

Text-Guided Image Generation using Conditional Flow Matching

Quoc-Thai Nguyen, Quang-Hien Ho và Quang-Vinh Dinh

Ngày 7 tháng 4 năm 2025

Phần 1. Giới thiệu



A photo of a chair swed in half



A photo of an open door

Hình 1: Text-Guided Image Generation.

Text-Guided Image Generation là một biến thể nâng cao của bài toán Text-to-Image Generation, trong đó mô hình không chỉ tạo hình ảnh từ văn bản mà còn có khả năng chỉnh sửa hình ảnh có sẵn theo yêu cầu văn bản. Điều này có nghĩa là mô hình có thể hiểu và điều chỉnh hình ảnh dựa trên yêu cầu văn bản mà vẫn giữ nguyên những phần không bị ảnh hưởng.

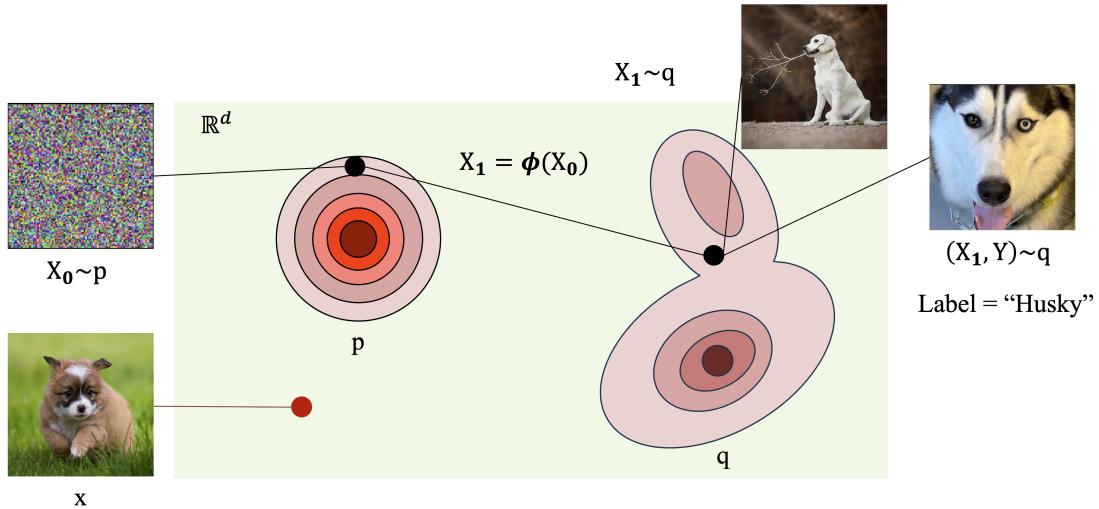
Trong phần nội dung này, chúng ta sẽ ứng dụng mô hình Conditional Flow Matching vào giải quyết bài toán Text-to-Image Generation, sau đó mở rộng áp dụng cho bài toán Text-guided Image Generation.

Conditional Flow Matching (CFM) mở rộng Flow Matching với khả năng điều kiện hóa (conditioning). Điều này cho phép mô hình tạo ra dữ liệu có điều kiện dựa trên các thuộc tính hoặc thông tin bổ sung.

Trong mô hình có điều kiện, chúng ta làm việc với:

