Domain-Specialized Text-Guided Image Editing: Improving Edit Accuracy via Targeted Diffusion Fine-Tuning (Fashion Domain)

CS5787 - Deep Learning @ Cornell Tech

Authors:

- Sona Krishnan (sk3449)
- Vishali Vallioor (vv266)
- David Thai (dt573)

Code Structure

1. Imports and Background
2. Data Preparation
3. Model Baseline
4. Fine-Tuning
5. Evaluation

This code goes through our code structure and background. Make sure to run all the necessary imports. An important note is that you must be in a GPU environment, like Google Colab GPU T4 100 (which is what we ran this on).

1. Imports and Background

```
# Run this cell for imports and necessary installs

import os
import json
import torch
from torch import nn
from torch.utils.data import DataLoader
import torch.nn.functional as F
import numpy as np
import PIL
from PIL import Image
from collections import defaultdict
import matplotlib.pyplot as plt
from tqdm import tqdm
import os
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import torchvision.transforms.functional as TF
from tqdm import tqdm
from PIL import Image
import gc
from diffusers import StableDiffusionInstructPix2PixPipeline, DDPMScheduler
from torch.optim.lr_scheduler import CosineAnnealingLR
from torchvision.transforms.functional import to_pil_image
from torch.optim import AdamW
from tqdm.auto import tqdm
import pandas as pd
import requests
from io import BytesIO
import matplotlib.pyplot as plt
import random

# models and dataset
from datasets import load_dataset
from diffusers import StableDiffusionInstructPix2PixPipeline, DPMSolverMultistepScheduler
from transformers import CLIPProcessor, CLIPModel
```

```
from datasets import load_dataset

# cloning pix2pix, getting it in

!git clone https://github.com/huggingface/diffusers

%cd examples/instruct_pix2pix
!pip install -r requirements.txt

# from pix2pix hf website

import torch, requests, PIL
from diffusers import StableDiffusionInstructPix2PixPipeline, EulerAncestralDiscreteScheduler
from accelerate.utils import write_basic_config
write_basic_config()
```

```
model_id = "timbrooks/instruct-pix2pix"
pipe = StableDiffusionInstructPix2PixPipeline.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    safety_checker=None,
)
pipe.to("cuda")

pipe.scheduler = DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)

url = "https://thumbs.dreamstime.com/b/fashion-model-poses-elegantly-modern-studio-showcasing-cont

def download_image(url):
    image = PIL.Image.open(requests.get(url, stream=True).raw)
    image = PIL.ImageOps.exif_transpose(image)
    return image.convert("RGB")

image = download_image(url)

prompt = "make her hair red"

images = pipe(
    prompt=prompt,
    image=image,
    num_inference_steps=100,
    guidance_scale=12,
    image_guidance_scale=2.0,
).images

images[0]
```

Show hidden output

```
url = "https://thumbs.dreamstime.com/b/fashion-model-poses-elegantly-modern-studio-showcasing-conte

def download_image(url):
    image = PIL.Image.open(requests.get(url, stream=True).raw)
    image = PIL.ImageOps.exif_transpose(image)
    return image.convert("RGB")

image = download_image(url)

prompt = "give her shirt a floral pattern"

images = pipe(
    prompt=prompt,
    image=image,
    num_inference_steps=100,
    guidance_scale=12,
    image_guidance_scale=2.0,
).images

images[0]
```

```
100%  100/100 [01:03<00:00,  1.54it/s]
```

2. Data Preparation

In the next following lines, we load the fashionpedia/deep fashion dataset we will be using for training. However, the important thing to note is that the DeepFashion dataset is used for training but the FashionPedia is used for validation at the end.

2.1 Load FashionPedia Dataset

```
#Loading fashionpedia
dataset_fashionpedia = load_dataset("detection-datasets/fashionpedia", split="train", streaming=Tr
fashionpedia_samples = list(dataset_fashionpedia.take(100)) #just load 100 for now to see if poss
print(fashionpedia_samples[0]['image'])


clip_model = CLIPModel.from_pretrained("openai/clip-vit-large-patch14").to("cuda")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-large-patch14")
```

Show hidden output

2.2 DeepFashion2 Dataset

Now loading in the deep fashion 2 dataset.

```
#load deepfashion2
dataset = load_dataset("SaffalPoosh/deepFashion-with-masks")

print(dataset)
print(dataset["train"].features)

sample = dataset["train"][0]
print(sample.keys())

