

Exercise: Text to Image Generation using Stable Diffusion Model

Quoc-Thai Nguyen, Xuan-Khai Trinh và Quang-Vinh Dinh
PR-Team: *Dăng-Nhă Nguyen, Minh-Châu Phạm và Hoàng-Nguyễn Vũ*

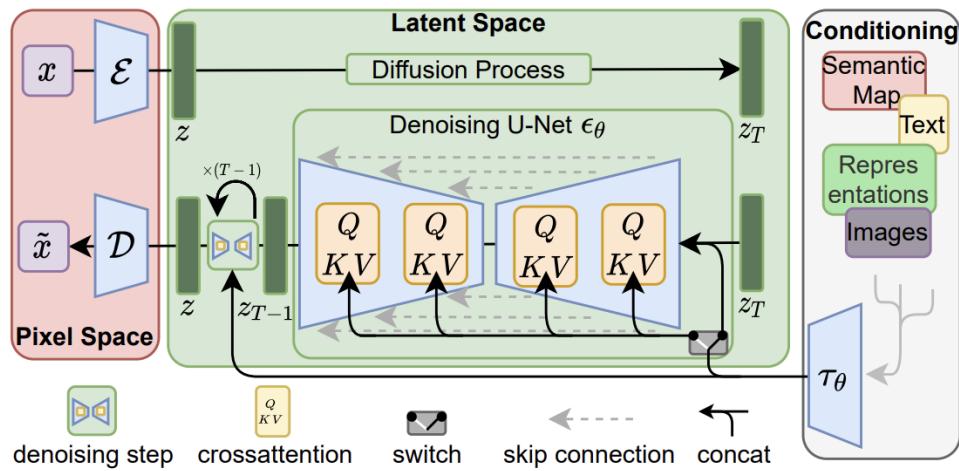
Ngày 8 tháng 4 năm 2024

Phần I. Giới thiệu

The person has high cheekbones, and pointy nose. She is wearing lipstick.



Hình 1: Ví dụ sinh hình ảnh từ văn bản sử dụng mô hình Stable Diffusion.



Hình 2: Mô hình Stable Diffusion.

Sinh hình ảnh từ văn bản hay tổng hợp văn bản thành hình ảnh (Text to Image Generation) là bài toán ngày càng được ứng dụng mạnh mẽ trong lĩnh vực trí tuệ nhân tạo. Các mô hình được huấn luyện với đầu vào là đoạn văn bản và đầu ra là hình ảnh mô tả hoặc chứa các đối tượng được mô tả trong đoạn văn bản. Ví dụ về tổng hợp hình ảnh từ văn bản được mô tả trong Hình 1.

Hiện nay, có nhiều mô hình có thể xây dựng và giải quyết bài toán này có thể kể đến như GANs, Diffusion Models,... Ở trong phần này chúng ta sẽ sử dụng mô hình Stable Diffusion để huấn luyện mô hình giải quyết bài toán này.

Mô hình Stable Diffusion được mô tả như Hình 2. Quá trình tính toán của mô hình được thực hiện như sau:

- Những điểm ảnh đầu vào x sẽ được đẩy qua bộ mã hoá (Encoder) của mô hình VQ-VAE chuyển thành dữ liệu z trong không gian tiềm ẩn có số chiều nhỏ hơn nhiều so với chiều không gian điểm ảnh.
- Dữ liệu trong không gian tiềm ẩn z sẽ được đẩy vào mô hình Diffusion để thêm nhiễu với bước thời gian T để thu được ảnh nhiễu z_T
- Ảnh nhiễu z_T có thể sẽ được kết hợp thêm các điều kiện khác như văn bản mô tả hình ảnh,... đẩy vào mô hình UNet để khử nhiễu khôi phục dữ liệu trong không gian tiềm ẩn z
- Cuối cùng dữ liệu tiềm ẩn z sau khi được khử nhiễu hoàn toàn sẽ đẩy vào khối giải mã (Decoder) của mô hình VQ-VAE để khôi phục ảnh gốc.

Phần II. Text To Image Generation using Stable Diffusion Model

Trong phần này chúng ta sẽ xây dựng và huấn luyện mô hình Stable Diffusion trên bộ dữ liệu CelebA-HQ (Có thể tải về trong phần thực nghiệm). Bộ dữ liệu CelebA-HQ bao gồm:

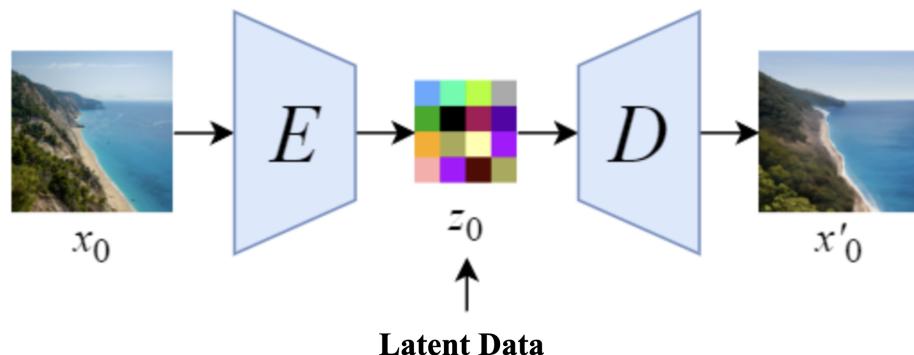
- Thư mục chứa hình ảnh.
- Thư mục chứa các đoạn văn bản mô tả hình ảnh.
- Thư mục chứa các file cho tập train và test.

Quá trình huấn luyện mô hình Stable Diffusion gồm 2 bước:

- Bước 1: Huấn luyện mô hình VQ-VAE phục vụ cho quá trình mã hoá ảnh đầu vào thành dữ liệu trong không gian tiềm ẩn và giải mã để sinh ra ảnh cần dự đoán. Để đơn giản, chúng ta sẽ sử dụng mô hình VQ-VAE đã được huấn luyện trên bộ dữ liệu CelebA-HQ.
- Bước 2: Huấn luyện mô hình Stable Diffusion.