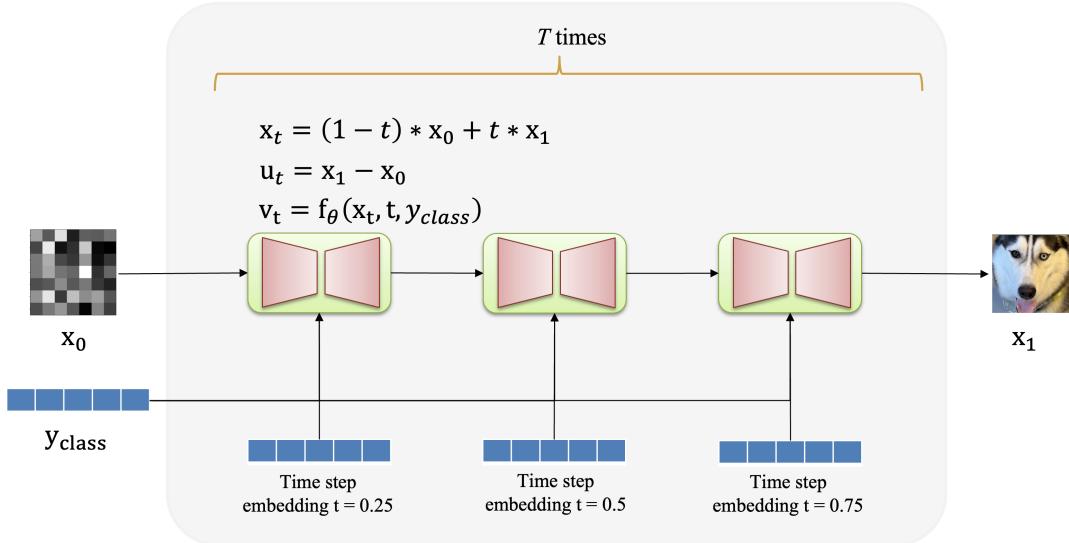
- $X_0 \sim p$: dữ liệu thực
- $X_1 \sim q$: noise
- $(X_1, Y) \sim q$: cặp noise và thông tin điều kiện

Thông tin điều kiện Y có thể là các nhãn lớp, văn bản, hoặc các dữ liệu khác giúp hướng dẫn quá trình sinh.



Hình 2: Conditional Flow Matching.

Huấn luyện Conditional Flow Matching



Hình 3: Flow Matching.

Quá trình huấn luyện cho Conditional Flow Matching tương tự như Flow Matching, nhưng bổ sung thông tin điều kiện vào mô hình học sâu để dự đoán các kết quả theo điều kiện:

- Chọn ngẫu nhiên một mẫu có điều kiện $(x_1, y_1) \sim q$
- Chọn ngẫu nhiên một mẫu dữ liệu sạch $x_0 \sim p_0$
- Chọn ngẫu nhiên một thời điểm $t \in [0, 1]$
- Tính điểm nội suy $x_t = (1 - t)x_0 + tx_1$
- Tính vận tốc thực $u_t = x_1 - x_0$

(f) Dự đoán vận tốc bằng mô hình có điều kiện $v_t = f_\theta(x_t, t, y_1)$

(g) Tính hàm mất mát $l_t = |v_t - u_t|^2$

(h) Cập nhật tham số mô hình θ bằng gradient descent

Hàm mất mát tổng quát trở thành:

$$\mathcal{L} = \mathbb{E}_{t, X_0, X_1, Y} |u_t^\theta(X_t, Y) - (X_1 - X_0)|^2$$

Lấy mẫu từ Conditional Flow Matching

Quá trình lấy mẫu cũng tương tự nhưng sử dụng thông tin điều kiện:

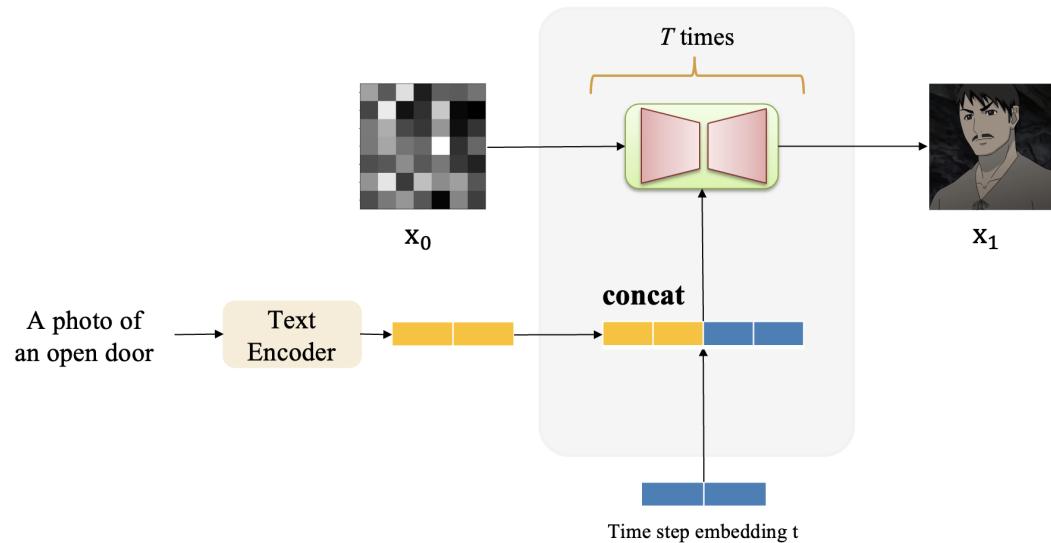
$$\frac{dy}{dt} = f_\theta(y, t, y_{class})$$