```

Show hidden output

2.3 Generating Triplets for fine-tune the model with (conditioned image, text, output image)

Now the next following blocks of code involves generating the triplets.

```
by_pid = defaultdict(list)
for i in range(len(dataset["train"])):
  by_pid[dataset["train"][i]["pid"]].append(i)

multi_outfit = {k: v for k, v in by_pid.items() if len(v) >= 2}
print(f"Total unique people: {len(by_pid)}")
print(f"People with 2+ images: {len(multi_outfit)}")
print(f"Potential pairs: {sum(len(v) for v in multi_outfit.values())}")
```

```
Total unique people: 7557
People with 2+ images: 6900
Potential pairs: 40001
```

```
multi_outfit_pids = [pid for pid, indices in by_pid.items() if len(indices) >= 3]

fig, axes = plt.subplots(3, 3, figsize=(12, 12))
for row, pid in enumerate(multi_outfit_pids[:3]):
  indices = by_pid[pid][:3]
  for col, idx in enumerate(indices):
    sample = dataset["train"][idx]
    axes[row, col].imshow(sample["images"])
    axes[row, col].set_title(sample["caption"], fontsize=8)
    axes[row, col].axis("off")

plt.suptitle("same-person pairs from DeepFashion2", fontsize=14)
plt.tight_layout()
plt.show()

```

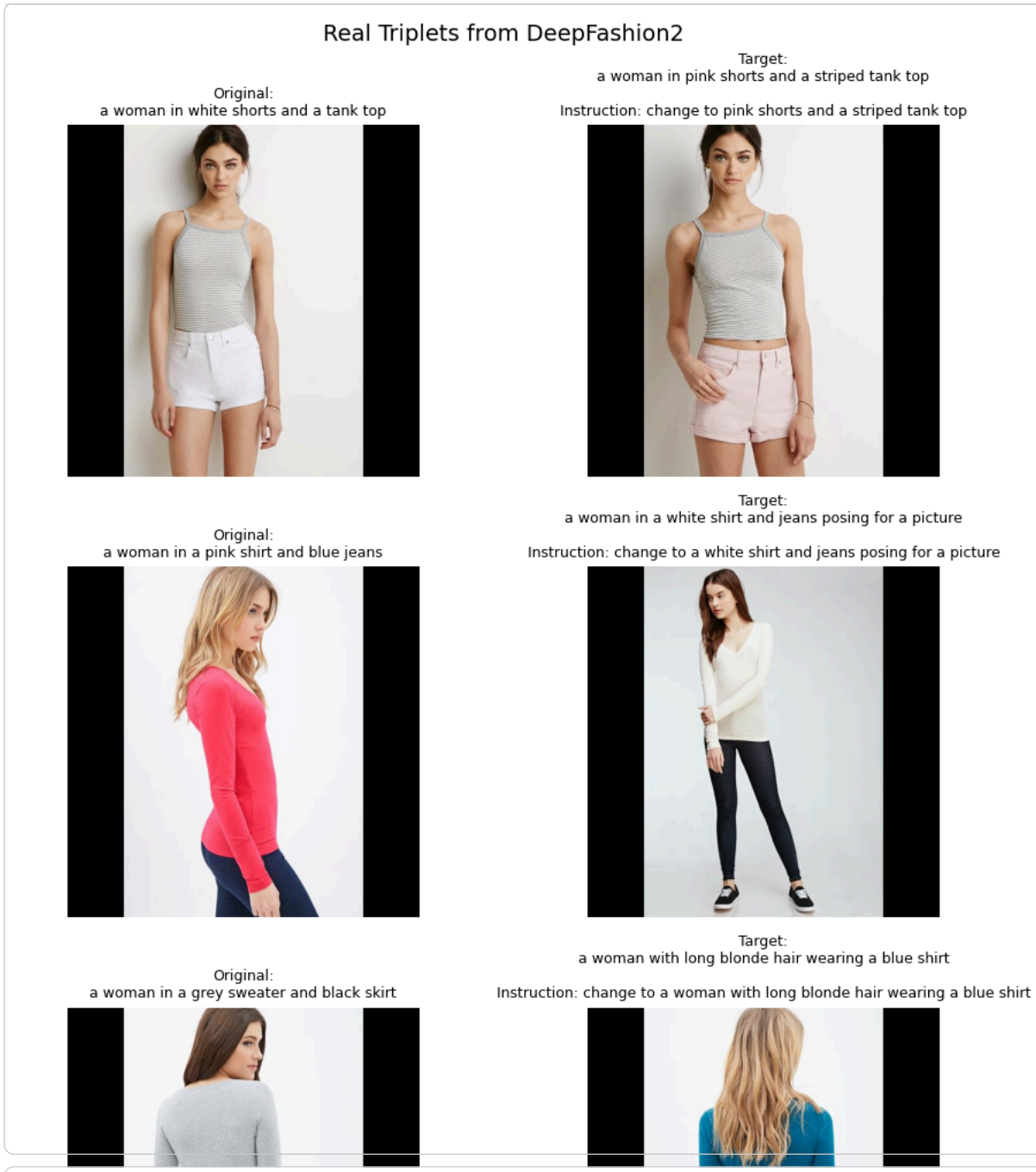
2.4 Generating Instructions

```
def generate_instruction(caption1, caption2):
  part2 = caption2.lower().split("in ")[-1] if "in " in caption2.lower() else caption2.lower()
  return f"change to {part2}"

MAX_TRIPLETS = 8000

triplets = []
for pid, indices in tqdm(by_pid.items(), desc="Building triplets"):
```

```
  if len(indices) < 2:
      continue

  for i in range(len(indices)):
      for j in range(len(indices)):
          if i == j:
              continue

          triplets.append({
                  "src_idx": indices[i],
                  "tgt_idx": indices[j],
              })

          if len(triplets) >= MAX_TRIPLETS:
              break
      if len(triplets) >= MAX_TRIPLETS:
          break
  if len(triplets) >= MAX_TRIPLETS:
      break

print(f"Total triplets: {len(triplets)}")
print(f"Sample: {triplets[0]}")

