Introduction

In this note book, we will show how to create a a photorealistic meeting of one from different times. I am going to generate a short video clip of me and me in the future meeting with each other.

My response to the homework questions

Q) What are the models that you use?

1. instruct-pix2pix (https://arxiv.org/abs/2211.09800) to generate the future verson of myself.
2. SAM (Segment Anything) to find a mask of the faces of characters in a real photo.
3. IP-Adapter to change the faces of them to my faces (my previous face and my future face).
4. Stable diffusion (https://stability.ai/research/stable-video-diffusion-scaling-latent-video-diffusion-models-to-large-datasets) to convert the image to a video.

Q) What prompts do you give to generate the image(s)/video(s)?

1. For instruct-pix2pix, I use the prompt "make him older" to to generate the future verson of myself.
2. For other models, no prompts are needed.

Q) What are your intermediate results?

The results are shown in next few sections.

Q) How to run this code:

Just run this notebook with necessary libararies installed. You can also use the google colab to directly run it: https://colab.research.google.com/drive/197259wMoHrw5WOSBr4EACVdgG6BTqVZg#scrollTo=cxx68W1rIGE4

Step 1: Install IP-Adapter

First, we are going to install the essential tools for IP-Adapter. This is an important package for one of the future step.

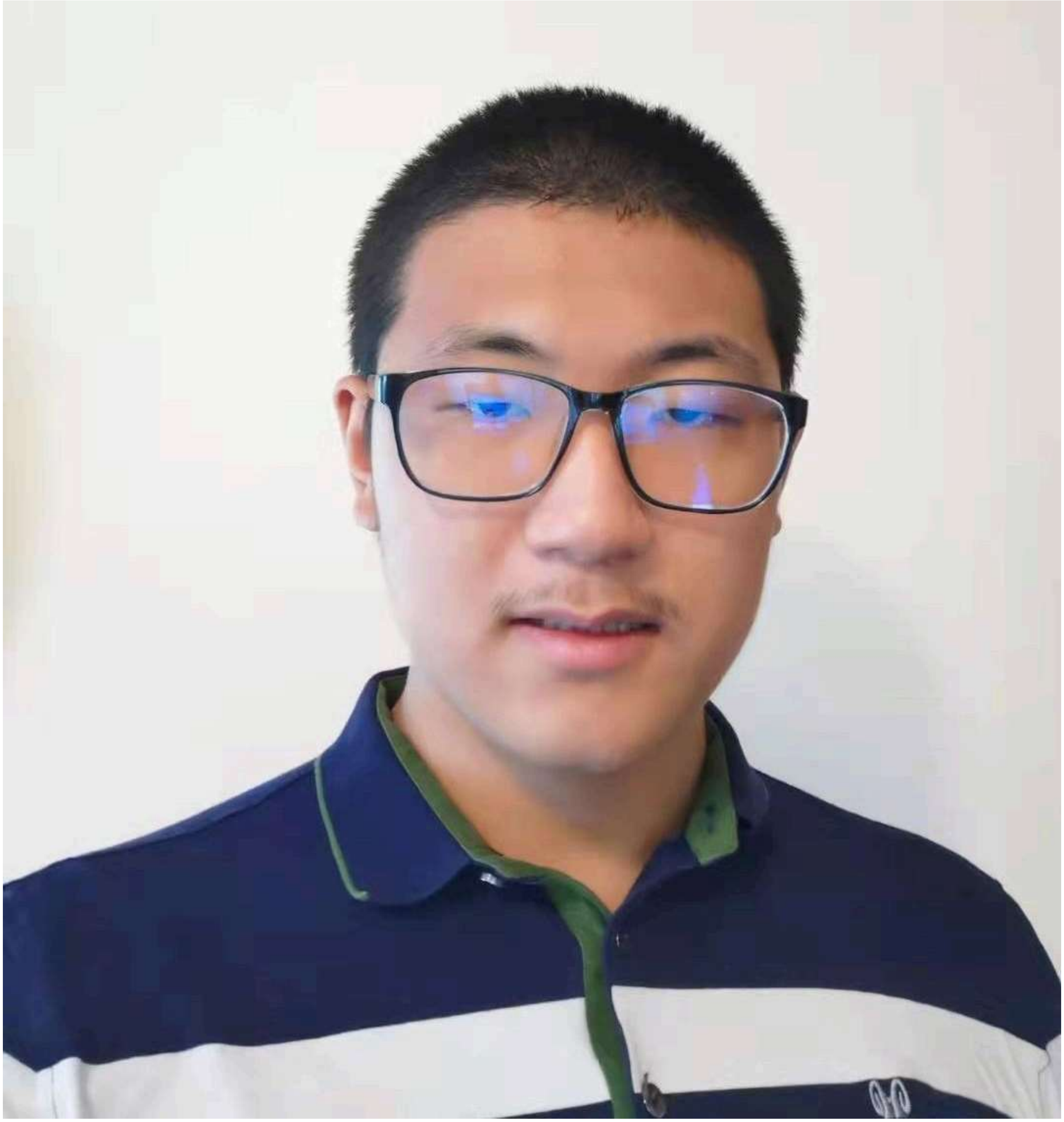
```
%cd /content/
!rm -rf IP-Adapter
!git clone https://github.com/tencent-ailab/IP-Adapter
!pip install git+https://github.com/tencent-ailab/IP-Adapter.git
%cd /content/IP-Adapter
```

```
/content
Cloning into 'IP-Adapter'...
remote: Enumerating objects: 497, done.
remote: Counting objects: 100% (201/201), done.
remote: Compressing objects: 100% (65/65), done.
remote: Total 497 (delta 170), reused 143 (delta 136), pack-reused 296 (from 1)
Receiving objects: 100% (497/497), 77.84 MiB | 7.30 MiB/s, done.
Resolving deltas: 100% (276/276), done.
Collecting git+https://github.com/tencent-ailab/IP-Adapter.git
  Cloning https://github.com/tencent-ailab/IP-Adapter.git to /tmp/pip-req-build-zyifs2t9
  Running command git clone --filter=blob:none --quiet https://github.com/tencent-ailab/IP-Adapter.git /tmp/pip-req-build-zyifs2t9
  Resolved https://github.com/tencent-ailab/IP-Adapter.git to commit 62e4af9d0c1ac7d5f8dd386a0ccf2211346af1a2
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
/content/IP-Adapter
```