$$y(t + \Delta t) \approx y(t) + f_\theta(y, t, y_{class})\Delta t$$

Trong đó y_{class} là thông tin điều kiện (ví dụ: nhãn lớp) mà chúng ta muốn sử dụng để hướng dẫn quá trình sinh.

Phần 2. Text-to-Image using Conditional Flow Matching

Mô hình áp dụng CFM cho bài toán Text-to-Image được mô tả như hình sau:



Hình 4: Text-to-Image using CFM.

Quá trình huấn luyện mô hình như sau:

1. Dataset Preparing

Để huấn luyện mô hình, bộ dữ liệu được sử dụng cho các hình ảnh tương ứng và đoạn văn bản mô tả, bộ dữ liệu được sử dụng ở đây là: "Naruto Captions". Mô hình sentence transformers được sử dụng để mã hoá cho văn bản thành vector có 768 chiều.

```

1 # Install libs
2 !pip install -q datasets torchcfm
3
4 # Dataset
5 from datasets import load_dataset
6
7 ds = load_dataset("wanhin/naruto-captions", split="train")
8
9 # Text Encoder:
10 import torch
11 from sentence_transformers import SentenceTransformer
12
13 device = torch.device(
14     "cuda" if torch.cuda.is_available() else "cpu"
15 )
16 text_encoder = SentenceTransformer("all-mnlp-base-v2").to(device)

```

2. Preprocessing

Xây dựng bộ dữ liệu cho quá trình huấn luyện mô hình, với dữ liệu đầu vào là text embedding, và đầu ra mô hình là biểu diễn của ảnh tương ứng.

```

1  from torchvision import transforms
2
3  transform = transforms.Compose([
4      transforms.Resize((64, 64)),
5      transforms.ToTensor()
6  ])
7
8
9  from torch.utils.data import Dataset, DataLoader
10
11 class CFMDataset(Dataset):
12     def __init__(self, dataset, transform, text_encoder, device):
13         self.dataset = dataset
14         self.transform = transform
15         self.text_encoder = text_encoder
16         self.device = device
17         self.images = dataset["image"]
18         self.captions = dataset["text"]
19         self.embed_captions = text_encoder.encode(
20             self.captions, convert_to_tensor=True, device=self.device
21         )
22
23     def __len__(self):
24         return len(self.images)
25
26     def __getitem__(self, idx):
27         # get a image
28         image = self.images[idx]
29         image = self.transform(image)
30
31         # get a text
32         caption = self.captions[idx]
33         caption_embedding = self.embed_captions[idx]
34
35         return {
36             "image": image,
37             "caption": caption,
38             "caption_embedding": caption_embedding,
39         }
40
41 train_ds = CFMDataset(ds, transform, text_encoder, device)
42 train_loader = DataLoader(train_ds, batch_size=256, shuffle=True)

```

3. Model

Mô hình kết hợp biểu diễn của văn bản vào với biểu diễn của time embedding. Tinh chỉnh UNet theo hướng dẫn sau:

```

1 import math
2 import torch.nn as nn
3 from torchcfm.models.unet import UNetModel
4
5 def timestep_embedding(timesteps, dim, max_period=10000):
6     """Create sinusoidal timestep embeddings.
7
8     :param timesteps: a 1-D Tensor of N indices, one per batch element. These may
9     be fractional.
10    :param dim: the dimension of the output.
11    :param max_period: controls the minimum frequency of the embeddings.
12    :return: an [N x dim] Tensor of positional embeddings.
13    """