```

```
Building triplets:   0%|          | 17/7557 [00:00<00:01, 4432.07it/s]Total triplets: 8000
Sample: {'src_idx': 0, 'tgt_idx': 11800}
```

```
fig, axes = plt.subplots(3, 2, figsize=(10, 12))

for row in range(3):
    t = triplets[row * 100]

    src = dataset["train"][t["src_idx"]]
    tgt = dataset["train"][t["tgt_idx"]]
    instruction = generate_instruction(src["caption"], tgt["caption"])

    axes[row, 0].imshow(src["images"])
    axes[row, 0].set_title(f"Original:\n{src['caption']}", fontsize=9)
    axes[row, 0].axis("off")

    axes[row, 1].imshow(tgt["images"])
    axes[row, 1].set_title(f"Target:\n{tgt['caption']}\n\nInstruction: {instruction}", fontsize=9)
    axes[row, 1].axis("off")

plt.suptitle("Real Triplets from DeepFashion2", fontsize=14)
plt.tight_layout()
plt.show()
```




```
CACHE_DIR = "deepfashion_cache"
os.makedirs(CACHE_DIR, exist_ok=True)

# get the unique indices we need
all_indices = set()
for t in triplets:
  all_indices.add(t["src_idx"])
  all_indices.add(t["tgt_idx"])

for idx in tqdm(all_indices, desc="caching the images"):
  cache_path = f"{CACHE_DIR}/{idx}.png"
  if not os.path.exists(cache_path):
      img = dataset["train"][idx]["images"]
      img.save(cache_path)

