probs = F.log_softmax(logits, dim=-1)
        selected = log_probs[range(len(logits)), y.view(-1)]
        return th.autograd.grad(selected.sum(), x_in)[0] * args.classifier_scale
```

https://sunlin-ai.github.io/2022/05/30/guided-diffusion.html#fn:1
https://arxiv.org/pdf/2105.05233.pdf
https://space.bilibili.com/13355688/?spm_id_from=333.999.0.0

GLIDE

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

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

Classifier-free guided

Modify during training the diffusion

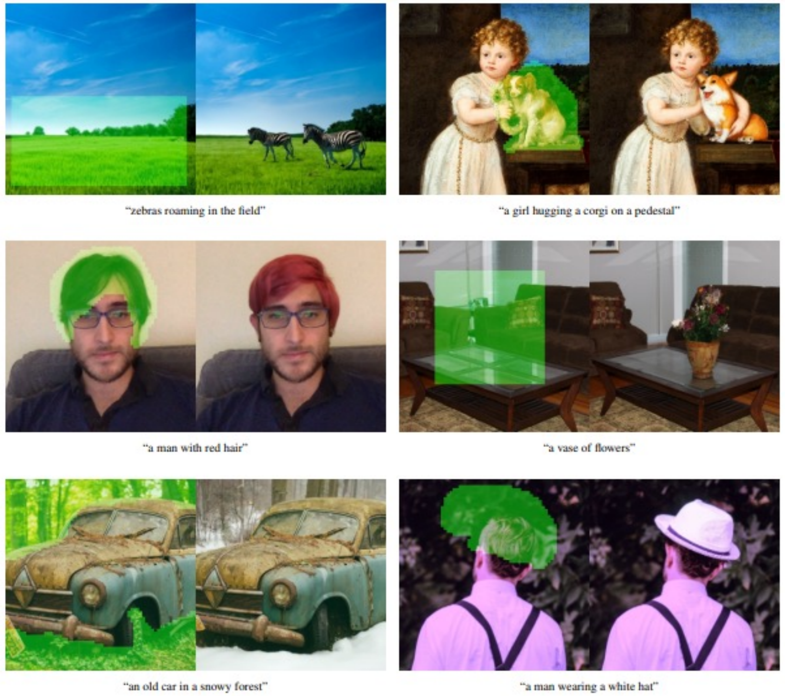


Figure 2. Text-conditional image inpainting examples from GLIDE. The green region is erased, and the model fills it in conditioned on the given prompt. Our model is able to match the style and lighting of the surrounding context to produce a realistic completion.

https://arxiv.org/abs/2112.10741
https://www.zhihu.com/question/507688429/answer/3111462670
link3

the label y in a class-conditional diffusion model $\epsilon_\theta(x_t|y)$ is replaced with a null label $\emptyset$ with a fixed probability during training.

During sampling, the output of the model is extrapolated further in the direction of $\epsilon_\theta(x_t|y)$ and away from $\epsilon_\theta(x_t|\emptyset)$ as follows:

$$\hat{\epsilon}_\theta(x_t|y) = \epsilon_\theta(x_t|\emptyset) + s \cdot (\epsilon_\theta(x_t|y) - \epsilon_\theta(x_t|\emptyset))$$

To implement generic text prompts, they sometimes replace text captions with an empty sequence $\emptyset$ during training.

Then guide towards the caption c using the modified prediction:

$$\hat{\epsilon}_\theta(x_t|c) = \epsilon_\theta(x_t|\emptyset) + s \cdot (\epsilon_\theta(x_t|c) - \epsilon_\theta(x_t|\emptyset))$$

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

CLIP guided

Modify during training the diffusion

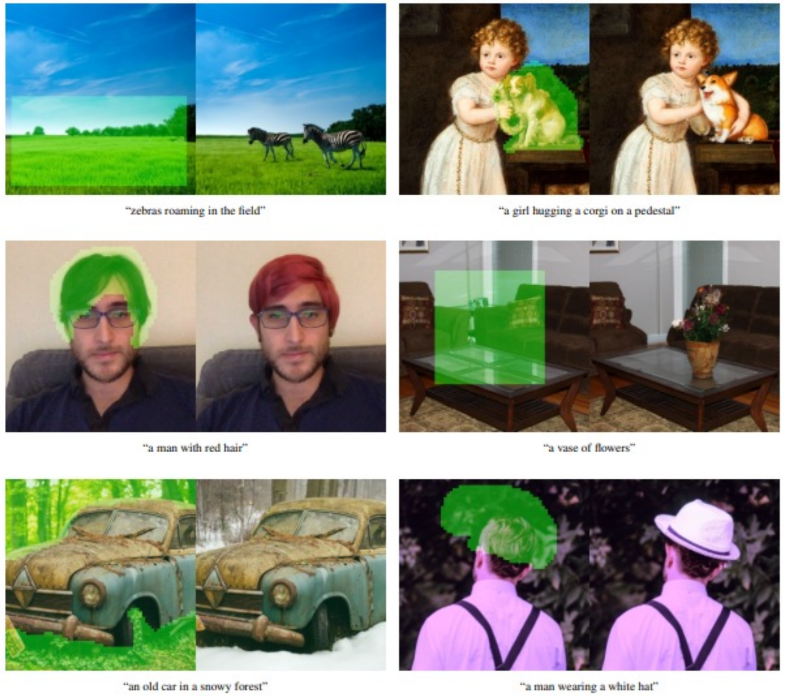


Figure 2. Text-conditional image inpainting examples from GLIDE. The green region is erased, and the model fills it in conditioned on the given prompt. Our model is able to match the style and lighting of the surrounding context to produce a realistic completion.

CLIP model consists of two separate pieces: an image encoder $f(x)$ and a caption encoder $g(c)$. During training, batches of $f(x) \cdot g(c)$pairs are sampled from a large dataset.

$$\hat{\mu}_\theta(x_t|c) = \mu_\theta(x_t|c) + s \cdot \Sigma_\theta(x_t|c)\nabla_{x_t}(f(x_t) \cdot g(c))$$

https://arxiv.org/abs/2112.10741
https://www.zhihu.com/question/507688429/answer/3111462670
link3

Classifier VS Classifier-free

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

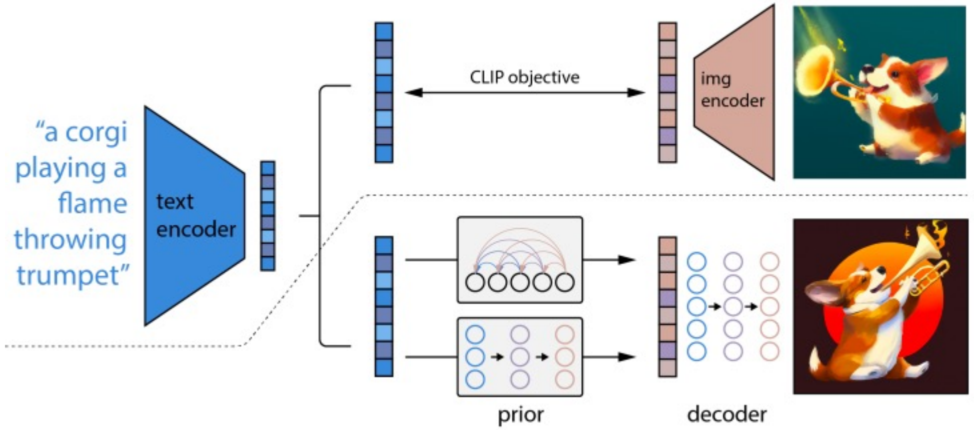
	Classifier guidance	Classifier free guidance
Retrain the model?	no	Yes
Train other model?	Yes, train a classifier model	No, directly use Clip
Results	Only can control the category in the classifier model	Any conditions.