```

```

13     half = dim // 2
14     freqs = torch.exp(
15         -math.log(max_period)
16         * torch.arange(start=0, end=half, dtype=torch.float32, device=timesteps.
17           device)
18         / half
19     )
20     args = timesteps[:, None].float() * freqs[None]
21     embedding = torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
22     if dim % 2:
23         embedding = torch.cat([embedding, torch.zeros_like(embedding[:, :1])], dim=-1)
24     return embedding
25
26 class UNetModelWithTextEmbedding(UNetModel):
27     def __init__(self, dim, num_channels, num_res_blocks, embedding_dim, *args,
28      **kwargs):
29         super().__init__(dim, num_channels, num_res_blocks, *args, **kwargs)
30
31         self.embedding_layer = nn.Linear(embedding_dim, num_channels*4)
32         self.fc = nn.Linear(num_channels*8, num_channels*4)
33
34     def forward(self, t, x, text_embeddings=None):
35         """Apply the model to an input batch, incorporating text embeddings."""
36         timesteps = t
37
38         while timesteps.dim() > 1:
39             timesteps = timesteps[:, 0]
40         if timesteps.dim() == 0:
41             timesteps = timesteps.repeat(x.shape[0])
42
43         hs = []
44         emb = self.time_embed(timestep_embedding(timesteps, self.model_channels))
45
46         if text_embeddings is not None:
47             text_embedded = self.embedding_layer(text_embeddings)
48             emb = torch.cat([emb, text_embedded], dim=1) # 128*2
49             emb = self.fc(emb)
50
51         h = x.type(self.dtype)
52         for module in self.input_blocks:
53             h = module(h, emb)
54             hs.append(h)
55         h = self.middle_block(h, emb)
56         for module in self.output_blocks:
57             h = torch.cat([h, hs.pop()], dim=1)
58             h = module(h, emb)
59         h = h.type(x.dtype)
60         return self.out(h)

```

4. Training

Huấn luyện mô hình dựa vào CFM dựa vào code sau:

```

1 from tqdm import tqdm
2 model = UNetModelWithTextEmbedding(
3     dim=(3, 64, 64), num_channels=32, num_res_blocks=1, embedding_dim=768
4 ).to(device)
5 optimizer = torch.optim.Adam(model.parameters())
6
7 n_epochs = 20000

```

```

8
9 for epoch in tqdm(range(n_epochs)):
10    losses = []
11    for batch in train_loader:
12        optimizer.zero_grad()
13        x1 = batch["image"].to(device)
14        text_embeddings = batch["caption_embedding"].to(device)
15        x0 = torch.randn_like(x1).to(device)
16        t = torch.rand(x0.shape[0], 1, 1, 1).to(device)
17        xt = t * x1 + (1 - t) * x0
18        ut = x1 - x0
19        t = t.squeeze()
20        vt = model(t, xt, text_embeddings=text_embeddings)
21        loss = torch.mean(((vt - ut) ** 2))
22        loss.backward()
23        optimizer.step()
24        losses.append(loss.item())
25
26    avg_loss = sum(losses) / len(losses)
27    if (epoch + 1) % 500 == 0:
28        print(f"Epoch [{epoch+1}/{n_epochs}], Loss: {avg_loss:.4f}")