Step 2: Get my photo

I am going to download a photo of myself.

```
import requests
img_data = requests.get("https://houmingchen.github.io/photo.jpg").content
with open('my_photo.jpg', 'wb') as handler:
    handler.write(img_data)

import PIL
from PIL import Image
past_photo = Image.open("my_photo.jpg")
past_photo
```



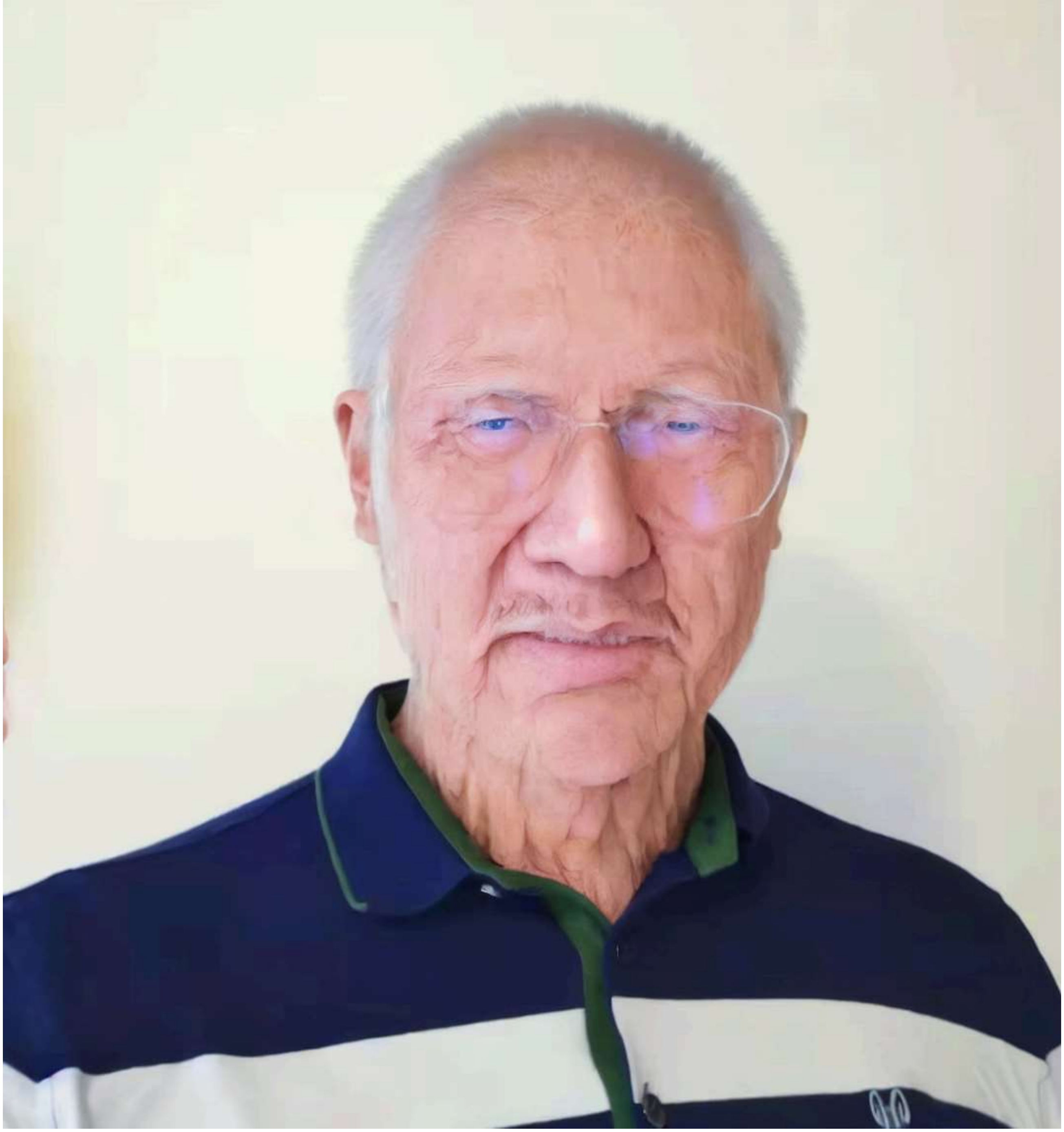
Step 3: Generate a future photo using instruct-pix2pix