1. Latent Data Extraction

Mô hình VQ-VAE sẽ chuyển từ dữ liệu điểm ảnh đầu vào x thành dữ liệu có chiều nhỏ hơn trong không gian tiềm ảnh z và giải mã z để sinh ra ảnh mới. Để đơn giản trong phần này chúng ta sẽ sử dụng mô hình VQ-VAE đã được huấn luyện trên bộ dữ liệu CelebA-HQ.



Hình 3: Mô hình VQ-VAE mã hoá và giải mã dữ liệu ảnh.

1.1. Data Preparing

```

1 # install libs
2 !pip install -q einops diffusers accelerate
3
4 # download Upsampling Block, ...
5 !wget https://raw.githubusercontent.com/explainingai-code/StableDiffusion-PyTorch/main
      /models/blocks.py
6
7 # import libs
8 import torch
9 import pickle
10 import torchvision
  
```

```

11 import numpy as np
12 import torch.nn as nn
13 from torch.optim import Adam
14
15 from einops import einsum
16 from blocks import get_time_embedding
17 from blocks import DownBlock, MidBlock, UpBlockUnet
18
19 from diffusers import VQModel
20 from transformers import CLIPTextModel
21
22 import os
23 import glob
24 import random
25 from PIL import Image
26 from tqdm import tqdm
27 from torchvision.utils import make_grid
28 from torch.utils.data import DataLoader
29 from torch.utils.data import Dataset
30
31 # download dataset
32 !mkdir CelebAMask-HQ
33 %cd CelebAMask-HQ
34
35 # Image
36 !gdown 1j1Q8umhpJo8lVgC9q4_1q_t_Frv1kZ3f
37 !unzip image.zip
38
39 # Caption
40 !gdown 1X1EFCyralNN2Bg3LhellL_lShrSrmTitW
41 !unzip text.zip
42
43 # Train / Test set
44 !gdown 1GdeTdBpi_IV7AuBpJAhLElqjswRm0y-7 -O train.pickle
45 !gdown 1JNxgdvPMI_HHUq2-JUuJp8L7cD-740Af -O test.pickle
46
47 !mv images CelebA-HQ-img
48 !rm image.zip text.zip

```

1.2. Dataset

```

1 def load_latents(latent_path):
2     """
3         Simple utility to save latents to speed up ldm training
4         :param latent_path:
5         :return:
6     """
7     latent_maps = {}
8     for fname in glob.glob(os.path.join(latent_path, '*.pkl')):
9         s = pickle.load(open(fname, 'rb'))
10        for k, v in s.items():
11            latent_maps[k] = v[0]
12    return latent_maps
13
14 class CelebDataset(Dataset):
15     """
16         Celeb dataset will by default centre crop and resize the images.
17         This can be replaced by any other dataset. As long as all the images
18         are under one directory.
19     """

```

```
20
21     def __init__(self, split, im_path, im_size=256, im_channels=3, im_ext='jpg',
22                  use_latents=False, latent_path=None, condition_config=None):
23         self.split = split
24         if self.split != 'all':
25             self.split_filter = pickle.load(open(f'./data/CelebAMask-HQ/{self.split}.pickle', 'rb'))
26
27         self.im_size = im_size
28         self.im_channels = im_channels
29         self.im_ext = im_ext
30         self.im_path = im_path
31         self.latent_maps = None
32         self.use_latents = False
33
34         self.condition_types = [] if condition_config is None else condition_config['condition_types']
35         self.images, self.texts = self.load_images(im_path)
36
37         # Whether to load images or to load latents
38         if use_latents and latent_path is not None:
39             latent_maps = load_latents(latent_path)
40             if len(latent_maps) == len(self.images):
41                 self.use_latents = True
42                 self.latent_maps = latent_maps
43                 print('Found {} latents'.format(len(self.latent_maps)))
44             else:
45                 print('Latents not found')
46
47     def load_images(self, im_path):
48         """
49             Gets all images from the path specified
50             and stacks them all up
51         """
52         assert os.path.exists(im_path), "images path {} does not exist".format(im_path)
53
54         ims = []
55         fnames = glob.glob(os.path.join(im_path, 'CelebA-HQ-img/*.{png}'.format('png')))
56         fnames += glob.glob(os.path.join(im_path, 'CelebA-HQ-img/*.{jpg}'.format('jpg')))
57         fnames += glob.glob(os.path.join(im_path, 'CelebA-HQ-img/*.{jpeg}'.format('jpeg')))
58
59         texts = []
60
61         for fname in tqdm(fnames):
62             im_name = os.path.split(fname)[1].split('.')[0]
63
64             if self.split != 'all':
65                 if im_name not in self.split_filter:
66                     continue
67
68             ims.append(fname)
69
70             if 'text' in self.condition_types:
71                 captions_im = []
72                 with open(os.path.join(im_path, 'celeba-caption/{}.txt'.format(im_name))) as f:
73                     for line in f.readlines():
74                         captions_im.append(line.strip())
75
76             texts.append(captions_im)
```

```

75     if 'text' in self.condition_types:
76         assert len(texts) == len(ims), "Condition Type Text but could not find
77         captions for all images"
78
78     print('Found {} images'.format(len(ims)))
79     print('Found {} captions'.format(len(texts)))
80
81     return ims, texts
82
83 def __len__(self):
84     return len(self.images)
85
86 def __getitem__(self, index):
87     ##### Set Conditioning Info #####
88     cond_inputs = {}
89     if 'text' in self.condition_types:
90         cond_inputs['text'] = random.sample(self.texts[index], k=1)[0]
91     #####
92
93     if self.use_latents:
94         latent = self.latent_maps[self.images[index]]
95         if len(self.condition_types) == 0:
96             return latent
97         else:
98             return latent, cond_inputs
99     else:
100        im = Image.open(self.images[index])
101        im_tensor = torchvision.transforms.Compose([
102            torchvision.transforms.Resize(self.im_size),
103            torchvision.transforms.CenterCrop(self.im_size),
104            torchvision.transforms.ToTensor(),
105        ])(im)
106        im.close()
107
108    # Convert input to -1 to 1 range.
109    im_tensor = (2 * im_tensor) - 1
110    if len(self.condition_types) == 0:
111        return im_tensor
112    else:
113        return im_tensor, cond_inputs

