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
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/
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.
Demonstrate that diffusion models are capable of generating high quality samples.
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
Diffusion process: adds Gaussian noise added for each step $q(x_t|x_{t-1})$

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

\[ q(x_t|x_{t-1}) \]

\[ x_t = x_{t-1} + \epsilon = x_0 + \epsilon_0 + \epsilon_1 + \ldots + \epsilon \]

\[ x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_t \]

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:

\[ x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon \]

\[ \alpha_t = 1 - \beta_t \]

\[ \bar{\alpha}_t = \alpha_1 \alpha_2 \ldots \alpha_t \]

\( \beta_t \) is a constant hyperparameter. As T increases, they become larger and larger.
DDPM

Denoise process: reverse the picture from noise

\[ q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t) \]
DDPM

Algorithm 1 Training
1: repeat
2: \( \mathbf{x}_0 \sim q(\mathbf{x}_0) \)
3: \( t \sim \text{Uniform}(\{1, \ldots, T\}) \)
4: \( \epsilon \sim \mathcal{N}(0, I) \)
5: Take gradient descent step on
\[
\nabla_{\theta} \left\| \epsilon - \epsilon_{\theta}(\sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \epsilon, t) \right\|^2
\]
6: until converged

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

\( x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon \)

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: \( \epsilon_{\theta}(\sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon, t) \)

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

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 \( x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon \), we can get the relationship between \( x_t \) and \( x_{t-1} \).

The \( \sigma_t \mathbf{z} \) is an additional term added to increase the randomness.

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

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

Do not need to retrain the DDPM.
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
Diffusion Models Beat GANs on Image Synthesis

Improve the UNet

- 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 $1/\sqrt{2}$

<table>
<thead>
<tr>
<th>Channels</th>
<th>Depth</th>
<th>Heads</th>
<th>Attention resolutions</th>
<th>BigGAN up/downsample</th>
<th>Rescale resblock</th>
<th>FID 700K</th>
<th>FID 1200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>2</td>
<td>1</td>
<td>16</td>
<td>x</td>
<td>x</td>
<td>15.33</td>
<td>13.21</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
<td>4</td>
<td>32,16,8</td>
<td>x</td>
<td>x</td>
<td>-0.21</td>
<td>-0.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32,16,8</td>
<td></td>
<td></td>
<td>-0.54</td>
<td>-0.82</td>
</tr>
<tr>
<td>160</td>
<td>2</td>
<td>4</td>
<td>32,16,8</td>
<td>x</td>
<td>x</td>
<td>-1.20</td>
<td>-1.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32,16,8</td>
<td>x</td>
<td></td>
<td>0.16</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td>-3.14</td>
<td>-3.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of heads</th>
<th>Channels per head</th>
<th>FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14.08</td>
</tr>
<tr>
<td>2</td>
<td>-0.50</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-0.97</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-1.17</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-1.36</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>-1.03</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>-1.08</td>
<td></td>
</tr>
</tbody>
</table>

https://sunlin-ai.github.io/2022/05/30/guided-diffusion.html#fn:1
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 \((\mu_0(x_t), \Sigma_0(x_t))\), classifier \(p_\theta(y|x_t)\), and gradient scale \(s\).

Input: class label \(y\), gradient scale \(s\)
\[ x_T \leftarrow \text{sample from } \mathcal{N}(0, I) \]
for all \(t\) from \(T\) to 1 do
\[ \mu, \Sigma \leftarrow \mu_0(x_t), \Sigma_0(x_t) \]
\[ x_{t-1} \leftarrow \text{sample from } \mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\theta(y|x_t), \Sigma) \]
end for
return \(x_0\)

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

Input: class label \(y\), gradient scale \(s\)
\[ x_T \leftarrow \text{sample from } \mathcal{N}(0, I) \]
for all \(t\) from \(T\) to 1 do
\[ \hat{\epsilon} \leftarrow \epsilon_\theta(x_t) - \sqrt{1 - \alpha_t} \nabla_{x_t} \log p_\theta(y|x_t) \]
\[ x_{t-1} \leftarrow \sqrt{\alpha_{t-1}} \left( \frac{x_t - \sqrt{1 - \alpha_t} \hat{\epsilon}}{\sqrt{\alpha_t}} \right) + \sqrt{1 - \alpha_{t-1}} \hat{\epsilon} \]
end for
return \(x_0\)

https://sunlin-ai.github.io/2022/05/30/guided-diffusion.html#fn:1
https://space.bilibili.com/13355688/?spm_id_from=333.999.0.0
Classifier-free guided
Modify during training the diffusion

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:

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

https://arxiv.org/abs/2112.10741
https://www.zhihu.com/question/507688429/answer/3111462670
link3
GLIDE: Towards Photorealistic Image Generation and Editing with Text–Guided Diffusion Models

CLIP guided

Modify during training the diffusion

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.

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

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
## Classifier VS Classifier-free

<table>
<thead>
<tr>
<th>Classifier guidance</th>
<th>Classifier free guidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrain the model?</td>
<td>no</td>
</tr>
<tr>
<td>Train other model?</td>
<td>Yes, train a classifier model</td>
</tr>
<tr>
<td>Results</td>
<td>Only can control the category in the classifier model</td>
</tr>
</tbody>
</table>

```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_scale # add gradient
```

```python
clip_model = ... # load clip model
text = "a dog" # text
text_embeddings = clip_model.text_encode(text) # encode text
eempty_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():
        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
```
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.
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
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.

Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3
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 $f: (x_t, t) \rightarrow x_\epsilon$ ($x_\epsilon$ is the sampled result), there are two features:

\[
    f(x_\epsilon, \epsilon) = x_\epsilon \\
    f(x_{t_1}, t_1) = f(x_{t_2}, t_2)
\]
Consistency Model

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

Consistency Model is to find a \( f_\theta \) to fit \( f \).

Step 1: according to [2], define \( f_\theta \):

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

\[
 c_{\text{skip}}(t) = \frac{\sigma^2_{\text{data}}}{(t - \epsilon)^2 + \sigma^2_{\text{data}}},
 c_{\text{out}}(t) = \frac{\sigma_{\text{data}}(t - \epsilon)}{\sqrt{t^2 + \sigma^2_{\text{data}}}}
\]

Step 2: Add consistency distillation loss

\[
 L_{CD}^N(\theta, \theta^-; \phi) :=
\]

\[
 \mathbb{E}[\lambda(t_n)d(f_\theta(x_{t_{n+1}}, t_{n+1}), f_\theta(\hat{x}_{t_n}^\phi, t_n))]
\]
Consistency Model

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

Then it can do the sampling:

Algorithm 1 Multistep Consistency Sampling

<table>
<thead>
<tr>
<th>Input: Consistency model ( f_\theta(\cdot, \cdot) ), sequence of time points ( \tau_1 &gt; \tau_2 &gt; \cdots &gt; \tau_{N-1} ), initial noise ( \hat{x}_T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leftarrow f_\theta(\hat{x}_T, T) )</td>
</tr>
<tr>
<td>for ( n = 1 ) to ( N - 1 ) do</td>
</tr>
<tr>
<td>( \text{Sample } z \sim \mathcal{N}(0, I) )</td>
</tr>
<tr>
<td>( \hat{x}_{\tau_n} \leftarrow x + \sqrt{\tau_n^2 - \epsilon^2} z )</td>
</tr>
<tr>
<td>( x \leftarrow f_\theta(\hat{x}_{\tau_n}, \tau_n) )</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>Output: ( x )</td>
</tr>
</tbody>
</table>
Latent Consistency Model

Algorithm 1 Latent Consistency Distillation (LCD)

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

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

$\theta^- \leftarrow \theta$

repeat

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

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

$\dot{z}_{t_{n+k}} \leftarrow z_{t_{n+k}} + (1 + \omega)\Psi(z_{t_{n+k}}, t_{n+k}, t_n, c) - \omega\Psi(z_{t_{n+k}}, t_{n+k}, t_n, \emptyset)$

$L(\theta, \theta^-; \Psi) \leftarrow d(f_\theta(z_{t_{n+k}}, \omega, c, t_{n+k}), f_{\theta^-}(\dot{z}_{t_{n+k}}, \omega, c, t_n))$

$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta, \theta^-)$

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

until convergence

Latent VAE $E(\cdot)$

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

Diffusion skip number $k$

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

Algorithm 1 Latent Consistency Distillation (LCD)

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

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

\( \theta^- \leftarrow \theta \)

repeat

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

Sample \( z_{n+k} \sim N(\alpha(t_{n+k})z; \sigma^2(t_{n+k})) \) 

\( z_{n+k}^\omega \leftarrow z_{n+k} + (1 + \omega)\Psi(z_{n+k}, t_{n+k}, t_n, c) - \omega\Psi(z_{n+k}, t_{n+k}, t_n, \emptyset) \)

\( \mathcal{L}(\theta, \theta^-; \Psi) \leftarrow \mathcal{L}(f_\theta(z_{n+k}^\omega, \omega, c, t_{n+k}), f_\theta^-(z_{n+k}^\omega, \omega, c, t_n)) \)

\( \theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta, \theta^-) \)

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

until convergence

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

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/
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 Diffusers?

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

For using DDPM, you can:

```python
>>> 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

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

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.

For text-to-image, you have to make a text-image pair.
How can we train

D\ diffusers

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

```python
import torch

data_loader = torch.utils.data.DataLoader(dataset, batch_size=config.train_batch_size, shuffle=True)
```
How can we train

D\textbackslash{}ffusers

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",
        "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",
        "UpBlock2D",
    },
)
```

Define the UNet2DModel

Define the DDPMScheduler
**How can we train**

**D\diffusers**

Image Generation STEP 4 – Define the training

```python
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
    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
**DDPM**: Diffusion Process

**How can we train**

**Diffusers**

Image Generation STEP 5 – Define the evaluate

```python
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]
```

- Load model
- Get images
Text-to-Image STEP 2: choose a base model and load model

https://huggingface.co/models

- 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\texttt{iffusers}

**Text-to-Image STEP 3:** choose a way to train: LoRA training


\begin{align*}
W_0 & \in \mathbb{R}^{d \times k} \\
W_0 + \Delta W &= W_0 + BA \\
B &\in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} \quad \text{and} \quad r \ll \min(d, k)
\end{align*}

```
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

🤗 D\text{ffusers}

Text-to-Image STEP 4: encode the text

```python
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("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

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"]).latent_dist.sample()
        latents = latents * vae.config.scaling_factor
        noise = torch.randn_like(latents)

        # 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)

        # Random timestamp
        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")

        # Model predict

        # 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
```

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

📝 Diffusers

Text-to-Image STEP 6: evaluate

```python
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.
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.
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.

https://dreambooth.github.io/
LoRA reduces the number of trainable parameters by learning pairs of rank-decomposition 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.
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
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