```

5. Inference

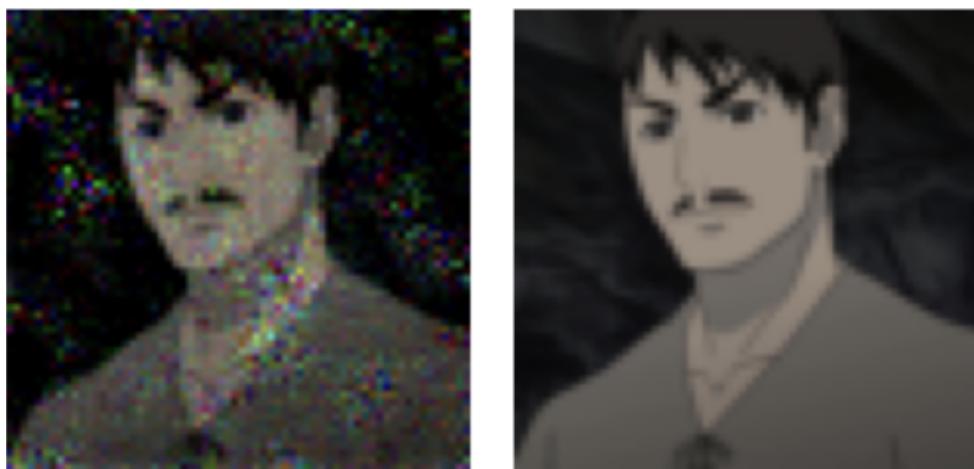
Sau khi huấn luyện, đoạn văn bản sau đây được sử dụng để giá mô hình: "A man with dark hair and brown eyes".

```

1 model.eval()
2 def euler_method(model, text_embedding, t_steps, dt, noise):
3     y = noise
4     y_values = [y]
5     with torch.no_grad():
6         for t in t_steps[1:]:
7             t = t.reshape(-1, )
8             dy = model(t.to(device), y, text_embeddings=text_embedding)
9             y = y + dy * dt
10            y_values.append(y)
11    return torch.stack(y_values)
12
13 # Initial random image and class (optional)
14 noise = torch.randn(3, 64, 64), device=device).unsqueeze(0)
15 # A man with dark hair and brown eyes
16 text_embedding = text_embeddings[1].unsqueeze(0).to(device)
17
18 # Time parameters
19 t_steps = torch.linspace(0, 1, 100, device=device)
20 dt = t_steps[1] - t_steps[0]
21
22 # Solve the ODE using Euler method
23 results = euler_method(model, text_embedding, t_steps, dt, noise)

```

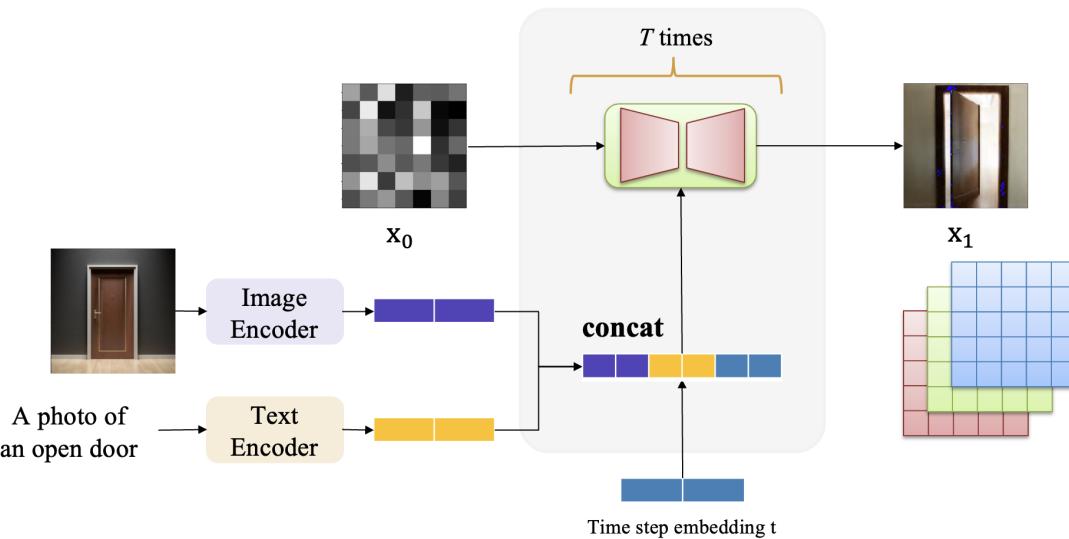
Kết quả sao sánh với ảnh thật:



Hình 5: Example of Text-to-Image Generation.

Phần 3. Text-Guided Image Generation using Conditional Flow Matching

Mô hình áp dụng CFM cho bài toán Text-Guided Image Generation được mô tả như hình sau:



Hình 6: Text-Guided Image Generation using CFM.