```

```
Caching 258 unique images...
Caching images: 100%|██████████| 258/258 [00:03<00:00, 68.09it/s]Caching complete!
```

```
class TripletDataset(Dataset):
    def __init__(self, triplet_list, hf_data, size=512):
        self.data = triplet_list
        self.hf = hf_data
        self.size = size
        self.transform = transforms.Compose([
            transforms.Resize((size, size), interpolation=transforms.InterpolationMode.BILINEAR),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
        ])

        self.mask_transform = transforms.Compose([
            transforms.Resize((size, size), interpolation=transforms.InterpolationMode.NEAREST),
            transforms.ToTensor(),
        ])
        self.color_jitter = transforms.ColorJitter(
            brightness=0.1,
            contrast=0.1,
            saturation=0.1,
            hue=0.05,
        )

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        t = self.data[idx]

        src = self.hf["train"][t["src_idx"]]
        tgt = self.hf["train"][t["tgt_idx"]]

        orig_img = src["images"]
        edit_img = tgt["images"]
        instruction = generate_instruction(src["caption"], tgt["caption"])
        src_mask = src["mask"]

        if orig_img.mode != "RGB":
            orig_img = orig_img.convert("RGB")
        if edit_img.mode != "RGB":
            edit_img = edit_img.convert("RGB")
        if src_mask.mode != "L":
          src_mask = src_mask.convert("L")

        if torch.rand(1).item() < 0.5:
            orig_img = TF.hflip(orig_img)
            edit_img = TF.hflip(edit_img)
            src_mask = TF.hflip(src_mask)

        orig_img = self.color_jitter(orig_img)
        edit_img = self.color_jitter(edit_img)

        conditioning_image = self.transform(orig_img)
        pixel_values = self.transform(edit_img)
        mask_tensor = self.mask_transform(src_mask)
        mask_tensor = (mask_tensor > 0.5).float()

        return {
            "conditioning_image": conditioning_image,
            "pixel_values": pixel_values,
            "prompt": instruction,
            "mask": mask_tensor,
        }

full_ds = TripletDataset(triplets, dataset, size=512)

train_size = int(len(full_ds) * 0.8)
val_size = int(len(full_ds) * 0.1)
test_size = len(full_ds) - train_size - val_size

train_ds, val_ds, test_ds = torch.utils.data.random_split(
```

```
    full_ds,
    [train_size, val_size, test_size],
    generator=torch.Generator().manual_seed(42)
)

BATCH_SIZE = 1
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True, num_workers=0)
val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False, num_workers=0)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False, num_workers=0)

print(f"train: {len(train_ds)}, Val: {len(val_ds)}, Test: {len(test_ds)}")

test_batch = next(iter(train_loader))
print(f"Conditioning shape: {test_batch['conditioning_image'].shape}")
print(f"Sample prompt: {test_batch['prompt'][0]}")
```

```
Train: 6400, Val: 800, Test: 800
Conditioning shape: torch.Size([1, 3, 512, 512])
Sample prompt: change to a man wearing a blue shirt and black pants
DataLoader working!
```

3. Model Preparation and Baseline

3.1 Calculating Clip Score and Directional Similarity Scores (Evaluation Metrics)

The upcoming code cell now includes a variety of different functions used for metrics.

```
#Define evaluation metrics

#clip score, which tells us how similar text is to image.
def calculate_clip_score(image, text, clip_model, clip_processor):
    """Author: David Thai -- function to calculate the clip score."""
    inputs = clip_processor(
        text=[text],
        images=image,
        return_tensors="pt",
        padding=True
    ).to("cuda")
    with torch.no_grad(): #the following is cosine similarity
        outputs = clip_model(**inputs)
        image_embeds = outputs.image_embeds / outputs.image_embeds.norm(dim=-1, keepdim=True)
        text_embeds = outputs.text_embeds / outputs.text_embeds.norm(dim=-1, keepdim=True)
        clip_score = (image_embeds @ text_embeds.T).item()
    return clip_score * 100



#directional simularity
def calculate_clip_directional_similarity(original_img, edited_img, text, clip_model, clip_proces
   """Author: David Thai -- function to calculate the clip directional similarity"""
    with torch.no_grad():
        #original embeddings
        inputs_orig = clip_processor(images=original_img, return_tensors="pt").to("cuda")
        orig_embeds = clip_model.get_image_features(**inputs_orig)
        orig_embeds = orig_embeds / orig_embeds.norm(dim=-1, keepdim=True)
        #edit embeddings
        inputs_edit = clip_processor(images=edited_img, return_tensors="pt").to("cuda")
        edit_embeds = clip_model.get_image_features(**inputs_edit)
        edit_embeds = edit_embeds / edit_embeds.norm(dim=-1, keepdim=True)
        #text embeddings
        inputs_text = clip_processor(text=[text], return_tensors="pt", padding=True).to("cuda")
        text_embeds = clip_model.get_text_features(**inputs_text)
        text_embeds = text_embeds / text_embeds.norm(dim=-1, keepdim=True)
        #calculate direction
        img_direction = edit_embeds - orig_embeds
        img_direction = img_direction / (img_direction.norm(dim=-1, keepdim=True) + 1e-8)
```

```
        #neutral starting point
        null_text = clip_processor(text=[""], return_tensors="pt", padding=True).to("cuda")
        null_embeds = clip_model.get_text_features(**null_text)
        null_embeds = null_embeds / null_embeds.norm(dim=-1, keepdim=True)
        #determine direction
        text_direction = text_embeds - null_embeds
        text_direction = text_direction / (text_direction.norm(dim=-1, keepdim=True) + 1e-8)
        #calculate cosine similarity
        directional_sim = (img_direction @ text_direction.T).item()
    return directional_sim * 100

#LPIPS Score

lpips_model = lpips.LPIPS(net='alex').to("cuda")

lpips_transform = T.Compose([
    T.Resize((512, 512)),
    T.ToTensor(),
    T.Normalize(mean=[0.5, 0.5, 0.5],
                std=[0.5, 0.5, 0.5]),
])

def calculate_lpips(img1_pil, img2_pil):
    t1 = lpips_transform(img1_pil).unsqueeze(0).to("cuda")
    t2 = lpips_transform(img2_pil).unsqueeze(0).to("cuda")
    with torch.no_grad():
        score = lpips_model(t1, t2).item()
    return score
```