```

1.3. Config

```

1 config = {
2     "dataset_params": {
3         "im_path": "data/CelebAMask-HQ",
4         "im_channels": 3,
5         "im_size": 256,
6         "name": "celebhq"
7     },
8     "diffusion_params": {
9         "num_timesteps": 1000,
10        "beta_start": 0.00085,
11        "beta_end": 0.012
12    },
13    "ldm_params": {
14        "down_channels": [256, 384, 512, 768],
15        "mid_channels": [768, 512],
16        "down_sample": [True, True, True],
17        "attn_down": [True, True, True],

```

```

18     "time_emb_dim": 512,
19     "norm_channels": 32,
20     "num_heads": 16,
21     "conv_out_channels": 128,
22     "num_down_layers": 2,
23     "num_mid_layers": 2,
24     "num_up_layers": 2,
25     "condition_config": {
26         "condition_types": ["text"],
27         "text_condition_config": {
28             "text_embed_model": "clip",
29             "train_text_embed_model": False,
30             "text_embed_dim": 512,
31             "cond_drop_prob": 0.1
32         }
33     },
34 },
35 "train_params": {
36     "seed": 1111,
37     "task_name": "celebhq",
38     "ldm_batch_size": 16,
39     "ldm_epochs": 100,
40     "num_samples": 1,
41     "num_grid_rows": 1,
42     "ldm_lr": 0.000005,
43     "save_latents": True,
44     "vqvae_latent_dir_name": 'vqvae_latents',
45     "cf_guidance_scale": 1.0,
46     "ldm_ckpt_name": "ddpm_ckpt_text_cond_clip.pth",
47 }
48 }
```

1.4. Latent Data Extraction

Trong phần này chúng ta sẽ trích xuất dữ liệu từ mô hình VQ-VAE sử dụng pre-trained trên bộ CelebA-HQ sau đó lưu vào tệp vqvae/diffusion_pytorch_model.bin

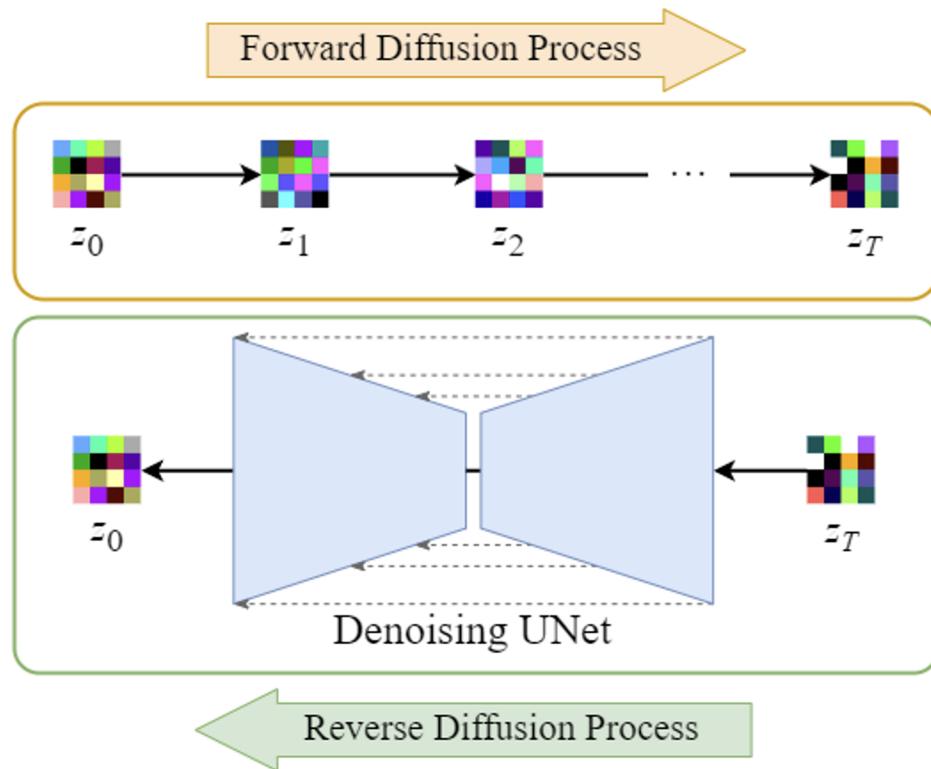
```

1 dataset_config = config['dataset_params']
2 train_config = config['train_params']
3 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
4
5 im_dataset = CelebDataset(split='all',
6                           im_path=dataset_config['im_path'],
7                           im_size=dataset_config['im_size'],
8                           im_channels=dataset_config['im_channels'])
9 data_loader = DataLoader(im_dataset, batch_size=1, shuffle=False)
10
11 num_images = train_config['num_samples']
12 ngrid = train_config['num_grid_rows']
13
14 idxs = torch.randint(0, len(im_dataset) - 1, (num_images,))
15 ims = torch.cat([im_dataset[idx][None, :] for idx in idxs]).float()
16 ims = ims.to(device)
17
18 vae = VQModel.from_pretrained("CompVis/ldm-celebahq-256", subfolder="vqvae")
19 vae.eval()
20 vae = vae.to(device)
21
22 os.makedirs(os.path.join(train_config['task_name']), exist_ok=True)
23 with torch.no_grad():
24     encoded_output = vae.encode(ims).latents
25     decoded_output = vae.decode(encoded_output).sample
```

```
26     encoded_output = torch.clamp(encoded_output, -1., 1.)
27     encoded_output = (encoded_output + 1) / 2
28     decoded_output = torch.clamp(decoded_output, -1., 1.)
29     decoded_output = (decoded_output + 1) / 2
30     ims = (ims + 1) / 2
31
32     encoder_grid = make_grid(encoded_output.cpu(), nrow=ngrid)
33     decoder_grid = make_grid(decoded_output.cpu(), nrow=ngrid)
34     input_grid = make_grid(ims.cpu(), nrow=ngrid)
35     encoder_grid = torchvision.transforms.ToPILImage()(encoder_grid)
36     decoder_grid = torchvision.transforms.ToPILImage()(decoder_grid)
37     input_grid = torchvision.transforms.ToPILImage()(input_grid)
38
39     input_grid.save(os.path.join(train_config['task_name'], 'input_samples.png'))
40     encoder_grid.save(os.path.join(train_config['task_name'], 'encoded_samples.png'))
41     decoder_grid.save(os.path.join(train_config['task_name'], 'reconstructed_samples.
42     png'))
43
43     os.makedirs(os.path.join(train_config['task_name'], train_config[',
44     vqvae_latent_dir_name]), exist_ok=True)
45     if train_config['save_latents']:
46         # save Latents (but in a very unoptimized way)
47         latent_path = os.path.join(train_config['task_name'], train_config[',
48         vqvae_latent_dir_name])
49         latent_fnames = glob.glob(os.path.join(train_config['task_name'], train_config[
50             'vqvae_latent_dir_name'], '*.pkl'))
51
51         assert len(latent_fnames) == 0, 'Latents already present. Delete all latent
52         files and re-run'
53         if not os.path.exists(latent_path):
54             os.mkdir(latent_path)
55             print('Saving Latents for {}'.format(dataset_config['name']))
56
56         fname_latent_map = {}
57         part_count = 0
58         count = 0
59         for idx, im in enumerate(tqdm(data_loader)):
60             encoded_output = vae.encode(im.float().to(device)).latents
61             fname_latent_map[im_dataset.images[idx]] = encoded_output.cpu()
62             # Save latents every 1000 images
63             if (count+1) % 1000 == 0:
64                 pickle.dump(fname_latent_map, open(os.path.join(latent_path,
65                     '{}.pkl'.format(part_count)), 'wb'))
66                 part_count += 1
67                 fname_latent_map = {}
68                 count += 1
69             if len(fname_latent_map) > 0:
70                 pickle.dump(fname_latent_map, open(os.path.join(latent_path,
71                     '{}.pkl'.format(part_count)), 'wb'))
72                 print('Done saving latents')
```