```python
classifier_model = ...  # load a classifier model
y = 1  # generate class=1 picture
guidance_scale = 7.5  # control the classifier guide
input = get_noise(...)  # sample a noise

for t in tqdm(scheduler.timesteps):

    # unet get noise
    with torch.no_grad():
        noise_pred = unet(input, t).sample

    # get x_t-1
    input = scheduler.step(noise_pred, t, latents).prev_sample

    # classifier guidance
    class_guidance = classifier_model.get_class_guidance(input, y)
    input += class_guidance * guidance_scals  # add gradient
```

```python
clip_model = ...  # load clip model
text = "a dog"  # text
text_embeddings = clip_model.text_encode(text)  # encode text
empty_embeddings = clip_model.text_encode("")  # encode null
text_embeddings = torch.cat(empty_embeddings, text_embeddings)  # concat

input = get_noise(...)  # get noise

for t in tqdm(scheduler.timesteps):

    # unet predict noise
    with torch.no_grad():
        # predict noise including text and null
        noise_pred = unet(input, t, encoder_hidden_states=text_embeddings).sample

    # Classifier-Free Guidance
    noise_pred_uncond, noise_pred_text = noise_pred.chunk(2)  # split unconditioned noise and conditioned noise
    noise_pred = noise_pred_uncond + guidance_scale * (noise_pred_text - noise_pred_uncond)