3.2 Evaluate Model

Now we prepare the model and use this baseline for InstructPix2Pix on fashionpedia, just to see how it performs as a baseline.

```
# Evaluating a baseline for InstructPix2Pix on fashionpedia

NUM_SAMPLES = 3  # num of images to test
NUM_INFERENCE_STEPS =50   # inference steps -- means lower = faster, higher = better quality
GUIDANCE_SCALE = 7.5
IMAGE_GUIDANCE_SCALE = 1.5

# this is the output directiory where the eval results are saved.
os.makedirs('evaluation_results', exist_ok=True)
os.makedirs('evaluation_results/images', exist_ok=True)

results = []
all_edits = []

for idx in range(3):
    sample = fashionpedia_samples[idx]
    original_image = sample['image']

    prompts = ['change the clothes this person is wearing to blue',
            'make the clothes this person is wearing red',
            'add stripes to the clothes',
            'change the clothes to a floral pattern']

    if max(original_image.size) > 1024:
      original_image.thumbnail((1024, 1024), Image.Resampling.LANCZOS)

    fig, ax = plt.subplots(1, 1, figsize=(8, 8))
    ax.imshow(original_image)
    ax.set_title(f'Original Image (Sample {idx + 1})', fontsize=14, fontweight='bold')
    ax.axis('off')
    plt.tight_layout()
    plt.show()

    for prompt_idx, text_prompt in enumerate(prompts):
      try:
```

```
        edited_images = pipe(
            prompt=text_prompt,
            image=original_image,
            num_inference_steps=NUM_INFERENCE_STEPS,
            guidance_scale=GUIDANCE_SCALE,
            image_guidance_scale=IMAGE_GUIDANCE_SCALE,
        ).images
        edited_image = edited_images[0]

        clip_score_original = calculate_clip_score(original_image, text_prompt, clip_model, clip
        clip_score_edited = calculate_clip_score(edited_image, text_prompt, clip_model, clip_pr
        clip_directional = calculate_clip_directional_similarity(
            original_image, edited_image, text_prompt, clip_model, clip_processor
        )

        fig, axes = plt.subplots(1, 2, figsize=(16, 8))

        axes[0].imshow(original_image)
        axes[0].set_title(f'Original CLIP Score: {clip_score_original:.2f}',
                          fontsize=12, fontweight='bold')
        axes[0].axis('off')

        axes[1].imshow(edited_image)
        axes[1].set_title(f'Edited LIP Score: {clip_score_edited:.2f} (Δ: {clip_score_edited - c
                          fontsize=12, fontweight='bold')
        axes[1].axis('off')

        metrics_text = f'Value of directional Similarity: {clip_directional:.2f}'
        plt.figtext(0.5, 0.02, metrics_text, ha='center', fontsize=11, bbox=dict(boxstyle='round

        plt.suptitle(f'Prompt: "{text_prompt}"', fontsize=13, fontweight='bold', style='italic')
        plt.tight_layout()
        plt.subplots_adjust(bottom=0.08)
        plt.show()

        result = {
            'sample_id': f"{idx}_{prompt_idx}",
            'sample_num': idx + 1,
            'edit_num': prompt_idx + 1,
            'prompt': text_prompt,
            'clip_score_original': clip_score_original,
            'clip_score_edited': clip_score_edited,
            'clip_score_improvement': clip_score_edited - clip_score_original,
            'clip_directional_similarity': clip_directional,
        }
        results.append(result)

        all_edits.append({
            'original': original_image,
            'edited': edited_image,
            'prompt': text_prompt,
            'metrics': result
        })

        print(f"   CLIP Score: {clip_score_edited:.2f} | improvement: {clip_score_edited - clip_

    except Exception as e:
        print(f"   Error: {e}")
        continue