2. Latent Data Extraction

Sau khi chúng ta đã trích xuất các đặc trưng ảnh dựa vào mô hình VQ-VAE. Trong phần này chúng ta sẽ huấn luyện mô hình Stable Diffusion. Mô hình CLIP sẽ được sử dụng để biểu diễn văn bản thành ma trận vector.



Hình 4: Mô hình Stable Diffusion.

Sau khi chúng ta đã cài đặt các thư viện, có file data được trích xuất từ Phần 1. Chúng ta tiến hành các bước sau:

2.1. Dataset

Phần này, thay vì dữ liệu đầy vào mô hình là các điểm ảnh, ta sẽ lấy dữ liệu tiềm ảnh đầy vào mô hình.

```

1 def load_latents(latent_path):
2     """
3         Simple utility to save latents to speed up ldm training
4         :param latent_path:
5         :return:
6         """
7     latent_maps = {}
8     for fname in glob.glob(os.path.join(latent_path, '*.pkl')):
9         s = pickle.load(open(fname, 'rb'))
10        for k, v in s.items():
11            latent_maps[k] = v[0]
12    return latent_maps
13
14 class CelebDataset(Dataset):
15     """
16         Celeb dataset will by default centre crop and resize the images.
17         This can be replaced by any other dataset. As long as all the images
18         are under one directory.