    # get x_t-1
    input = scheduler.step(noise_pred, t, input).prev_sample
```

DALL·E 2

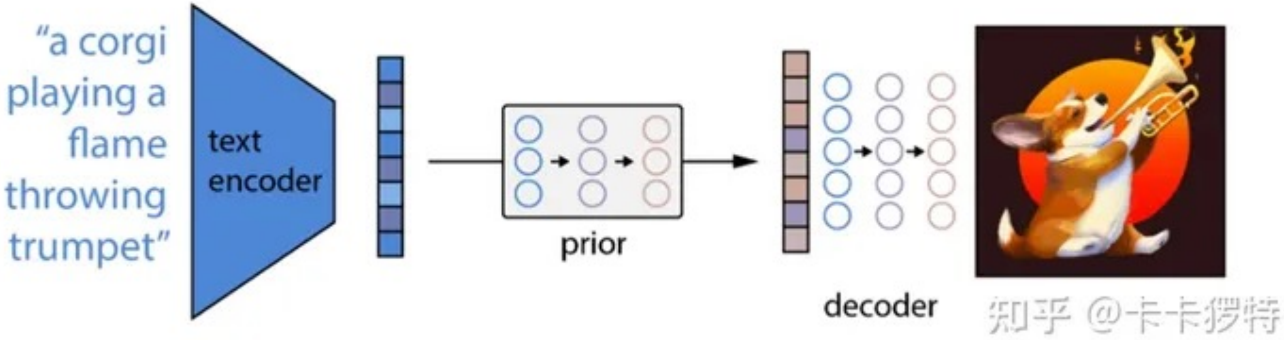
Hierarchical Text-Conditional Image Generation with CLIP Latents



Step 1: train the **CLIP model** to get text encoder and image encoder.

Step 2: train the **prior**, try to make text representation to image representation(diffusion model).

Step 3: train the **decoder,** rebuild the picture from text representation.



https://arxiv.org/pdf/2204.06125.pdf
https://zhuanlan.zhihu.com/p/526438544

Outline

Diffusion Model

- DDPM, DDIM
- OpenAI help push the diffusion model
- Latent Diffusion Models and Latent Consistency Model
- How can we use it?
- Newest applications of the diffusion model

Latent Diffusion Models

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

High-Resolution Image Synthesis with Latent Diffusion Models

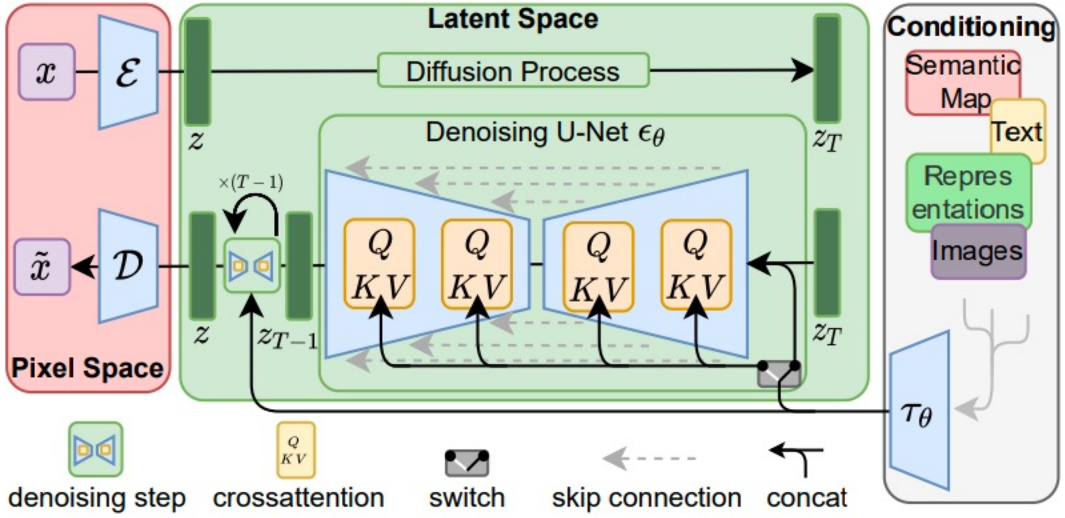


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3

To lower the computational demands of training diffusion models towards high-resolution image synthesis, we observe that although diffusion models allow to ignore perceptually irrelevant details by undersampling the corresponding loss terms.

https://arxiv.org/pdf/2204.06125.pdf
https://zhuanlan.zhihu.com/p/526438544

Latent Consistency Model

Consistency Model

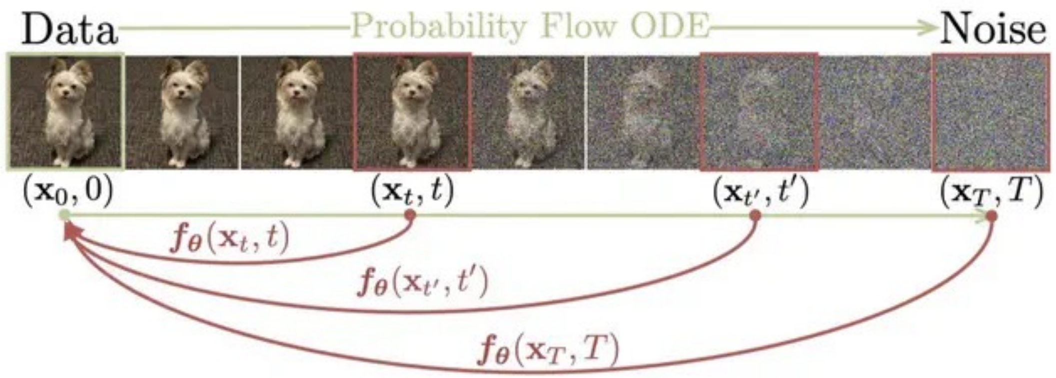


Figure 1: Given a Probability Flow (PF) ODE that smoothly converts data to noise, we learn to map any point (*e.g.*, $\mathbf{x}_t$, $\mathbf{x}_{t'}$, and $\mathbf{x}_T$) on the ODE trajectory to its origin (*e.g.*, $\mathbf{x}_0$) for generative modeling. Models of these mappings are called consistency models, as their outputs are trained to be consistent for points on the same trajectory.

https://wrong.wang/blog/20231111-consistency-is-all-you-need/

Consistency Model adds a new constraint:

every point on the noisy trajectory from a certain sample to a certain noise can be mapped to the starting point of this trajectory through a function f. Obviously, the points on the same trajectory will be the same point after f mapping. This is also the loss constraint used when training the Consistency Model.

CM defines a consistency function $f: (x_t, t) \rightarrow x_\epsilon$ (x_ϵ is the sampled result), there are two features:

$$f(x_\epsilon, \epsilon) = x_\epsilon$$
$$f(x_{t_1}, t_1) = f(x_{t_2}, t_2)$$

Latent Consistency Model

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Consistency Model

$$f(x_\epsilon, \epsilon) = x_\epsilon$$
$$f(x_{t_1}, t_1) = f(x_{t_2}, t_2)$$

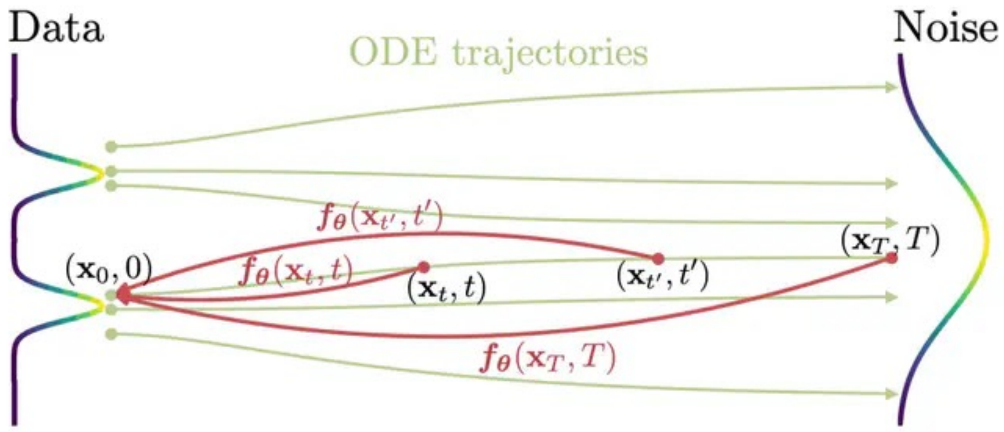


Figure 2: Consistency models are trained to map points on any trajectory of the PF ODE to the trajectory's origin.

Consistency Model is to find a f_θ to fit f.

Step 1: according to [2], define f_θ:

$$f_\theta(x, t) = c_{skip}(t)x + c_{out}(t)F_\theta(x, t)$$

$$c_{skip}(t) = \frac{\sigma_{data}^2}{(t-\epsilon)^2 + \sigma_{data}^2}, \; c_{out}(t) = \frac{\sigma_{data}(t-\epsilon)}{\sqrt{t^2 + \sigma_{data}^2}}$$

Step 2: Add consistency distillation loss

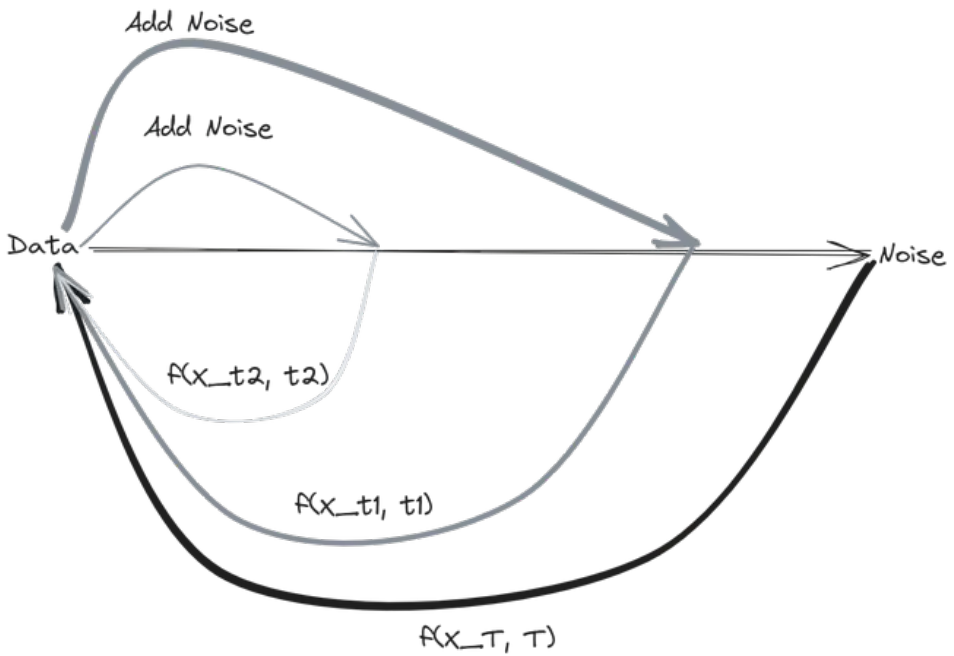
$$\mathcal{L}_{CD}^{N}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi}) := \mathbb{E}[\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}}, t_n))],$$

[1] https://wrong.wang/blog/20231111-consistency-is-all-you-need/
[2] Elucidating the Design Space of Diffusion-Based Generative Models

Latent Consistency Model

Consistency Model

$$f(x_\epsilon, \epsilon) = x_\epsilon$$
$$f(x_{t_1}, t_1) = f(x_{t_2}, t_2)$$



https://wrong.wang/blog/20231111-consistency-is-all-you-need/

Then it can do the sampling:

Algorithm 1 Multistep Consistency Sampling

Input: Consistency model $\boldsymbol{f_\theta}(\cdot, \cdot)$, sequence of time points $\tau_1 > \tau_2 > \cdots > \tau_{N-1}$, initial noise $\hat{\mathbf{x}}_T$

$\mathbf{x} \leftarrow \boldsymbol{f_\theta}(\hat{\mathbf{x}}_T, T)$

for $n = 1$ **to** $N - 1$ **do**

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

  $\hat{\mathbf{x}}_{\tau_n} \leftarrow \mathbf{x} + \sqrt{\tau_n^2 - \epsilon^2}\mathbf{z}$

  $\mathbf{x} \leftarrow \boldsymbol{f_\theta}(\hat{\mathbf{x}}_{\tau_n}, \tau_n)$

end for

Output: $\mathbf{x}$

Latent Consistency Model

Algorithm 1 Latent Consistency Distillation (LCD)