I am going ot use the instruct-pix2pix (https://arxiv.org/abs/2211.09800) to generate a photo of myself in the future.

```
import requests
import torch
from diffusers import StableDiffusionInstructPix2PixPipeline, EulerAncestralDiscreteScheduler

model_id = "timbrooks/instruct-pix2pix"
pipe = StableDiffusionInstructPix2PixPipeline.from_pretrained(model_id, torch_dtype=torch.float16, safety_checker=None)
pipe.to("cuda")
g = torch.Generator(device="cuda")
g.manual_seed(0)
pipe.scheduler = EulerAncestralDiscreteScheduler.from_config(pipe.scheduler.config)

prompt = "make him older"
images = pipe(prompt, image=past_photo, num_inference_steps=10, image_guidance_scale=1.2).images
future_photo = images[0]
del pipe
torch.cuda.empty_cache()

future_photo
```

```
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and re
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

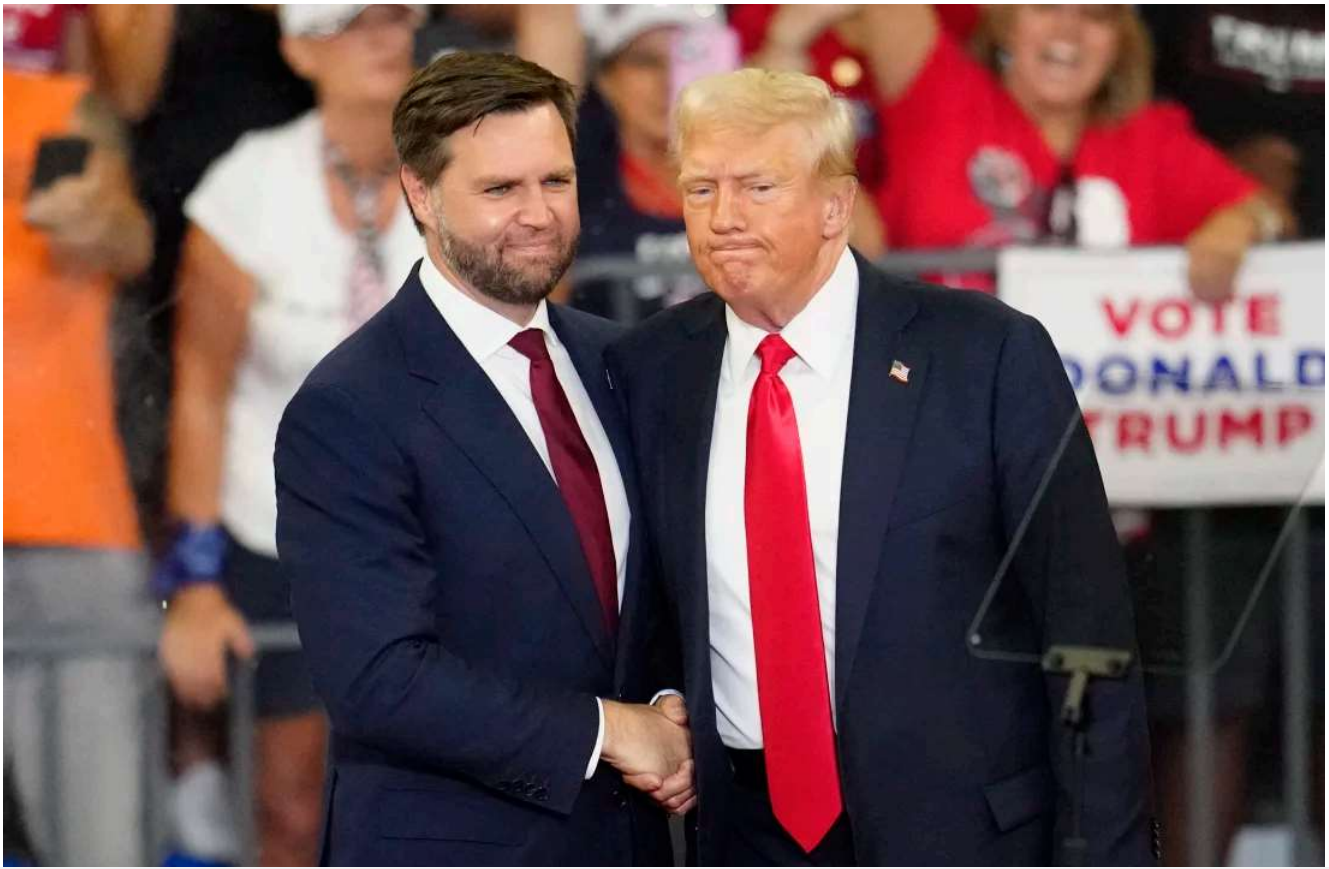
```
Loading pipeline components...: 100%                                    6/6 [00:32<00:00,  8.04s/it]

100%                                              10/10 [00:18<00:00,  1.95s/it]
```



Step 4: Download a real photo that contains two people.

Now, I download a real photo that has two main characters.

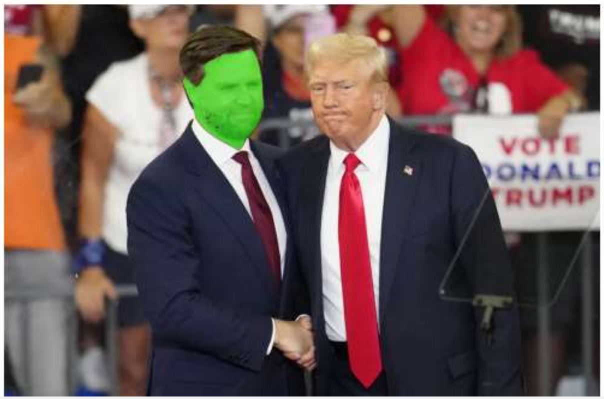
```
real_img_url = "https://npr.brightspotcdn.com/dims3/default/strip/false/crop/3161x2107+0+0/resize/1100/quality/85/format/webp/?url=http%3A%2F%2Fnpr-brightspot.s3.amazonaws.
img_data = requests.get(real_img_url).content
with open('real_photo.jpg', 'wb') as handler:
    handler.write(img_data)