```

```
19      """
20
21  def __init__(self, split, im_path, im_size=256, im_channels=3, im_ext='jpg',
22               use_latents=False, latent_path=None, condition_config=None):
23      self.split = split
24      if self.split != 'all':
25          self.split_filter = pickle.load(open(f'/kaggle/working/data/CelebAMask-HQ/{self.split}.pickle', 'rb'))
26
27      self.im_size = im_size
28      self.im_channels = im_channels
29      self.im_ext = im_ext
30      self.im_path = im_path
31      self.latent_maps = None
32      self.use_latents = False
33
34      self.condition_types = [] if condition_config is None else condition_config['condition_types']
35      self.images, self.texts = self.load_images(im_path)
36
37      # Whether to load images or to load latents
38      if use_latents and latent_path is not None:
39          latent_maps = load_latents(latent_path)
40          if len(latent_maps) >= len(self.images):
41              self.use_latents = True
42              self.latent_maps = latent_maps
43              print('Found {} latents'.format(len(self.latent_maps)))
44      else:
45          print('Latents not found')
46
47  def load_images(self, im_path):
48      """
49      Gets all images from the path specified
50      and stacks them all up
51      """
52      assert os.path.exists(im_path), "images path {} does not exist".format(im_path)
53
54      ims = []
55      fnames = glob.glob(os.path.join(im_path, 'CelebA-HQ-img/*.{png}'))
56      fnames += glob.glob(os.path.join(im_path, 'CelebA-HQ-img/*.{jpg}'))
57      fnames += glob.glob(os.path.join(im_path, 'CelebA-HQ-img/*.{jpeg}'))
58
59      texts = []
60
61      for fname in tqdm(fnames):
62          im_name = os.path.split(fname)[1].split('.')[0]
63
64          if self.split != 'all':
65              if im_name not in self.split_filter:
66                  continue
67
68          ims.append(fname)
69
70          if 'text' in self.condition_types:
71              captions_im = []
72              with open(os.path.join(im_path, 'celeba-caption/{}.txt'.format(im_name))) as f:
73                  for line in f.readlines():
74                      captions_im.append(line.strip())
75
76          texts.append(captions_im)
```

```

74     if 'text' in self.condition_types:
75         assert len(texts) == len(ims), "Condition Type Text but could not find
76         captions for all images"
77
78         print('Found {} images'.format(len(ims)))
79         print('Found {} captions'.format(len(texts)))
80
81     return ims, texts
82
83 def __len__(self):
84     return len(self.images)
85
86 def __getitem__(self, index):
87     ##### Set Conditioning Info #####
88     cond_inputs = {}
89     if 'text' in self.condition_types:
90         cond_inputs['text'] = random.sample(self.texts[index], k=1)[0]
91     #####
92
93     if self.use_latents:
94         latent = self.latent_maps[self.images[index]]
95         if len(self.condition_types) == 0:
96             return latent
97         else:
98             return latent, cond_inputs
99     else:
100        im = Image.open(self.images[index])
101        im_tensor = torchvision.transforms.Compose([
102            torchvision.transforms.Resize(self.im_size),
103            torchvision.transforms.CenterCrop(self.im_size),
104            torchvision.transforms.ToTensor(),
105        ])(im)
106        im.close()
107
108        # Convert input to -1 to 1 range.
109        im_tensor = (2 * im_tensor) - 1
110        if len(self.condition_types) == 0:
111            return im_tensor
112        else:
113            return im_tensor, cond_inputs

```

2.2. Linear Noise Scheduler

```

1 class LinearNoiseScheduler:
2     """
3         Linear noise scheduler      c      s      d ng   trong DDPM
4     """
5
6     def __init__(self, num_timesteps, beta_start, beta_end):
7         self.num_timesteps = num_timesteps
8         self.beta_start = beta_start
9         self.beta_end = beta_end
10
11         # Mimicking how compvis repo creates schedule
12         # Test later
13         self.betas = (
14             torch.linspace(beta_start ** 0.5, beta_end ** 0.5, num_timesteps) ** 2
15         )
16

```

```

17
18     self.alphas = 1. - self.betas
19     self.alpha_cum_prod = torch.cumprod(self.alphas, dim=0)
20     self.sqrt_alpha_cum_prod = torch.sqrt(self.alpha_cum_prod)
21     self.sqrt_one_minus_alpha_cum_prod = torch.sqrt(1 - self.alpha_cum_prod)
22
23 def add_noise(self, original, noise, t):
24     """
25         H m forward cho qu tr nh diffusion
26     :param original: Image on which noise is to be applied
27     :param noise: Random Noise Tensor (from normal dist)
28     :param t: timestep of the forward process of shape -> (B,)
29     :return:
30     """
31
32     original_shape = original.shape
33     batch_size = original_shape[0]
34
35     sqrt_alpha_cum_prod = self.sqrt_alpha_cum_prod.to(original.device)[t].reshape(
36     batch_size)
36     sqrt_one_minus_alpha_cum_prod = self.sqrt_one_minus_alpha_cum_prod.to(original
37     .device)[t].reshape(batch_size)
38
38     # Reshape till (B,) becomes (B,1,1,1) if image is (B,C,H,W)
39     for _ in range(len(original_shape) - 1):
40         sqrt_alpha_cum_prod = sqrt_alpha_cum_prod.unsqueeze(-1)
41     for _ in range(len(original_shape) - 1):
42         sqrt_one_minus_alpha_cum_prod = sqrt_one_minus_alpha_cum_prod.unsqueeze
43         (-1)
44
44     # Apply and Return Forward process equation
45     return (sqrt_alpha_cum_prod.to(original.device) * original
46             + sqrt_one_minus_alpha_cum_prod.to(original.device) * noise)
47
47 def sample_prev_timestep(self, xt, noise_pred, t):
48     """
49         Use the noise prediction by model to get
50         xt-1 using xt and the nosie predicted
51     :param xt: current timestep sample
52     :param noise_pred: model noise prediction
53     :param t: current timestep we are at
54     :return:
55     """
56
56     x0 = ((xt - (self.sqrt_one_minus_alpha_cum_prod.to(xt.device)[t] * noise_pred)
57     ) /
57         torch.sqrt(self.alpha_cum_prod.to(xt.device)[t]))
58     x0 = torch.clamp(x0, -1., 1.)
59
59     mean = xt - ((self.betas.to(xt.device)[t]) * noise_pred) / (self.
60     sqrt_one_minus_alpha_cum_prod.to(xt.device)[t])
61     mean = mean / torch.sqrt(self.alphas.to(xt.device)[t])
62
63     if t == 0:
64         return mean, x0
65     else:
66         variance = (1 - self.alpha_cum_prod.to(xt.device)[t - 1]) / (1.0 - self.
67         alpha_cum_prod.to(xt.device)[t])
68         variance = variance * self.betas.to(xt.device)[t]
69         sigma = variance ** 0.5
70         z = torch.randn(xt.shape).to(xt.device)
71         return mean + sigma * z, x0