Input: dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate η, ODE solver $\Psi(\cdot,\cdot,\cdot,\cdot)$, distance metric $d(\cdot,\cdot)$, EMA rate μ, noise schedule $\alpha(t), \sigma(t)$, guidance scale $[w_{\min}, w_{\max}]$, skipping interval k, and encoder $E(\cdot)$

Encoding training data into latent space: $\mathcal{D}_z = \{(\boldsymbol{z}, \boldsymbol{c}) | \boldsymbol{z} = E(\boldsymbol{x}), (\boldsymbol{x}, \boldsymbol{c}) \in \mathcal{D}\}$

$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$

repeat

  Sample $(\boldsymbol{z}, \boldsymbol{c}) \sim \mathcal{D}_z$, $n \sim \mathcal{U}[1, N-k]$ and $\omega \sim [\omega_{\min}, \omega_{\max}]$

  Sample $\boldsymbol{z}_{t_{n+k}} \sim \mathcal{N}(\alpha(t_{n+k})\boldsymbol{z}; \sigma^2(t_{n+k})\mathbf{I})$

  $\hat{\boldsymbol{z}}_{t_n}^{\Psi,\omega} \leftarrow \boldsymbol{z}_{t_{n+k}} + (1+\omega)\Psi(\boldsymbol{z}_{t_{n+k}}, t_{n+k}, t_n, \boldsymbol{c}) - \omega\Psi(\boldsymbol{z}_{t_{n+k}}, t_{n+k}, t_n, \varnothing)$

  $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \Psi) \leftarrow d(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_{t_{n+k}}, \omega, \boldsymbol{c}, t_{n+k}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\boldsymbol{z}}_{t_n}^{\Psi,\omega}, \omega, \boldsymbol{c}, t_n))$

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)$

  $\boldsymbol{\theta}^- \leftarrow \text{stopgrad}(\mu\boldsymbol{\theta}^- + (1-\mu)\boldsymbol{\theta})$

until convergence

Latent VAE $E(\cdot)$

CFG Guidance Scale $[w_{min}, w_{max}]$

Diffusion skip number k

https://wrong.wang/blog/20231111-consistency-is-all-you-need/

Latent Consistency Model

Latent Consistency Model

Algorithm 1 Latent Consistency Distillation (LCD)

Input: dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate η, ODE solver $\Psi(\cdot,\cdot,\cdot,\cdot)$, distance metric $d(\cdot,\cdot)$, EMA rate μ, noise schedule $\alpha(t), \sigma(t)$, guidance scale $[w_{\min}, w_{\max}]$, skipping interval k, and encoder $E(\cdot)$
Encoding training data into latent space: $\mathcal{D}_z = \{(\boldsymbol{z}, \boldsymbol{c}) | \boldsymbol{z} = E(\boldsymbol{x}), (\boldsymbol{x}, \boldsymbol{c}) \in \mathcal{D}\}$
$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
repeat
  Sample $(\boldsymbol{z}, \boldsymbol{c}) \sim \mathcal{D}_z$, $n \sim \mathcal{U}[1, N-k]$ and $\omega \sim [\omega_{\min}, \omega_{\max}]$
  Sample $\boldsymbol{z}_{t_{n+k}} \sim \mathcal{N}(\alpha(t_{n+k})\boldsymbol{z}; \sigma^2(t_{n+k})\mathbf{I})$
  $\hat{\boldsymbol{z}}_{t_n}^{\Psi,\omega} \leftarrow \boldsymbol{z}_{t_{n+k}} + (1+\omega)\Psi(\boldsymbol{z}_{t_{n+k}}, t_{n+k}, t_n, \boldsymbol{c}) - \omega\Psi(\boldsymbol{z}_{t_{n+k}}, t_{n+k}, t_n, \varnothing)$
  $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \Psi) \leftarrow d(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_{t_{n+k}}, \omega, \boldsymbol{c}, t_{n+k}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\boldsymbol{z}}_{t_n}^{\Psi,\omega}, \omega, \boldsymbol{c}, t_n))$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)$
  $\boldsymbol{\theta}^- \leftarrow \text{stopgrad}(\mu\boldsymbol{\theta}^- + (1-\mu)\boldsymbol{\theta})$
until convergence

https://wrong.wang/blog/20231111-consistency-is-all-you-need/

Step 1: sample dataset (z, c) (the picture latent and text), choose n as the diffusion timestamp, and w as Guidance Scale.

Step 2: Do the diffusion, get z_{n+k}

Step 3: Do the denoise with DDIM diffusion scheduler or DPM-Solver.

Step 4: Compute the consistency loss based on $z_{n+k}, \hat{z}_n$.

Step 5: Update model.

Step 6: Do the EMA.

Latent Consistency Model

2-Steps Inference 1-Step Inference

https://wrong.wang/blog/20231111-consistency-is-all-you-need/

Latent Consistency Model

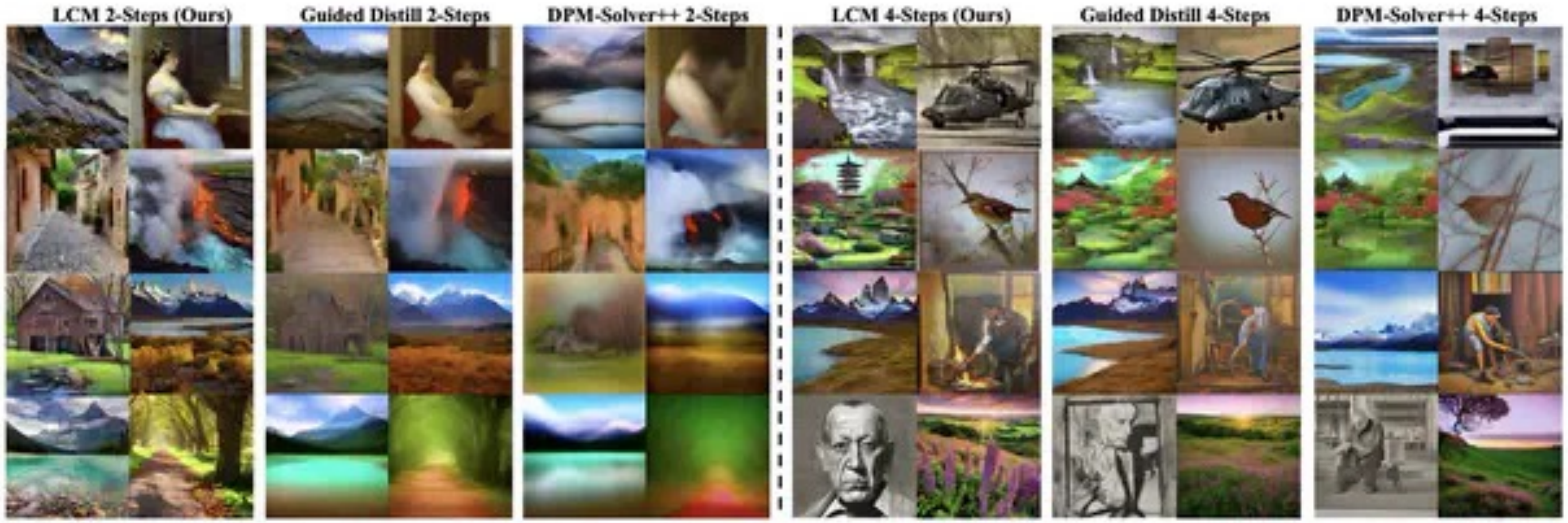


Figure 2: Text-to-Image generation results on LAION-Aesthetic-6.5+ with 2-, 4-step inference. Images generated by LCM exhibit superior detail and quality, outperforming other baselines by a large margin.

https://wrong.wang/blog/20231111-consistency-is-all-you-need/

Outline

Diffusion Model

- DDPM, DDIM
- OpenAI help push the diffusion model, GLIDE, DALLE2
- Latent Diffusion Models and Latent Consistency Model
- How can we use it?
- Newest applications of the diffusion model

How can we use

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Diffusers

Diffusers

https://huggingface.co/docs/diffusers/index

For using DDPM, you can:

