Generative AI - Hackathon Assignment

My Experience :

* This was a great learning experience for me by engaging myself in this Generative AI hackathon.
* I had zero knowledge about the stable diffusion model at the beginning of the week. But currently I developed a working generative AI model to generate the illustration of superman with my face in it .
* This assessment was really challenging and also tested how fast I can learn from scratch to final.
* I was not having any issues in creating the model but I am facing problems in creating the API for the generative AI model which I created. But I will learn more about the API and create one for my model in the future.
* It was so fun to play with this generative AI model while generating images by giving prompts as inputs. This model illustrated myself as superman.

Sample Diffusion Models :

* Sample Diffusion Models are generative models, meaning that they are used to generate data similar to the data on which they are trained.
* Diffusion Models work by destroying training data through the successive addition of Gaussian noise, and then learning to recover the data by reversing this noising process.
* This type of model is used to develop doppelgangers which are used as characters in the gaming field.

Generative AI for Avatar Faces :

* This Artificial Intelligence model generates avatar faces using a custom stable diffusion model.

Tools and Libraries used :

1. Google Colab.
2. PyTorch library.
3. Transformers.
4. DALLE - PyTorch library.
5. Flask framework.

* Usage:

avatar\_face = generate\_avatar\_face(image\_path)

Here, we are providing the image’s file path as an input directory to the stable diffusion model for generating the avatar face.

def generate\_avatar\_face(image\_path):

This function is used to generate the avatar faces using a custom stable diffusion model.

Parameters :

image\_path (str): The path to the input image.

Returns:

avatar\_face (PIL.Image): The generated avatar face.

Code Implementation :

#Basic Requirements:

!wget -q https://github.com/ShivamShrirao/diffusers/raw/main/examples/dreambooth/train\_dreambooth.py

!wget -q https://github.com/ShivamShrirao/diffusers/raw/main/scripts/convert\_diffusers\_to\_original\_stable\_diffusion.py

%pip install -qq git+https://github.com/ShivamShrirao/diffusers

%pip install -q -U --pre triton

%pip install -q accelerate transformers ftfy bitsandbytes==0.35.0 gradio natsort safetensors xformers

!apt-get install wget

!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip

!unzip ngrok-stable-linux-amd64.zip

!./ngrok authtoken 2N8CgXpCOPUYLXNVNIy6KNCV5nD\_3Z735wg5PANop9cKoxyYN

#Hugging Face Token:

!mkdir -p ~/.huggingface

HUGGINGFACE\_TOKEN = "hf\_JNAFaNvxoAdiCRgtWhVKGlzKGhbVhxgwHo"

!echo -n "{HUGGINGFACE\_TOKEN}" > ~/.huggingface/token

#Connecting to google drive :

save\_to\_gdrive = True

if save\_to\_gdrive:

from google.colab import drive

drive.mount('/content/drive')

MODEL\_NAME = "runwayml/stable-diffusion-v1-5"

OUTPUT\_DIR = "stable\_diffusion\_weights/cxyz"

if save\_to\_gdrive:

OUTPUT\_DIR = "/content/drive/MyDrive/" + OUTPUT\_DIR

else:

OUTPUT\_DIR = "/content/" + OUTPUT\_DIR

print(f"[\*] Weights will be saved at {OUTPUT\_DIR}")

!mkdir -p $OUTPUT\_DIR

#Creating directories :

concepts\_list = [

{

"instance\_prompt": "cxyz",

"class\_prompt": "human image",

"instance\_data\_dir": "/content/drive/MyDrive/myimage",

"class\_data\_dir": "/content/drive/MyDrive/human pics"

}

]

import json

import os

for c in concepts\_list:

os.makedirs(c["instance\_data\_dir"], exist\_ok=True)

with open("concepts\_list.json", "w") as f:

json.dump(concepts\_list, f, indent=4)

#Uploading the Dataset :

import os

from google.colab import files

import shutil

for c in concepts\_list:

print(f"Uploading instance images for `{c['instance\_prompt']}`")

uploaded = files.upload()

for filename in uploaded.keys():

dst\_path = os.path.join(c['instance\_data\_dir'], filename)

shutil.move(filename, dst\_path)

#Pre-training of the AI model :