import PIL
from PIL import Image
real_photo = Image.open("real_photo.jpg")
real_photo
```



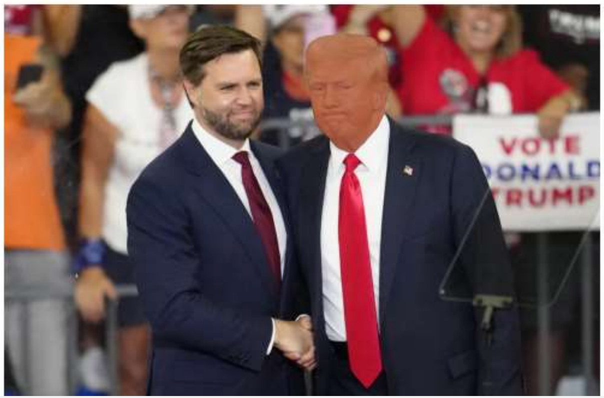
Step 5: Use SAM for segmentation.

I am going to use the SAM (https://segment-anything.com/) to do a segmentation to detect the face of the two characters.

```
from transformers import pipeline
generator =  pipeline("mask-generation", device = 'cuda', points_per_batch = 256, model="facebook/sam-vit-large")
outputs = generator(real_img_url, points_per_batch = 256)
del pipeline
torch.cuda.empty_cache()


def select_smallest_mask_containing_position(masks, position):
    x, y = position
    containing_masks = [mask for mask in masks if mask[y, x] == 1]
    if not containing_masks:
        return None  # No mask contains the position
    # Find the smallest mask by area (number of non-zero pixels)
    smallest_mask = min(containing_masks, key=lambda mask: np.sum(mask))
    return smallest_mask


import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30 / 255, 144 / 255, 255 / 255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

first_person_face = select_smallest_mask_containing_position(outputs["masks"], (450, 200)) # I used to few trials to find this (450, 200).
first_person_face = first_person_face
plt.imshow(np.array(real_photo))
ax = plt.gca()
for mask in [first_person_face]:
    show_mask(mask, ax=ax, random_color=True)
plt.axis("off")
plt.show()
```



```
second_person_face = select_smallest_mask_containing_position(outputs["masks"], (600, 200))
second_person_hair = select_smallest_mask_containing_position(outputs["masks"], (580, 80)) # I used to few trials to find these pixel positions to select the mask I want.

second_person_face = second_person_face | second_person_hair
plt.imshow(np.array(real_photo))
ax = plt.gca()
for mask in [second_person_face]:
    show_mask(mask, ax=ax, random_color=True)
plt.axis("off")
plt.show()
```



Step 6: Using IP-Adapter to edit face

Next, we are going to use the IP-Adapter (https://arxiv.org/abs/2308.06721) to change their face to me and the future me.

```
from huggingface_hub import hf_hub_download

import torch
from diffusers import StableDiffusionPipeline, StableDiffusionImg2ImgPipeline, StableDiffusionInpaintPipelineLegacy, DDIMScheduler, AutoencoderKL


from ip_adapter import IPAdapter

base_model_path = "runwayml/stable-diffusion-v1-5"
vae_model_path = "stabilityai/sd-vae-ft-mse"
image_encoder_path = "models/image_encoder/"
device = "cuda"

noise_scheduler = DDIMScheduler(
    num_train_timesteps=1000,
    beta_start=0.00085,
    beta_end=0.012,
    beta_schedule="scaled_linear",
    clip_sample=False,
    set_alpha_to_one=False,
    steps_offset=1,
)
vae = AutoencoderKL.from_pretrained(vae_model_path).to(dtype=torch.float16)
```

6.1 Swap the first face

```
torch.cuda.empty_cache()
pipe = StableDiffusionInpaintPipelineLegacy.from_pretrained(
    base_model_path,
    torch_dtype=torch.float16,
    scheduler=noise_scheduler,
    vae=vae,
    feature_extractor=None,
    safety_checker=None
)
```