```
>>> from diffusers import DDPMPipeline

>>> ddpm = DDPMPipeline.from_pretrained("google/ddpm-cat-256", use_safetensors=True).to("cuda")
>>> image = ddpm(num_inference_steps=25).images[0]
>>> image
```

How can we use

Diffusers

Diffusers

https://huggingface.co/docs/diffusers/index

For using LDM, you can:

```python
from diffusers import AutoPipelineForText2Image
import torch

pipeline = AutoPipelineForText2Image.from_pretrained(
    "runwayml/stable-diffusion-v1-5", torch_dtype=torch.float16, use_safetensors=True
).to("cuda")
prompt = "peasant and dragon combat, wood cutting style, viking era, bevel with rune"

image = pipeline(prompt, num_inference_steps=25).images[0]
image
```

How can we train

Diffusers

Diffusers

For training diffusion model, you can follow the step:

STEP 1: Build your own dataset.

https://huggingface.co/docs/datasets/v2.4.0/en/image_load#imagefolder-with-metadata

For DDPM, you can just need some pictures.

	edan_url string	source string	stage float64	image image	image_hash string	sim_score float64
et/ark:/65665/35f90bc1d-3c-611d7b358636	edanmdm:nmnheducation_11038234	Smithsonian Education…	null		fb0b8749d437efc70a26e54212b3572c	0.80552
et/ark:/65665/3f4d0cc10-54-1371681e92b4	edanmdm:nmnheducation_11038220	Smithsonian Education…	null		9657726e69494021d1c9929ee7b375fa	0.810842
et/ark:/65665/33446acdf-5a-d789385acdcc	edanmdm:nmnheducation_11038238	Smithsonian Education…	null		9303ab0ac75fd0d3047ba987d268f871	0.813563
et/ark:/65665/359f1f2d7-e8-034a14006aa6	edanmdm:nmnheducation_11038217	Smithsonian Education…	null		3726a5faf63c3d70db5d433705b53ba9	0.813736
et/ark:/65665/3f50ca15b-c9-cfe993468eca	edanmdm:nmnheducation_11038213	Smithsonian Education…	null		3aa4d93629910b3fe1165c4fc20033fc	0.81446

For text-to-image, you have to make a text-image pair.

image image	text string · lengths 24 97
	a drawing of a green pokemon with red eyes
	a green and yellow toy with a red nose
	a red and white ball with an angry look on its face
	a cartoon ball with a smile on it's face

How can we train

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

🤗 D🧨ffusers

Image Generation STEP 2: make it into train dataloader.
https://huggingface.co/docs/diffusers/tutorials/basic_training

```python
config = TrainingConfig()
config.dataset_name = "huggan/smithsonian_butterflies_subset"
dataset = load_dataset(config.dataset_name, split="train")
```

✓ 1m 40.4s

```
Using the latest cached version of the dataset since huggan/smithsonian_butterflies_subset couldn't be found on the Hugging Face Hub
Found the latest cached dataset configuration 'default' at /home/chenweilong/.cache/huggingface/datasets/huggan___smithsonian_butterflies_subset/default/0.0.0/3cdedf844922ab403
```

```python
import torch

train_dataloader = torch.utils.data.DataLoader(dataset, batch_size=config.train_batch_size, shuffle=True)
```

How can we train

Diffusers

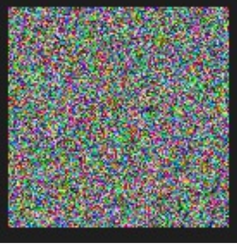
Image Generation STEP 3 – Model and Scheduler Build

```python
from diffusers import UNet2DModel

model = UNet2DModel(
    sample_size=config.image_size,  # the target image resolution
    in_channels=3,  # the number of input channels, 3 for RGB images
    out_channels=3,  # the number of output channels
    layers_per_block=2,  # how many ResNet layers to use per UNet block
    block_out_channels=(128, 128, 256, 256, 512, 512),  # the number of output channels for each UNet block
    down_block_types=(
        "DownBlock2D",  # a regular ResNet downsampling block
        "DownBlock2D",
        "DownBlock2D",
        "DownBlock2D",
        "AttnDownBlock2D",  # a ResNet downsampling block with spatial self-attention
        "DownBlock2D",
    ),
    up_block_types=(
        "UpBlock2D",  # a regular ResNet upsampling block
        "AttnUpBlock2D",  # a ResNet upsampling block with spatial self-attention
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
    ),
)
```

Define the UNet2DModel

```python
from diffusers import DDPMScheduler
sample_image = dataset[0]["images"].unsqueeze(0)
noise_scheduler = DDPMScheduler(num_train_timesteps=1000)
noise = torch.randn(sample_image.shape)
timesteps = torch.LongTensor([999])
noisy_image = noise_scheduler.add_noise(sample_image, noise, timesteps)
```

✓ 0.0s



Define the DDPMScheduler

How can we train

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Diffusers

Image Generation STEP 4 – Define the training

```
for step, batch in enumerate(train_dataloader):
    clean_images = batch["images"]
    # Sample noise to add to the images
    noise = torch.randn(clean_images.shape, device=clean_images.device)
    bs = clean_images.shape[0]

    # Sample a random timestep for each image
    timesteps = torch.randint(
        0, noise_scheduler.config.num_train_timesteps, (bs,), device=clean_images.device,
        dtype=torch.int64
    )

    # 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)

    with accelerator.accumulate(model):
        # Predict the noise residual
        noise_pred = model(noisy_images, timesteps, return_dict=False)[0]
        loss = F.mse_loss(noise_pred, noise)
        accelerator.backward(loss)
```

Random some timestep

Add some noise

Model predict

Get the loss

How can we train

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Diffusers

Image Generation STEP 5 – Define the evaluate

```
from diffusers import DDPMPipeline

# load model and scheduler
pipe = DDPMPipeline.from_pretrained("/home/chenweilong/diffusion_model_learn/ddpm-butterflies-128")

# run pipeline in inference (sample random noise and denoise)
image = pipe().images[0]

image
```

✓ 4m 17.5s

Loading pipeline components...: 100% 2/2 [00:00<00:00, 3.58it/s]

100% 1000/1000 [04:16<00:00, 3.94it/s]

Load model

Get images

How can we train

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Diffusers

Text-to-Image STEP 2: choose a base model and load model
https://huggingface.co/models