```

Original Image (Sample 1)

100% 50/50 [00:35<00:00, 1.39it/s]

Prompt: "change the clothes this person is wearing to blue"

Original
CLIP Score: 18.47

Edited
CLIP Score: 18.16 (Δ: -0.31)

Directional Similarity: 3.76

✅ CLIP Score: 18.16 | Improvement: -0.31 | Directional Sim: 3.76

100% 50/50 [00:36<00:00, 1.36it/s]

Prompt: "make the clothes this person is wearing red"

Original
CLIP Score: 19.26

Edited
CLIP Score: 24.88 (Δ: +5.62)

Directional Similarity: 16.51

✅ CLIP Score: 24.88 | Improvement: +5.62 | Directional Sim: 16.51

100% 50/50 [00:36<00:00, 1.35it/s]

Prompt: "add stripes to the clothes"

Original
CLIP Score: 17.42

Edited
CLIP Score: 23.02 (Δ: +5.60)

Directional Similarity: 13.38

✅ CLIP Score: 23.02 | Improvement: +5.60 | Directional Sim: 13.38

100% 50/50 [00:36<00:00, 1.33it/s]

Prompt: "change the clothes to a floral pattern"

Original
CLIP Score: 18.83

Edited
CLIP Score: 20.45 (Δ: +1.62)

Directional Similarity: 8.30

✅ CLIP Score: 20.45 | Improvement: +1.62 | Directional Sim: 8.30

Original Image (Sample 2)

100% 50/50 [00:37<00:00, 1.33it/s]

Prompt: "change the clothes this person is wearing to blue"

Original
CLIP Score: 22.59

Edited
CLIP Score: 26.26 (Δ: +3.67)

Directional Similarity: 3.15

✅ CLIP Score: 26.26 | Improvement: +3.67 | Directional Sim: 3.15

100% 50/50 [00:37<00:00, 1.32it/s]

Prompt: "make the clothes this person is wearing red"

Original
CLIP Score: 24.07

Edited
CLIP Score: 30.63 (Δ: +6.56)

Directional Similarity: 9.25

✅ CLIP Score: 30.63 | Improvement: +6.56 | Directional Sim: 9.25

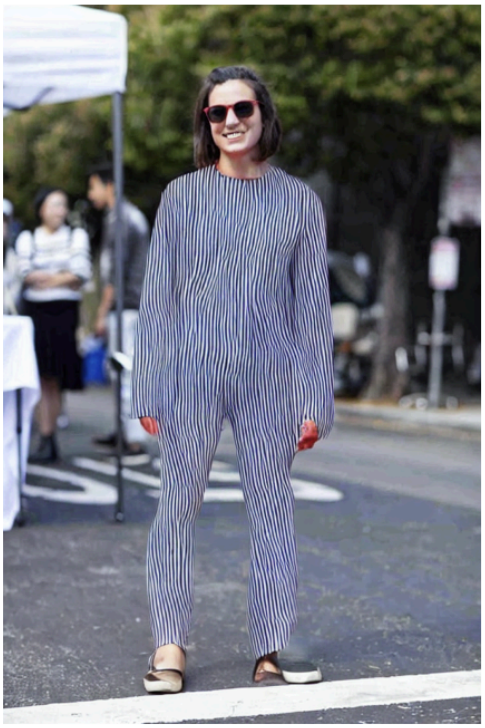
100% 50/50 [00:37<00:00, 1.32it/s]

Prompt: "add stripes to the clothes"

Prompt: "add stripes to the clothes"

Original
CLIP Score: 18.50

Edited
CLIP Score: 21.07 (Δ: +2.57)



Directional Similarity: 6.83

✅ `CLIP Score: 21.07 | Improvement: +2.57 | Directional Sim: 6.83`

100% 50/50 [00:37<00:00, 1.30it/s]

Prompt: "change the clothes to a floral pattern"

Original
CLIP Score: 15.52

Edited
CLIP Score: 24.86 (Δ: +9.34)