```
Loading pipeline components...: 100%                                    5/5 [00:24<00:00,  6.97s/it]
/usr/local/lib/python3.10/dist-packages/diffusers/pipelines/deprecated/stable_diffusion_variants/pipeline_stable_diffusion_inpaint_legacy.py:141: FutureWarning: The cla
  deprecate("legacy is outdated", "1.0.0", deprecation_message, standard_warn=False)
You have disabled the safety checker for <class 'diffusers.pipelines.deprecated.stable_diffusion_variants.pipeline_stable_diffusion_inpaint_legacy.StableDiffusionInpain
```

```
import os
ip_adapter_download_path = hf_hub_download(repo_id="h94/IP-Adapter", filename="models/ip-adapter_sd15.bin")
hf_hub_download(repo_id="h94/IP-Adapter", filename="models/image_encoder/config.json")
hf_hub_download(repo_id="h94/IP-Adapter", filename="models/image_encoder/model.safetensors")
image_encoder_download_path = os.path.dirname(hf_hub_download(repo_id="h94/IP-Adapter", filename="models/image_encoder/pytorch_model.bin"))
```

```
masked_image = real_photo.resize((768, 512))
mask = Image.fromarray(first_person_face).resize((768, 512))
```

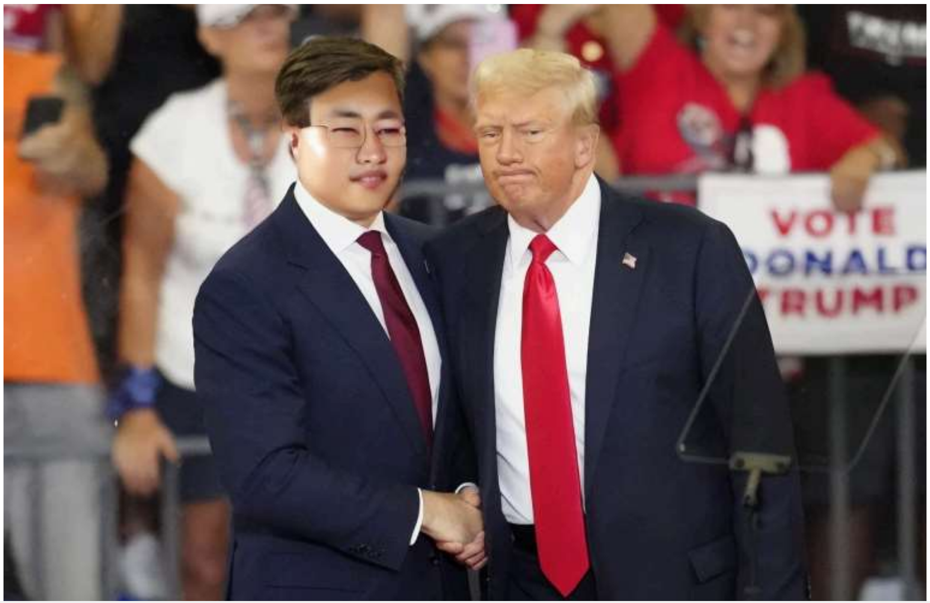
```
ip_model = IPAdapter(pipe, image_encoder_download_path, ip_adapter_download_path, device)
```

```
/content/IP-Adapter/ip_adapter/ip_adapter.py:134: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the defaul
  state_dict = torch.load(self.ip_ckpt, map_location="cpu")
```

```
images = ip_model.generate(pil_image=past_photo, num_samples=1, num_inference_steps=50,
                           seed=42, image=masked_image, mask_image=mask, strength=0.8)
```

```
100%                                              40/40 [00:10<00:00,  3.91it/s]
```