```

2.3. UNet Model

```

1 class Unet(nn.Module):
2     r"""
3         Unet model comprising
4             Down blocks, Midblocks and Uplocks
5     """
6
7     def __init__(self, im_channels, model_config):
8         super().__init__()
9         self.down_channels = model_config['down_channels']
10        self.mid_channels = model_config['mid_channels']
11        self.t_emb_dim = model_config['time_emb_dim']
12        self.down_sample = model_config['down_sample']
13        self.num_down_layers = model_config['num_down_layers']
14        self.num_mid_layers = model_config['num_mid_layers']
15        self.num_up_layers = model_config['num_up_layers']
16        self.attns = model_config['attn_down']
17        self.norm_channels = model_config['norm_channels']
18        self.num_heads = model_config['num_heads']
19        self.conv_out_channels = model_config['conv_out_channels']
20
21        # Validating Unet Model configurations
22        assert self.mid_channels[0] == self.down_channels[-1]
23        assert self.mid_channels[-1] == self.down_channels[-2]
24        assert len(self.down_sample) == len(self.down_channels) - 1
25        assert len(self.attns) == len(self.down_channels) - 1
26
27        ##### Class, Mask and Text Conditioning Config #####
28        self.condition_config = model_config['condition_config']
29        self.text_cond = True
30        self.text_embed_dim = self.condition_config['text_condition_config'][
31            'text_embed_dim']
32        self.cond = self.text_cond
33        #####
34        self.conv_in = nn.Conv2d(im_channels, self.down_channels[0], kernel_size=3,
35        padding=1)
36
37        # Initial projection from sinusoidal time embedding
38        self.t_proj = nn.Sequential(
39            nn.Linear(self.t_emb_dim, self.t_emb_dim),
40            nn.SILU(),
41            nn.Linear(self.t_emb_dim, self.t_emb_dim)
42        )
43
44        self.up_sample = list(reversed(self.down_sample))
45        self.downs = nn.ModuleList([])
46
47        # Build the Downblocks
48        for i in range(len(self.down_channels) - 1):
49            # Cross Attention and Context Dim only needed if text condition is present
50            self.downs.append(DownBlock(self.down_channels[i], self.down_channels[i + 1],
51                self.t_emb_dim,
52                down_sample=self.down_sample[i],
53                num_heads=self.num_heads,
54                num_layers=self.num_down_layers,
55                attn=self.attns[i], norm_channels=self.
56                norm_channels,
57                cross_attn=self.text_cond,
58                context_dim=self.text_embed_dim))

```

```
55     self.mids = nn.ModuleList([])
56     # Build the Midblocks
57     for i in range(len(self.mid_channels) - 1):
58         self.mids.append(MidBlock(self.mid_channels[i], self.mid_channels[i + 1],
59                                   self.t_emb_dim,
60                                   num_heads=self.num_heads,
61                                   num_layers=self.num_mid_layers,
62                                   norm_channels=self.norm_channels,
63                                   cross_attn=self.text_cond,
64                                   context_dim=self.text_embed_dim))
65
66     self.ups = nn.ModuleList([])
67     # Build the Upblocks
68     for i in reversed(range(len(self.down_channels) - 1)):
69         self.ups.append(
70             UpBlockUnet(self.down_channels[i] * 2, self.down_channels[i - 1] if i
71 != 0 else self.conv_out_channels,
72                         self.t_emb_dim, up_sample=self.down_sample[i],
73                         num_heads=self.num_heads,
74                         num_layers=self.num_up_layers,
75                         norm_channels=self.norm_channels,
76                         cross_attn=self.text_cond,
77                         context_dim=self.text_embed_dim))
78
79     self.norm_out = nn.GroupNorm(self.norm_channels, self.conv_out_channels)
80     self.conv_out = nn.Conv2d(self.conv_out_channels, im_channels, kernel_size=3,
81                            padding=1)
82
83     def forward(self, x, t, cond_input=None):
84         # Shapes assuming downblocks are [C1, C2, C3, C4]
85         # Shapes assuming midblocks are [C4, C4, C3]
86
87         # B x C x H x W -> # B x C1 x H x W
88         out = self.conv_in(x)
89
90         # t_emb -> B x t_emb_dim
91         t_emb = get_time_embedding(torch.as_tensor(t).long(), self.t_emb_dim)
92         t_emb = self.t_proj(t_emb)
93
94         context_hidden_states = None
95         context_hidden_states = cond_input['text']
96         down_outs = []
97
98         for idx, down in enumerate(self.downs):
99             down_outs.append(out)
100            out = down(out, t_emb, context_hidden_states)
101            # down_outs [B x C1 x H x W, B x C2 x H/2 x W/2, B x C3 x H/4 x W/4]
102            # out B x C4 x H/4 x W/4
103
104         for mid in self.mids:
105             out = mid(out, t_emb, context_hidden_states)
106             # out B x C3 x H/4 x W/4
107
108         for up in self.ups:
109             down_out = down_outs.pop()
110             out = up(out, down_out, t_emb, context_hidden_states)
111             # out [B x C2 x H/4 x W/4, B x C1 x H/2 x W/2, B x 16 x H x W]
112             out = self.norm_out(out)
113             out = nn.SiLU()(out)
```

```

112     out = self.conv_out(out)
113     # out B x C x H x W
114     return out

```

2.4. Config

```

1 config = {
2     "dataset_params": {
3         "im_path": "data/CelebAMask-HQ",
4         "im_channels": 3,
5         "im_size": 256,
6         "name": "celebhq"
7     },
8     "diffusion_params": {
9         "num_timesteps": 1000,
10        "beta_start": 0.00085,
11        "beta_end": 0.012
12    },
13    "ldm_params": {
14        "down_channels": [128, 256, 384, 512],
15        "mid_channels": [512, 384],
16        "down_sample": [True, True, True],
17        "attn_down": [True, True, True],
18        "time_emb_dim": 512,
19        "norm_channels": 32,
20        "num_heads": 16,
21        "conv_out_channels": 128,
22        "num_down_layers": 2,
23        "num_mid_layers": 2,
24        "num_up_layers": 2,
25        "condition_config": {
26            "condition_types": ["text"],
27            "text_condition_config": {
28                "text_embed_model": "clip",
29                "train_text_embed_model": False,
30                "text_embed_dim": 512,
31                "cond_drop_prob": 0.1
32            }
33        }
34    },
35    "train_params": {
36        "seed": 1111,
37        "task_name": "celebhq",
38        "ldm_batch_size": 1,
39        "ldm_epochs": 30,
40        "num_samples": 1,
41        "num_grid_rows": 1,
42        "ldm_lr": 0.000005,
43        "save_latents": True,
44        "vqvae_latent_dir_name": 'vqvae_latents',
45        "cf_guidance_scale": 1.0,
46        "ldm_ckpt_name": "ddpm_ckpt_text_cond_clip.pth",
47    }
48}