Directional Similarity: 12.73

✅ `CLIP Score: 24.86 | Improvement: +9.34 | Directional Sim: 12.73`

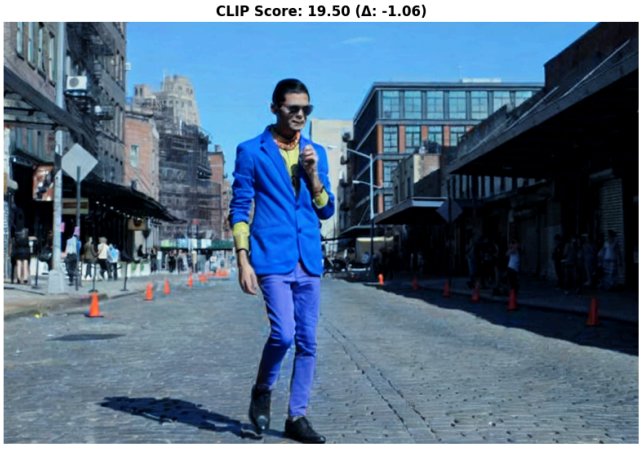
Original Image (Sample 3)

100% 50/50 [00:37<00:00, 1.32it/s]

Prompt: "change the clothes this person is wearing to blue"

Original
CLIP Score: 20.56

Edited
CLIP Score: 19.50 (Δ: -1.06)



Directional Similarity: -3.27

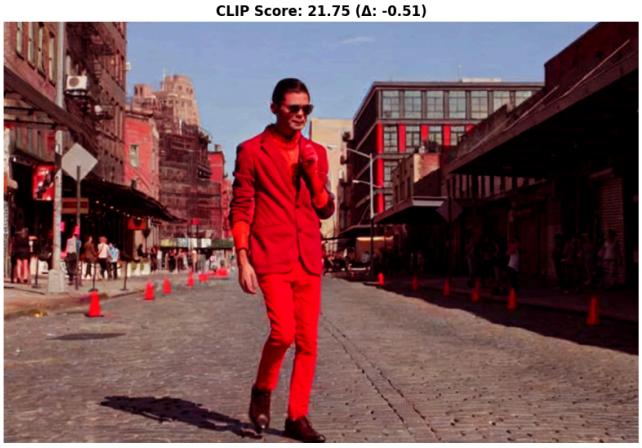
✅ `CLIP Score: 19.50 | Improvement: −1.06 | Directional Sim: −3.27`

100% 50/50 [00:37<00:00, 1.30it/s]

Prompt: "make the clothes this person is wearing red"

Original
CLIP Score: 22.26

Edited
CLIP Score: 21.75 (Δ: -0.51)



Directional Similarity: -0.38

✅ CLIP Score: 21.75 | Improvement: −0.51 | Directional Sim: −0.38

100% 50/50 [00:38<00:00, 1.31it/s]

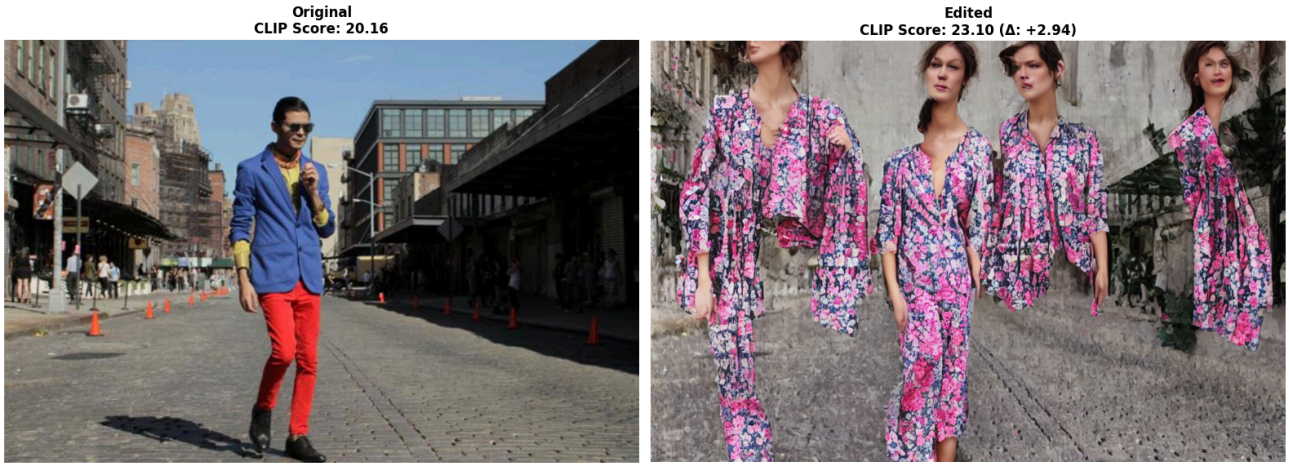
Prompt: "add stripes to the clothes"