```
editied_photo = images[0]
editied_photo
```



6.2 Swap the second face

```
torch.cuda.empty_cache()
pipe = StableDiffusionInpaintPipelineLegacy.from_pretrained(
    base_model_path,
    torch_dtype=torch.float16,
    scheduler=noise_scheduler,
    vae=vae,
    feature_extractor=None,
    safety_checker=None
)
```

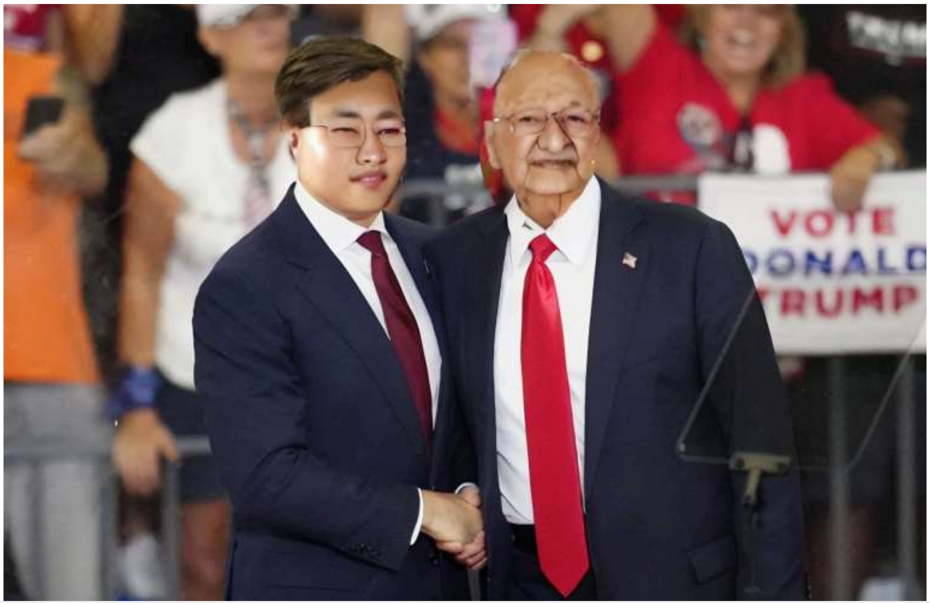
```
Loading pipeline components...: 100%                                    5/5 [00:24<00:00,  6.93s/it]
You have disabled the safety checker for <class 'diffusers.pipelines.deprecated.stable_diffusion_variants.pipeline_stable_diffusion_inpaint_legacy.StableDiffusionInpain
```

```
import os
ip_adapter_download_path = hf_hub_download(repo_id="h94/IP-Adapter", filename="models/ip-adapter_sd15.bin")
hf_hub_download(repo_id="h94/IP-Adapter", filename="models/image_encoder/config.json")
hf_hub_download(repo_id="h94/IP-Adapter", filename="models/image_encoder/model.safetensors")
image_encoder_download_path = os.path.dirname(hf_hub_download(repo_id="h94/IP-Adapter", filename="models/image_encoder/pytorch_model.bin"))
```

```
masked_image = editied_photo.resize((768, 512))
mask = Image.fromarray(second_person_face).resize((768, 512))


images = ip_model.generate(pil_image=future_photo, num_samples=1, num_inference_steps=50,
                           seed=42, image=masked_image, mask_image=mask, strength=0.8)
```

```
100%                                             40/40 [00:10<00:00,  3.91it/s]
```

```
editied_photo = images[0]
editied_photo
```



```
del pipe, ip_model
torch.cuda.empty_cache()
```

Step 7: Image to Video

Finally, we use the img2vid model developed by stable diffusion (https://stability.ai/research/stable-video-diffusion-scaling-latent-video-diffusion-models-to-large-datasets) to make this image to a short video clip.

```
from diffusers import StableVideoDiffusionPipeline
from diffusers.utils import load_image, export_to_video

pipeline = StableVideoDiffusionPipeline.from_pretrained(
    "stabilityai/stable-video-diffusion-img2vid", torch_dtype=torch.float16, variant="fp16"
)
pipeline.enable_model_cpu_offload()

image = editied_photo
image = image.resize((384, 256))

generator = torch.manual_seed(42)
frames_1 = pipeline(image, decode_chunk_size=8, generator=generator).frames[0]
frames_2 = pipeline(frames_1[0], decode_chunk_size=8, generator=generator).frames[0]
```

```
Loading pipeline components...: 100%                                    5/5 [00:01<00:00,  9.59it/s]

100%                                             25/25 [03:35<00:00,  7.58s/it]

100%                                             25/25 [03:12<00:00,  7.58s/it]
```

```
gif = frames_1+ frames_2
gif[0].save("out.gif", save_all=True, append_images=gif[1:], loop=0, duration=2)
from IPython import display
display.Image(open("out.gif", 'rb').read())  # local
```