```

2.5. Text Representation

Phần này chúng ta sẽ sử dụng CLIP model để chuyển văn bản thành vector.

```

1 def get_text_representation(text, text_tokenizer, text_model, device,
2                             truncation=True,
3                             padding='max_length',

```

```

4             max_length=77):
5     token_output = text_tokenizer(text,
6                                     truncation=truncation,
7                                     padding=padding,
8                                     return_attention_mask=True,
9                                     max_length=max_length)
10    indexed_tokens = token_output['input_ids']
11    att_masks = token_output['attention_mask']
12    tokens_tensor = torch.tensor(indexed_tokens).to(device)
13    mask_tensor = torch.tensor(att_masks).to(device)
14    text_embed = text_model(tokens_tensor, attention_mask=mask_tensor).
15    last_hidden_state
16    return text_embed
17
18 def drop_text_condition(text_embed, im, empty_text_embed, text_drop_prob):
19     if text_drop_prob > 0:
20         text_drop_mask = torch.zeros((im.shape[0]), device=im.device).float().uniform_
21         (0,
22
23         1) < text_drop_prob
24         assert empty_text_embed is not None, ("Text Conditioning required as well as"
25                                             " text dropping but empty text representation
26                                             not created")
27         text_embed[text_drop_mask, :, :] = empty_text_embed[0]
28     return text_embed

```

2.6. Training

```

1 diffusion_config = config['diffusion_params']
2 dataset_config = config['dataset_params']
3 diffusion_model_config = config['ldm_params']
4 train_config = config['train_params']
5 latents_channel = 3
6 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
7
8 # Create the noise scheduler
9 scheduler = LinearNoiseScheduler(num_timesteps=diffusion_config['num_timesteps'],
10                                   beta_start=diffusion_config['beta_start'],
11                                   beta_end=diffusion_config['beta_end'])
12
13 # Instantiate Condition related components
14 text_tokenizer = None
15 text_model = None
16 empty_text_embed = None
17 condition_config = diffusion_model_config['condition_config']
18 condition_types = condition_config['condition_types']
19
20 with torch.no_grad():
21     # Load tokenizer and text model based on config
22     text_tokenizer = CLIPTextModel.from_pretrained('openai/clip-vit-base-patch16')
23     text_model = CLIPTextModel.from_pretrained('openai/clip-vit-base-patch16').to(
24     device)
25     text_model.eval()
26
27     empty_text_embed = get_text_representation([''], text_tokenizer, text_model,
28                                                 device)
29
30 im_dataset = CelebDataset(split='train',
31                           im_path=dataset_config['im_path'],
32                           im_size=dataset_config['im_size'],
33                           )

```

```
31             im_channels=dataset_config['im_channels'],
32             use_latents=True,
33             latent_path=os.path.join(train_config['task_name'],
34                                     train_config['vqvae_latent_dir_name'],
35                                     ]),
36             condition_config=condition_config)
37
38 filters = list(range(0, len(im_dataset), 10))
39 im_dataset = torch.utils.data.Subset(im_dataset, filters)
40
41 data_loader = DataLoader(im_dataset,
42                         batch_size=train_config['ldm_batch_size'],
43                         shuffle=True)
44
45 model = Unet(im_channels=latents_channel,
46               model_config=diffusion_model_config).to(device)
47 model.train()
48
49 num_epochs = train_config['ldm_epochs']
50 optimizer = Adam(model.parameters(), lr=train_config['ldm_lr'])
51 criterion = torch.nn.MSELoss()
52
53 # Run training
54 for epoch_idx in range(num_epochs):
55     losses = []
56     for data in tqdm(data_loader):
57         cond_input = None
58         if condition_config is not None:
59             im, cond_input = data
60         else:
61             im = data
62         optimizer.zero_grad()
63         im = im.float().to(device)
64
65         ##### Handling Conditional Input #####
66         if 'text' in condition_types:
67             with torch.no_grad():
68                 text_condition = get_text_representation(cond_input['text'],
69                                              text_tokenizer,
70                                              text_model,
71                                              device)
72
73             text_drop_prob = condition_config['text_condition_config'][
74             'cond_drop_prob']
75             text_condition = drop_text_condition(text_condition, im,
76             empty_text_embed, text_drop_prob)
77             cond_input['text'] = text_condition
78
79         # Sample random noise
80         noise = torch.randn_like(im).to(device)
81
82         # Sample timestep
83         t = torch.randint(0, diffusion_config['num_timesteps'],
84                           (im.shape[0],)).to(
85                           device)
86
87         # Add noise to images according to timestep
88         noisy_im = scheduler.add_noise(im, noise, t)
89         noise_pred = model(noisy_im, t, cond_input=cond_input)
90         loss = criterion(noise_pred, noise)
91         losses.append(loss.item())
```

```

87         loss.backward()
88         optimizer.step()
89
90     print('Finished epoch:{} | Loss : {:.4f}'.format(
91         epoch_idx + 1,
92         np.mean(losses)))
93     torch.save(model.state_dict(), os.path.join(train_config['task_name'],
94                                                 train_config['ldm_ckpt_name']))
95
96 print('Done Training ...')