Directional Similarity: 6.34

✅ CLIP Score: 20.05 | Improvement: +1.63 | Directional Sim: 6.34

100% 50/50 [00:37<00:00, 1.30it/s]

Prompt: "change the clothes to a floral pattern"



Directional Similarity: 7.16

✅ CLIP Score: 23.10 | Improvement: +2.94 | Directional Sim: 7.16

```
lpips_value = calculate_lpips(original_image, edited_image)
print(f"LPIPS (baseline): {lpips_value:.4f}")
```

```
num_edits = len(all_edits)
if num_edits == 0:
    print("Issue: no edits found in all_edits.")
else:
    prompts_per_sample = 4
    num_samples = num_edits // prompts_per_sample

    fig, axes = plt.subplots(num_samples, prompts_per_sample, figsize=(14, 3 * num_samples))

    if num_samples == 1:
        axes = [axes]

    for idx, edit in enumerate(all_edits):
        row = idx // prompts_per_sample
        col = idx % prompts_per_sample

        ax = axes[row][col]

        orig = edit["original"]
        edited = edit["edited"]

        small_orig = orig.copy()
        small_edit = edited.copy()
        small_orig.thumbnail((256, 256))
        small_edit.thumbnail((256, 256))

        combined = np.hstack((small_orig, small_edit))

        ax.imshow(combined)
        ax.set_title(edit["prompt"], fontsize=9)
        ax.axis("off")

    plt.tight_layout()
    plt.show()

```

4. Fine Tuning

We first start with parameter efficient fine tuning:

```
gc.collect()
torch.cuda.empty_cache()

pipe = StableDiffusionInstructPix2PixPipeline.from_pretrained(
    "timbrooks/instruct-pix2pix",
    torch_dtype=torch.float16,
    safety_checker=None,
)
pipe.to("cuda")

pipe.scheduler = DDPMScheduler.from_config(pipe.scheduler.config)

unet = pipe.unet
vae = pipe.vae
tokenizer = pipe.tokenizer
text_encoder = pipe.text_encoder
noise_scheduler = pipe.scheduler
device = "cuda"

for param in unet.parameters():
    param.requires_grad = False

for name, param in unet.named_parameters():
```

```
    should_unfreeze = False

    if "attn2" in name and ("to_k" in name or "to_v" in name or "to_out" in name):
        should_unfreeze = True

    if "attn1" in name and ("to_k" in name or "to_v" in name or "to_q" in name):
        should_unfreeze = True

    if "mid_block" in name:
        should_unfreeze = True

    if "up_blocks.2" in name or "up_blocks.3" in name:
        if "resnets" in name and "conv" in name:
            should_unfreeze = True

    if should_unfreeze:
        param.requires_grad = True

trainable_params = [p for p in unet.parameters() if p.requires_grad]

optimizer = AdamW(trainable_params, lr=1e-5, eps=1e-4, weight_decay=0.01)

print(f"Trainable params: {sum(p.numel() for p in trainable_params):,}")
print(f"Total params: {sum(p.numel() for p in unet.parameters()):,}")
print(f"Trainable %: {100 * sum(p.numel() for p in trainable_params) / sum(p.numel() for p in unet
print("\nUnfrozen layers:")
for name, param in unet.named_parameters():
    if param.requires_grad:
        print(f"  {name}: {param.numel():,}")
```

Show hidden output

4.1 Training Loop for Fine-Tuning

```
def encode_prompt(texts):
    tokens = tokenizer(
        list(texts),
        padding="max_length",
        max_length=tokenizer.model_max_length,
        truncation=True,
        return_tensors="pt",
    ).input_ids.to(device)
    with torch.no_grad():
        text_embeds = text_encoder(tokens)[0]
    return text_embeds

def encode_vae_images(images):
    images = images.to(device, dtype=torch.float16)
    with torch.no_grad():
        latents = vae.encode(images).latent_dist.sample()
    return latents * vae.config.scaling_factor

@torch.no_grad()
def validate(val_loader, num_samples=50):
  unet.eval()

  val_losses = []
  clip_scores = []
  dir_sims = []

  samples_processed = 0

  for batch in val_loader:
      if samples_processed >= num_samples:
          break

      pixel_values = batch["pixel_values"].to(device, dtype=torch.float16)
      cond_images = batch["conditioning_image"].to(device, dtype=torch.float16)
```