```
noise_scheduler = DDPMScheduler.from_pretrained(args.pretrained_model_name_or_path, subfolder="scheduler")
tokenizer = CLIPTokenizer.from_pretrained(
    args.pretrained_model_name_or_path, subfolder="tokenizer", revision=args.revision
)
text_encoder = CLIPTextModel.from_pretrained(
    args.pretrained_model_name_or_path, subfolder="text_encoder", revision=args.revision
)
vae = AutoencoderKL.from_pretrained(
    args.pretrained_model_name_or_path, subfolder="vae", revision=args.revision, variant=args.variant
)
unet = UNet2DConditionModel.from_pretrained(
    args.pretrained_model_name_or_path, subfolder="unet", revision=args.revision, variant=args.variant
)
# freeze parameters of models to save more memory
unet.requires_grad_(False)
vae.requires_grad_(False)
text_encoder.requires_grad_(False)
```

Load noise scheduler

Load text encoder

Load VAE

Load UNet

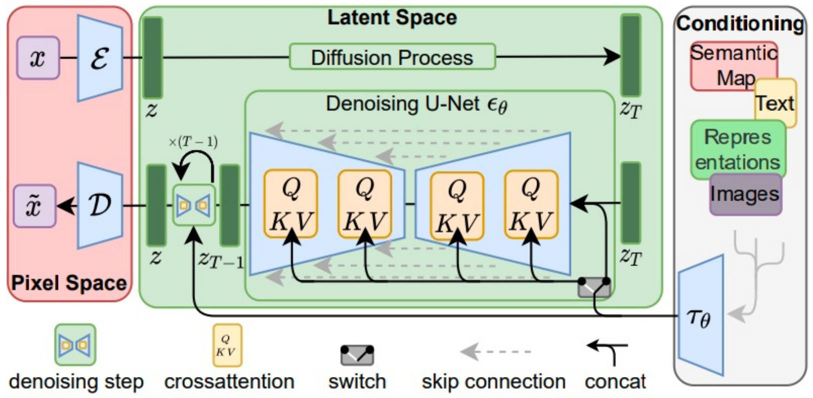
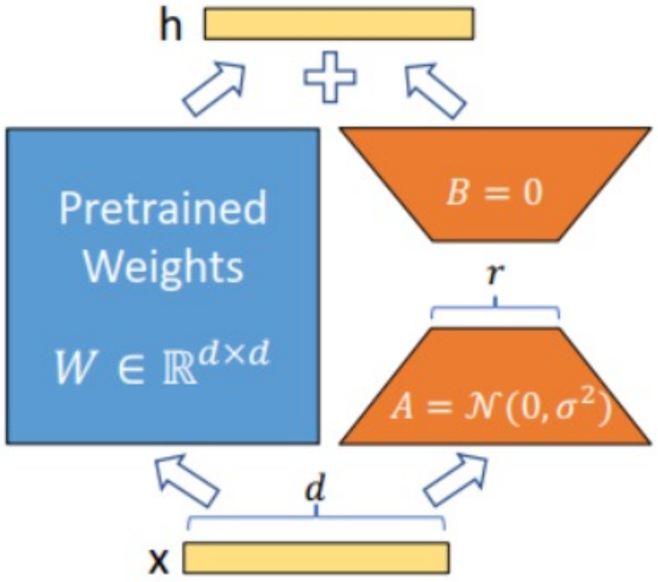


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3

How can we train

🤗 D🧨ffusers

Text-to-Image STEP 3: choose a way to train: LoRA training
https://github.com/huggingface/diffusers/tree/main/examples/text_to_image



$W_0 \in \mathbb{R}^{d\times k}$

$W_0 + \Delta W = W_0 + BA \qquad B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k} \quad and \quad r \ll min(d, k)$

```
unet_lora_config = LoraConfig(
    r=args.rank,
    lora_alpha=args.rank,
    init_lora_weights="gaussian",
    target_modules=["to_k", "to_q", "to_v", "to_out.0"],
)
# Add adapter and make sure the trainable params are in float32.
unet.add_adapter(unet_lora_config)
```

How can we train

Diffusers

Text-to-Image STEP 4: encode the text

```
def tokenize_captions(examples, is_train=True):
    captions = []
    for caption in examples[caption_column]:
        if isinstance(caption, str):
            captions.append(caption)
        elif isinstance(caption, (list, np.ndarray)):
            # take a random caption if there are multiple
            captions.append(random.choice(caption) if is_train else caption[0])
        else:
            raise ValueError(
                f"Caption column `{caption_column}` should contain either strings or lists of strings."
            )
    inputs = tokenizer(
        captions, max_length=tokenizer.model_max_length, padding="max_length", truncation=True, return_tensors="pt"
    )
    return inputs.input_ids
```

Text tokenizer to make text to IDs.

How can we train

Diffusers

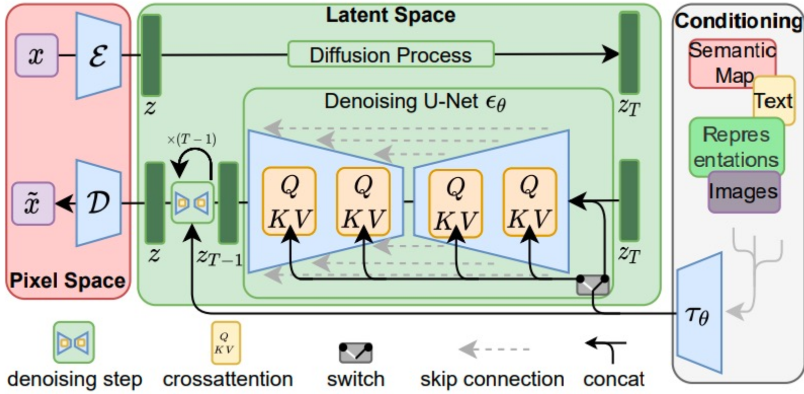


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3

Text-to-Image STEP 5: train the model

```python
for step, batch in enumerate(train_dataloader):
    with accelerator.accumulate(unet):
        # Convert images to latent space
        latents = vae.encode(batch["pixel_values"].to(dtype=weight_dtype)).latent_dist.sample()
        latents = latents * vae.config.scaling_factor
        noise = torch.randn_like(latents)
        bsz = latents.shape[0]
        # Sample a random timestep for each image
        timesteps = torch.randint(0, noise_scheduler.config.num_train_timesteps, (bsz,), device=latents.device)
        noisy_latents = noise_scheduler.add_noise(latents, noise, timesteps)
        encoder_hidden_states = text_encoder(batch["input_ids"])[0]
        model_pred = unet(noisy_latents, timesteps, encoder_hidden_states).sample
        loss = F.mse_loss(model_pred.float(), target.float(), reduction="mean")

        # Gather the losses across all processes for logging (if we use distributed training).
        avg_loss = accelerator.gather(loss.repeat(args.train_batch_size)).mean()
        train_loss += avg_loss.item() / args.gradient_accumulation_steps
```

get latent representation

Random timestamp

Add noise

Text representation

Model predict

How can we train

Diffusers

Text-to-Image STEP 6: evaluate

```
from diffusers import StableDiffusionPipeline
import torch

model_path = "/home/chenweilong/diffusion_model_learn/diffusers/examples/text_to_image/sd-pokemon-model-lora/"
pipe = StableDiffusionPipeline.from_pretrained("/home/chenweilong/diffusion_model_learn/sd1-4/", torch_dtype=torch.float16)
pipe.unet.load_attn_procs(model_path)
pipe.to("cuda")