Quá trình huấn luyện mô hình như sau:

1. Data Preparing

Để huấn luyện mô hình, bộ dữ liệu được sử dụng cho các hình ảnh tương ứng và đoạn văn bản mô tả, bộ dữ liệu được sử dụng ở đây là: "tedbench", mỗi mẫu dữ liệu gồm: ảnh gốc, đoạn văn bản chỉnh sửa và ảnh sau chỉnh sửa. Mô hình sentence transformers được sử dụng để mã hoá cho văn bản thành vector có 768 chiều.

```

1 # Install libs
2 !pip install -q datasets torchcfm
3
4 # Dataset
5 from datasets import load_dataset
6
7 ds = load_dataset("bahjat-kawar/tedbench", split="val")
8
9 # Text Encoder
10 import torch
11 from sentence_transformers import SentenceTransformer
12
13 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
14 text_encoder = SentenceTransformer("all-mpnet-base-v2").to(device)

```

2. Preprocessing

Xây dựng bộ dữ liệu cho quá trình huấn luyện mô hình, với dữ liệu đầu vào là text embedding và hình ảnh gốc, và đầu ra mô hình là biểu diễn của ảnh chỉnh sửa tương ứng.

```

1 from torchvision import transforms
2 from torch.utils.data import Dataset, DataLoader

```

```

3
4 class TextGuidedImageGenerationDataset(Dataset):
5     def __init__(self, dataset, transform, text_encoder, device):
6         self.dataset = dataset
7         self.transform = transform
8         self.text_encoder = text_encoder
9         self.device = device
10        self.original_images = dataset["original_image"]
11        self.captions = dataset["caption"]
12        self.edited_images = dataset["edited_image"]
13        self.embed_captions = text_encoder.encode(
14            self.captions, convert_to_tensor=True, device=self.device
15        )
16
17    def __len__(self):
18        return len(self.captions)
19
20    def __getitem__(self, idx):
21        # get a image
22        original_image = self.original_images[idx]
23        original_image = self.transform(original_image)
24
25        edited_image = self.edited_images[idx]
26        edited_image = self.transform(edited_image)
27
28        # get a text
29        caption = self.captions[idx]
30        caption_embedding = self.embed_captions[idx]
31
32        return {
33            "original_image": original_image,
34            "edited_image": edited_image,
35            "caption": caption,
36            "caption_embedding": caption_embedding,
37        }
38
39    transform = transforms.Compose([
40        transforms.Resize((256, 256)),
41        transforms.ToTensor()
42    ])
43
44 train_ds = TextGuidedImageGenerationDataset(
45     ds, transform, text_encoder, device
46 )
47 train_loader = DataLoader(train_ds, batch_size=256, shuffle=True)

```

3. Model

Mô hình kết hợp biểu diễn của văn bản và hình ảnh gốc vào với biểu diễn của time embedding. Tinh chỉnh UNet theo hướng dẫn sau:

```

1 import math
2 import torch.nn as nn
3 from torchcfm.models.unet import UNetModel
4
5 def timestep_embedding(timesteps, dim, max_period=10000):
6     """Create sinusoidal timestep embeddings.
7
8     :param timesteps: a 1-D Tensor of N indices, one per batch element. These may
9     be fractional.
10    :param dim: the dimension of the output.

```

```

10     :param max_period: controls the minimum frequency of the embeddings.
11     :return: an [N x dim] Tensor of positional embeddings.
12     """
13
14     half = dim // 2
15     freqs = torch.exp(
16         -math.log(max_period)
17         * torch.arange(start=0, end=half, dtype=torch.float32, device=timesteps.
18                       device)
18         / half
19     )
20     args = timesteps[:, None].float() * freqs[None]
21     embedding = torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
22     if dim % 2:
23         embedding = torch.cat([embedding, torch.zeros_like(embedding[:, :1])],
24                               dim=-1)
25     return embedding
26
27 class UNetModelWithTextEmbedding(UNetModel):
28     def __init__(self, dim, num_channels, num_res_blocks, embedding_dim, *args,
29                  **kwargs):
30         super().__init__(dim, num_channels, num_res_blocks, *args, **kwargs)
31
32         self.image_encoder = nn.Sequential(
33             nn.Conv2d(3, num_channels, kernel_size=3, stride=2, padding=1),
34             nn.ReLU(),
35             nn.Conv2d(num_channels, num_channels*2, kernel_size=3, stride=2,
36                      padding=1),
37             nn.ReLU(),
38             nn.Conv2d(num_channels*2, num_channels*4, kernel_size=3, stride=2,
39                      padding=1),
40             nn.ReLU(),
41             nn.AdaptiveAvgPool2d(1)
42         )
43
44         self.embedding_layer = nn.Linear(embedding_dim, num_channels*4)
45         self.fc = nn.Linear(num_channels*12, num_channels*4)
46
47     def forward(self, t, x, text_embeddings=None, original_image=None):
48         """Apply the model to an input batch, incorporating text embeddings."""
49         timesteps = t
50
51         while timesteps.dim() > 1:
52             timesteps = timesteps[:, 0]
53         if timesteps.dim() == 0:
54             timesteps = timesteps.repeat(x.shape[0])
55
56         hs = []
57         emb = self.time_embed(timestep_embedding(timesteps, self.model_channels))
58
59         if (text_embeddings is not None) and (original_image is not None):
60             text_embedded = self.embedding_layer(text_embeddings)
61             image_embedded = self.image_encoder(original_image).squeeze(2, 3)
62             emb = torch.cat([emb, text_embedded, image_embedded], dim=1)
63             emb = self.fc(emb)
64
64         h = x.type(self.dtype)
65         for module in self.input_blocks:
66             h = module(h, emb)
67             hs.append(h)
68         h = self.middle_block(h, emb)