```

3. Inference

Sau khi huấn luyện mô hình xong, chúng ta sẽ sinh ra ảnh mới dựa vào văn bản mô tả.

```

1 # prepare model
2 model = Unet(im_channels=latents_channel,
3               model_config=diffusion_model_config).to(device)
4 model.eval()
5 if os.path.exists(os.path.join(train_config['task_name'],
6                             train_config['ldm_ckpt_name'])):
7     print('Loaded unet checkpoint')
8     model.load_state_dict(torch.load(os.path.join(train_config['task_name'],
9                                     train_config['ldm_ckpt_name']),
10                                    map_location=device))
11 else:
12     raise Exception('Model checkpoint {} not found'.format(os.path.join(train_config['task_name'],
13                                                               train_config['ldm_ckpt_name'])))
14
15 vae = VQModel.from_pretrained("CompVis/ldm-celebahq-256", subfolder="vqvae")
16 vae.eval()
17 vae = vae.to(device)
18
19 # generate
20 with torch.no_grad():
21     xt = torch.randn((1, latents_channel, 64, 64)).to(device)
22     text_prompt = ['She is a woman with blond hair. She is wearing lipstick.']
23     neg_prompt = ['He is a man.']
24     empty_prompt = []
25     text_prompt_embed = get_text_representation(text_prompt,
26                                               text_tokenizer,
27                                               text_model,
28                                               device)
29     # Can replace empty prompt with negative prompt
30     empty_text_embed = get_text_representation(empty_prompt, text_tokenizer,
31                                                text_model, device)
32     assert empty_text_embed.shape == text_prompt_embed.shape
33
34     uncond_input = {
35         'text': empty_text_embed
36     }
37     cond_input = {
38         'text': text_prompt_embed
39     }
39 ##########
40
41     # By default classifier free guidance is disabled
42     # Change value in config or change default value here to enable it
43     cf_guidance_scale = train_config['cf_guidance_scale']
44

```

```
45 ##### Sampling Loop #####
46 for i in tqdm(reversed(range(diffusion_config['num_timesteps']))):
47     # Get prediction of noise
48     t = (torch.ones((xt.shape[0],)) * i).long().to(device)
49     noise_pred_cond = model(xt, t, cond_input)
50
51     if cf_guidance_scale > 1:
52         noise_pred_uncond = model(xt, t, uncond_input)
53         noise_pred = noise_pred_uncond + cf_guidance_scale * (noise_pred_cond -
54         noise_pred_uncond)
55     else:
56         noise_pred = noise_pred_cond
57
58     # Use scheduler to get x0 and xt-1
59     xt, x0_pred = scheduler.sample_prev_timestep(xt, noise_pred, torch.as_tensor(i)
60 .to(device))
61
62     # Save x0
63     # ims = torch.clamp(xt, -1., 1.).detach().cpu()
64     if i == 0:
65         # Decode ONLY the final iamge to save time
66         ims = vae.decode(xt).sample
67     else:
68         ims = x0_pred
69
70     ims = torch.clamp(ims, -1., 1.).detach().cpu()
71     ims = (ims + 1) / 2
72     grid = make_grid(ims, nrow=1)
73     img = torchvision.transforms.ToPILImage()(grid)
74
75     if not os.path.exists(os.path.join(train_config['task_name'], 'cond_text_samples')):
76         os.mkdir(os.path.join(train_config['task_name'], 'cond_text_samples'))
77         img.save(os.path.join(train_config['task_name'], 'cond_text_samples', 'x0_{}.png'.format(i)))
78         img.close()
79
80 # show image
81 test_image = Image.open(os.path.join(train_config['task_name'], 'cond_text_samples', 'x0_{}.png'.format(0)))
```

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Mục đích của bài toán Text to Image Generation là gì?

- a) Sinh hình ảnh dựa vào văn bản
- b) Phân loại hình ảnh
- c) Phân loại văn bản
- d) Sinh mô tả cho hình ảnh

Câu hỏi 2 Mô hình nào có thể được huấn luyện để sinh hình ảnh từ văn bản?

- a) Stable Diffusion
- b) BERT
- c) CLIP
- d) RoBERTa

Câu hỏi 3 Khối Encoder và Decoder trong phần thực nghiệm được sử dụng từ mô hình nào?

- a) BART
- b) BERT
- c) VQ-VAE
- d) CLIP

Câu hỏi 4 Mô hình nào sau đây không được sử dụng để biểu diễn văn bản thành vector?

- a) BERT
- b) CLIP
- c) VGG
- d) GPT

Câu hỏi 5 Trong quá trình denoising được mô tả trong bài, condition nào không thể được bổ sung vào quá trình huấn luyện mô hình Stable Diffusion?

- a) Semantic Map
- b) Text
- c) Images
- d) Sound

Câu hỏi 6 Trong phần thực nghiệm, bộ dữ liệu nào được sử dụng huấn luyện mô hình Stable Diffusion để giải quyết bài toán Text-to-Image Generation?

- a) CIFAR10
- b) CIFAR100
- c) IMAGENET
- d) CelebA-HQ

Câu hỏi 7 Đầu vào của mô hình UNet không bao gồm dữ liệu nào sau đây?

- a) Random Latent Noise
- b) Timestemp Embedding
- c) Conditioning Embedding
- d) Pixel

Câu hỏi 8 Số lượng ảnh trong bộ dữ liệu được sử dụng huấn luyện mô hình Text-to-Image Generation là?

- a) 20000
- b) 30000
- c) 40000
- d) 50000

Câu hỏi 9 Mỗi ảnh trong bộ dữ liệu được sử dụng huấn luyện mô hình Text-to-Image Generation sẽ có bao nhiêu caption?

- a) 5
- b) 7
- c) 8
- d) 10

Câu hỏi 10 Mô hình VQ-VAE nào sau đây được sử dụng trong phần thực nghiệm?

- a) CompVis/ldm-celebahq-256
- b) CompVis/stable-diffusion-v1-1
- c) CompVis/ldm-text2im-large-256
- d) CompVis/stable-diffusion-safety-checker

- *Hết* -