prompt = "A pokemon with green eyes and red legs."
image = pipe(prompt, num_inference_steps=30, guidance_scale=7.5).images[0]
image
```

Load Unet and lora weights

Do the text-to-image

How can we use

Diffusers

https://huggingface.co/docs/diffusers/training/overview

You can find different training ways in the website.

Training	SDXL-support	LoRA-support	Flax-support
unconditional image generation Open in Colab			
text-to-image	👍	👍	👍
textual inversion Open in Colab			👍
DreamBooth Open in Colab	👍	👍	👍
ControlNet	👍		👍
InstructPix2Pix	👍		
Custom Diffusion			
T2I-Adapters	👍		
Kandinsky 2.2		👍	
Wuerstchen		👍	

Outline

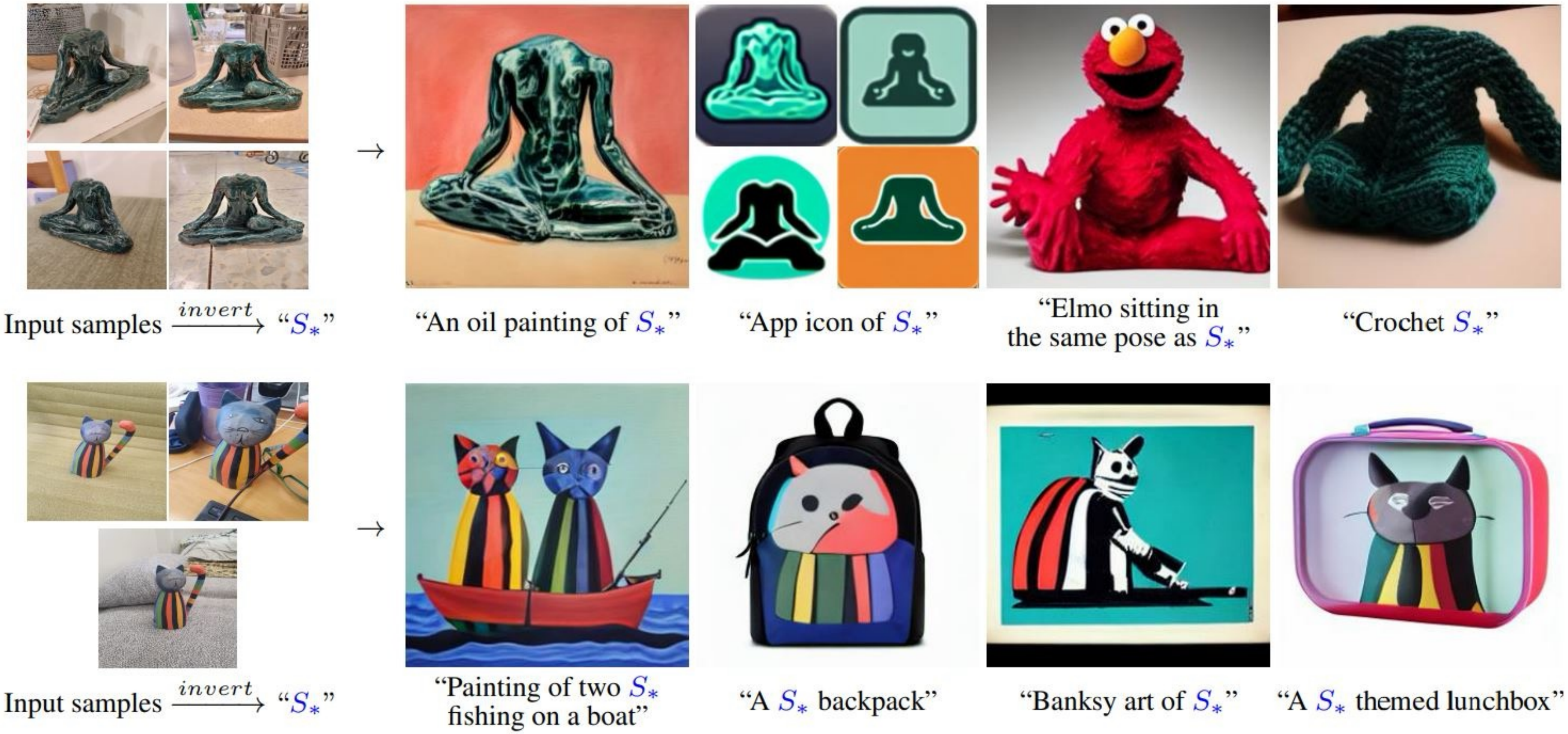
UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING
Department of Electrical & Computer Engineering

Diffusion Model

- DDPM, DDIM
- OpenAI help push the diffusion model, GLIDE, DALLE2
- Latent Diffusion Models and Latent Consistency Model
- How can we use it?
- Newest applications of the diffusion model

Newest applications

https://textual-inversion.github.io/



We learn to generate specific concepts, like personal objects or artistic styles, by describing them using new "words" in the embedding space of pre-trained text-to-image models. These can be used in new sentences, just like any other word.

Newest applications

https://dreambooth.github.io/

Given as input just a few images of a subject, we fine-tune a pretrained text-to-image model (Imagen, although our method is not limited to a specific model) such that it learns to bind a unique identifier with that specific subject.

Newest applications

https://github.com/microsoft/LoRA

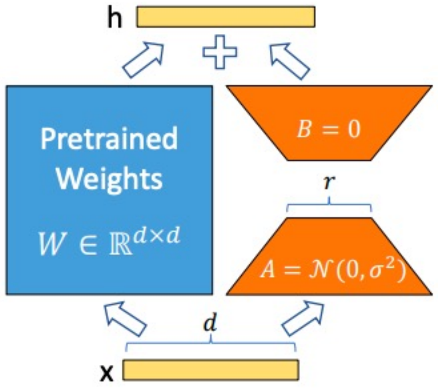


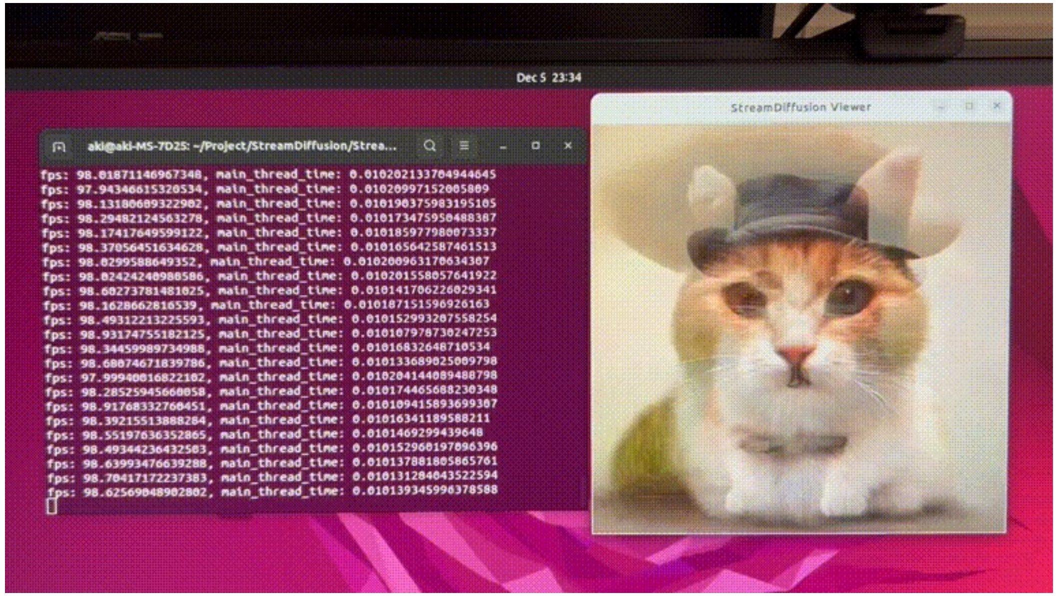