```

```

65         for module in self.output_blocks:
66             h = torch.cat([h, hs.pop()], dim=1)
67             h = module(h, emb)
68             h = h.type(x.dtype)
69     return self.out(h)

```

4. Training

Huấn luyện mô hình dựa vào CFM dựa vào code sau:

```

1 from tqdm import tqdm
2
3 model = UNetModelWithTextEmbedding(
4     dim=(3, 256, 256), num_channels=32, num_res_blocks=1, embedding_dim=768
5 ).to(device)
6 optimizer = torch.optim.Adam(model.parameters())
7
8 n_epochs = 5000
9 for epoch in tqdm(range(n_epochs)):
10     losses = []
11     for batch in train_loader:
12         original_image = sample["original_image"].to(device)
13         edited_image = sample["edited_image"].to(device)
14         caption_embedding = sample["caption_embedding"].to(device)
15         optimizer.zero_grad()
16         x1 = edited_image
17         x0 = torch.randn_like(x1).to(device)
18         t = torch.rand(x0.shape[0], 1, 1, 1).to(device)
19         xt = t * x1 + (1 - t) * x0
20         ut = x1 - x0
21         t = t.squeeze()
22         vt = model(t, xt, text_embeddings=caption_embedding, original_image=
23                     original_image)
24         loss = torch.mean(((vt - ut) ** 2))
25         loss.backward()
26         optimizer.step()
27
28     avg_loss = sum(losses)/len(losses)
29     if (epoch + 1) % 500 == 0:
30         print(f"Epoch [{epoch+1}/{n_epochs}], Loss: {avg_loss:.4f}")

```

5. Inference

Sau khi huấn luyện, đoạn văn bản sau đây được sử dụng để giá mô hình: "A photo of an open door" và hình ảnh gốc từ mẫu dữ liệu thứ 5 trong dữ liệu gốc.

```

1
2 model.eval()
3 def euler_method(model, text_embedding, t_steps, dt, noise, original_image):
4     y = noise
5     y_values = [y]
6     with torch.no_grad():
7         for t in t_steps[1:]:
8             t = t.reshape(-1, )
9             dy = model(t.to(device), y, text_embeddings=text_embedding,
10             original_image=original_image)
11             y = y + dy * dt
12             y_values.append(y)
13     return torch.stack(y_values)
14
# Initial random image and class (optional)

```

```

15 # A photo of an open door.
16 sample = train_ds[5]
17 original_image = sample["original_image"].unsqueeze(0).to(device)
18 edited_image = sample["edited_image"].unsqueeze(0).to(device)
19 caption_embedding = sample["caption_embedding"].unsqueeze(0).to(device)
20 noise = torch.randn_like(original_image, device=device)
21 text_embedding = caption_embedding
22
23 # Time parameters
24 t_steps = torch.linspace(0, 1, 150, device=device)
25 dt = t_steps[1] - t_steps[0]
26
27 # Solve the ODE using Euler method
28 results = euler_method(model, text_embedding, t_steps, dt, noise, original_image)