Figure 1: Our reparametrization. We only train A and B.

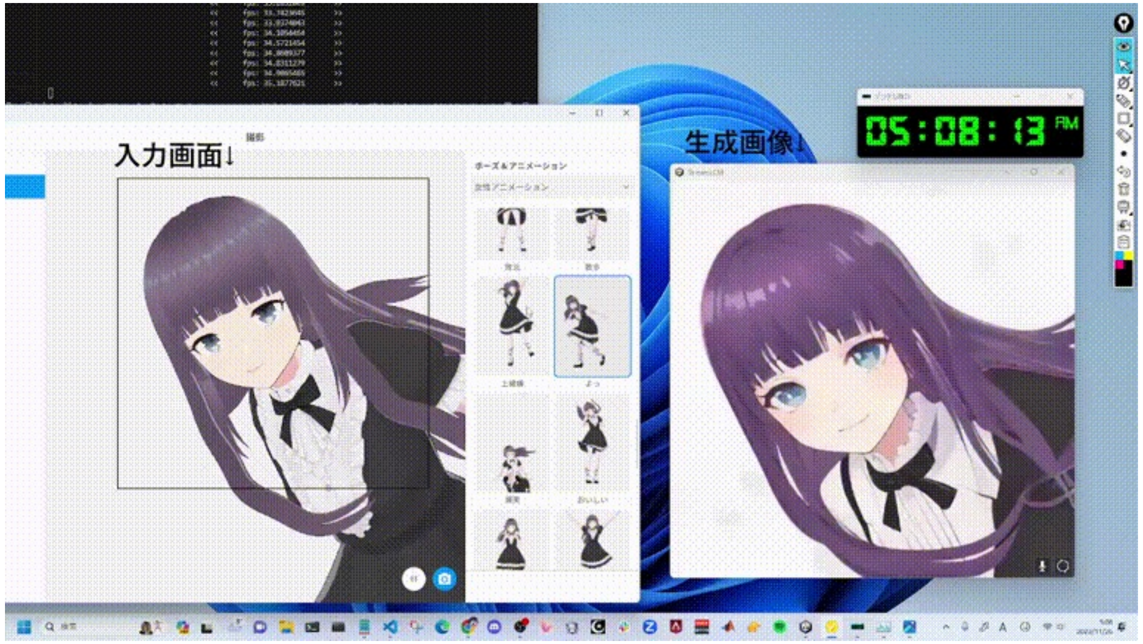
	RoBERTa base Fine-tune	RoBERTa base LoRA	DeBERTa XXL Fine-tune	DeBERTa XXL LoRA
# of Trainable Params.	125M	0.8M	1.5B	4.7M
MNLI (m-Acc/mm-Acc)	**87.6**	87.5±.3/86.9±.3	91.7/**91.9**	**91.9±.1**/91.9±.2
SST2 (Acc)	94.8	**95.1±.2**	**97.2**	96.9±.2
MRPC (Acc)	**90.2**	89.7±.7	92.0	**92.6±.6**
CoLA (Matthew's Corr)	**63.6**	63.4±1.2	**72.0**	72.4±1.1
QNLI (Acc)	92.8	**93.3±.3**	**96.0**	**96.0±.1**
QQP (Acc)	**91.9**	90.8±.1	92.7	**92.9±.1**
RTE (Acc)	78.7	**86.6±.7**	93.9	**94.9±.4**
STSB (Pearson/Spearman Corr)	91.2	**91.5±.2/91.3±.2**	**92.9**/92.6	**93.0±.2/92.9±.3**
Average	86.40	**87.24**	91.06	**91.32**

LoRA reduces the number of trainable parameters by learning pairs of rank-decompostion matrices while freezing the original weights. This vastly reduces the storage requirement for large language models adapted to specific tasks and enables efficient task-switching during deployment all without introducing inference latency.

Newest applications

https://github.com/cumulo-autumn/StreamDiffusion





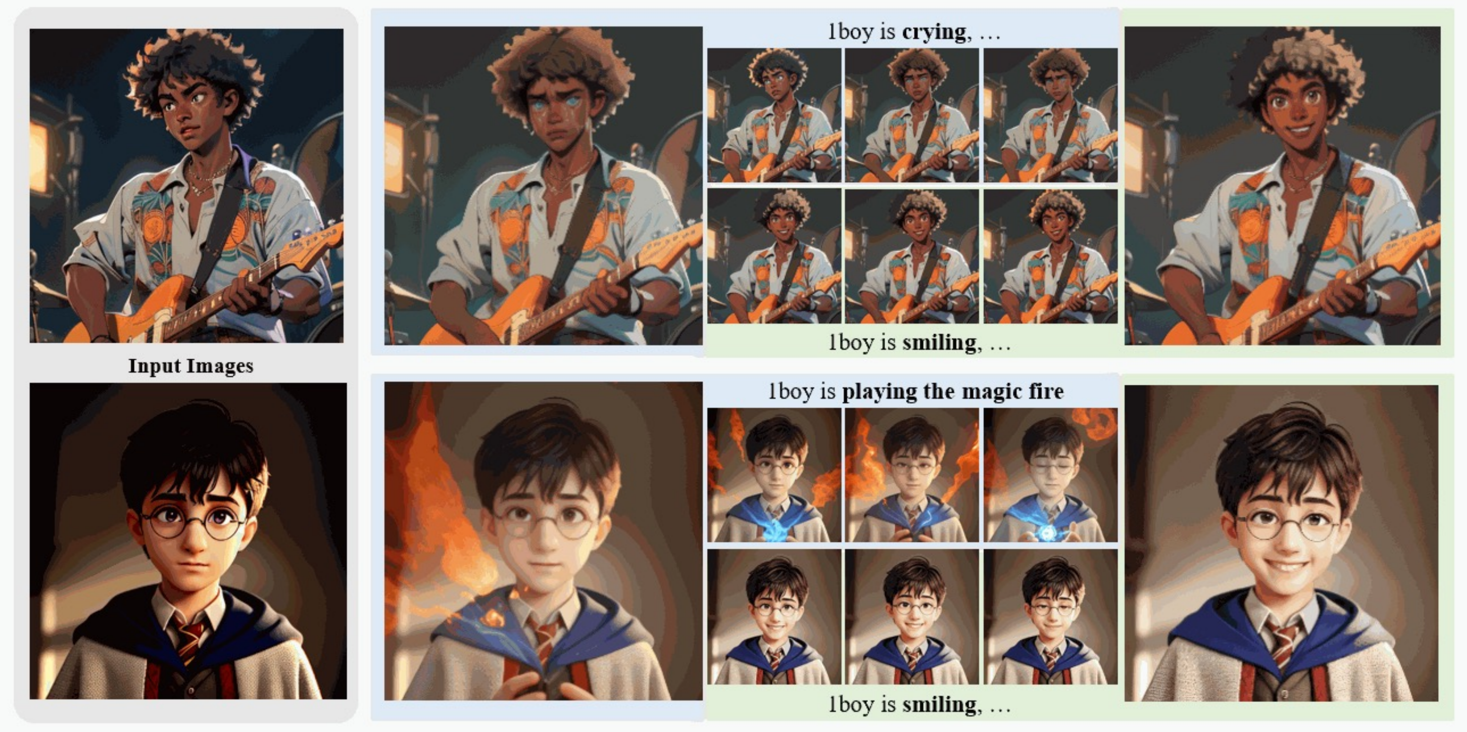
StreamDiffusion is an innovative diffusion pipeline designed for real-time interactive generation. It introduces significant performance enhancements to current diffusion-based image generation techniques.

Newest applications

https://github.com/ChenHsing/Awesome-Video-Diffusion-Models

Newest applications

https://github.com/open-mmlab/PIA



PIA is a personalized image animation method which can generate videos with **high motion controllability** and **strong text and image alignment**.

Newest applications

https://pika.art/my-library