```

6. Deployment

Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

Text-Guided Image Generation using Conditional Flow Matching

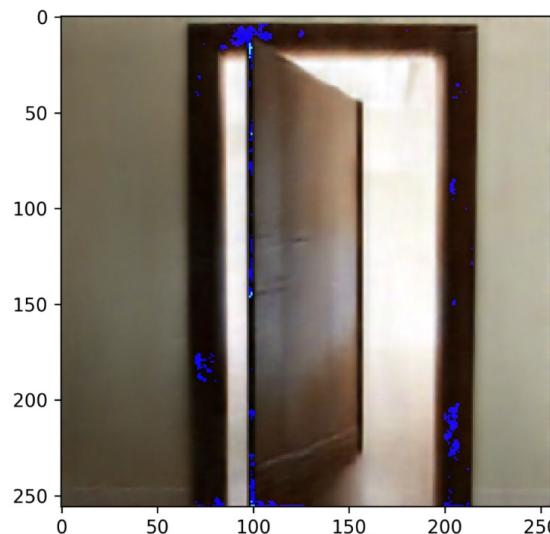
Model: Conditional Flow Matching. Dataset: Tedbench

Instruction:

A photo of an open door.

How would you like to give the input?

Run Example Image



Hình 7: Text-Guided Image Generation deployment on Streamlit.

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Phương pháp nào sau đây không được sử dụng trong Text-to-Image Generation?

- a) Flow Matching
- b) VAE
- c) K-means clustering
- d) Diffusion Models

Câu hỏi 2 Phương pháp nào được sử dụng trong Text-to-Image để xử lý và mã hóa văn bản hiệu quả nhất?

- a) BERT
- b) LSTM
- c) CNN
- d) VAE

Câu hỏi 3 Công thức nào sau đây mô tả sự thay đổi giữa dữ liệu nhiễu và dữ liệu sạch trong Flow Matching?

- a) $v_t = x_1 - x_0$
- b) $v_t = x_1 + x_0$
- c) $v_t = x_1/x_0$
- d) $v_t = \sigma(t) + x_0$

Câu hỏi 4 Mỗi văn bản, đoạn text sẽ được biểu diễn thành vector bao nhiêu chiều?

- a) 512
- b) 768
- c) 256
- d) 128

Câu hỏi 5 Đầu vào của mô hình UNet trong mô hình Text-to-Image Generation là bao nhiêu?

- a) 2
- b) 3
- c) 4
- d) 5

Câu hỏi 6 Đầu vào của mô hình UNet trong mô hình Text-Guided Image Generation là bao nhiêu?

- a) 2
- b) 3
- c) 4
- d) 5

Câu hỏi 7 Kích thước ảnh áp dụng cho mô hình Text-Guided Image Generation là bao nhiêu?

- a) 256x256
- b) 128x128
- c) 64x64
- d) 32x32

Câu hỏi 8 Phương pháp biểu diễn cho time trong phần mô hình UNet là?

- a) RoPE

- b) Hàm sin cos
- c) ALiBi
- d) nn.Embedding

Câu hỏi 9 Mô hình nào phù hợp để thay thế quá trình biểu diễn cho cả văn bản và hình ảnh gốc trong bài toán Text-Guided Image Generation?

- a) BERTs
- b) LSTM
- c) CLIP
- d) ResNet

Câu hỏi 10 Công thức nào đúng trong phần code của Conditional Flow matching?

- a) $y = y + dy/dt$
- b) $y = y + dy * dt$
- c) $y = y + dy + dt$
- d) $y = y + dy - dt$

Phần 5. Phụ lục

1. **Hint:** Dựa vào file tải về **Text-Guided-Image-Generation-Conditional-Flow-Matching** để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Hiểu rõ bài toán Text-to-Image Generation và Text-Guided Image Generation - Hiểu rõ mô hình Conditional Flow Matching 	<ul style="list-style-type: none"> - Các bước thực hiện trong mô hình với Flow Matching và Conditional Flow Matching áp dụng cho bài toán trên.
2.	<ul style="list-style-type: none"> - Hiểu rõ mô hình UNet sử dụng trong Conditional Flow Matching - Các kỹ thuật cải tiến mô hình UNet và tinh chỉnh phù hợp với bài toán có nhiều kiểu dữ liệu 	<ul style="list-style-type: none"> - Xây dựng mô hình Text-to-Image Generation và Text-Guided Image Generation - Đánh giá mô hình Conditional Flow Matching với các mô hình khác như Diffusion hoặc GANs.

- Kết -