!accelerate launch train\_dreambooth.py \

--pretrained\_model\_name\_or\_path=$MODEL\_NAME \

--pretrained\_vae\_name\_or\_path="stabilityai/sd-vae-ft-mse" \

--output\_dir=$OUTPUT\_DIR \

--revision="fp16" \

--with\_prior\_preservation --prior\_loss\_weight=1.0 \

--seed=1337 \

--resolution=512 \

--train\_batch\_size=1 \

--train\_text\_encoder \

--mixed\_precision="fp16" \

--use\_8bit\_adam \

--gradient\_accumulation\_steps=1 \

--learning\_rate=1e-6 \

--lr\_scheduler="constant" \

--lr\_warmup\_steps=168 \ #lr\_warmup\_steps = int(max\_train\_steps / 10)

--num\_class\_images=252 \ #num\_class\_images = num\_instance\_ images \* 12

--sample\_batch\_size=4 \

--max\_train\_steps=1680 \ #max\_train\_steps = num\_instance\_ images \* 80

--save\_interval=10000 \

--save\_sample\_prompt="cxyz" \

--concepts\_list="concepts\_list.json"

#Weight directories :

WEIGHTS\_DIR = ""

if WEIGHTS\_DIR == "":

from natsort import natsorted

from glob import glob

import os

WEIGHTS\_DIR = natsorted(glob(OUTPUT\_DIR + os.sep + "\*"))[-1]

print(f"[\*] WEIGHTS\_DIR={WEIGHTS\_DIR}")

#Generating a grid of preview images from the last saved weights :

import os

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

weights\_folder = OUTPUT\_DIR

folders = sorted([f for f in os.listdir(weights\_folder) if f != "0"], key=lambda x: int(x))

row = len(folders)

col = len(os.listdir(os.path.join(weights\_folder, folders[0], "samples")))

scale = 4

fig, axes = plt.subplots(row, col, figsize=(col\*scale, row\*scale), gridspec\_kw={'hspace': 0, 'wspace': 0})

for i, folder in enumerate(folders):

folder\_path = os.path.join(weights\_folder, folder)

image\_folder = os.path.join(folder\_path, "samples")

images = [f for f in os.listdir(image\_folder)]

for j, image in enumerate(images):

if row == 1:

currAxes = axes[j]

else:

currAxes = axes[i, j]

if i == 0:

currAxes.set\_title(f"Image {j}")

if j == 0:

currAxes.text(-0.1, 0.5, folder, rotation=0, va='center', ha='center', transform=currAxes.transAxes)

image\_path = os.path.join(image\_folder, image)

img = mpimg.imread(image\_path)

currAxes.imshow(img, cmap='gray')

currAxes.axis('off')

plt.tight\_layout()

plt.savefig('grid.png', dpi=72)

#Inference :

import torch

from torch import autocast

from diffusers import StableDiffusionPipeline, DDIMScheduler

from IPython.display import display

from flask import Flask

app = Flask(\_\_name\_\_)

model\_path = WEIGHTS\_DIR

pipe = StableDiffusionPipeline.from\_pretrained(model\_path, safety\_checker=None, torch\_dtype=torch.float16).to("cuda")

pipe.scheduler = DDIMScheduler.from\_config(pipe.scheduler.config)

pipe.enable\_xformers\_memory\_efficient\_attention()

g\_cuda = None

#Setting random seed here for reproducibility :

g\_cuda = torch.Generator(device='cuda')

seed = 52362

g\_cuda.manual\_seed(seed)

#Creating API for the model and fine tuning it using PyTorch libraries :

from flask import Flask

from flask import request

from flask import Response

from flask\_ngrok import run\_with\_ngrok

import os

import json

app = Flask(\_\_name\_\_)

run\_with\_ngrok(app)

if not os.path.exists('images'):

os.makedirs('images')

@app.route("/test")

def test():

return "Test is successful"

@app.route("/avatar-face")

def generateImage():

prompt = request.form['prompt']

negative\_prompt = request.form['negative\_prompt']

num\_samples = request.form['num\_samples']

guidance\_scale = 7.9

num\_inference\_steps = 50

height = 512

width = 512

images\_list = []

with autocast("cuda"), torch.inference\_mode():

images = pipe(

prompt,

height=height,

width=width,

negative\_prompt=negative\_prompt,

num\_images\_per\_prompt=num\_samples,

num\_inference\_steps=num\_inference\_steps,

guidance\_scale=guidance\_scale,

generator=g\_cuda

).images

for img in images:

os.rename(os.getcwd()+"/images",img)

images\_list.append("http://eb12-34-141-227-3.ngrok.io/"+os.getcwd()+"/images"+img)

#display(img)

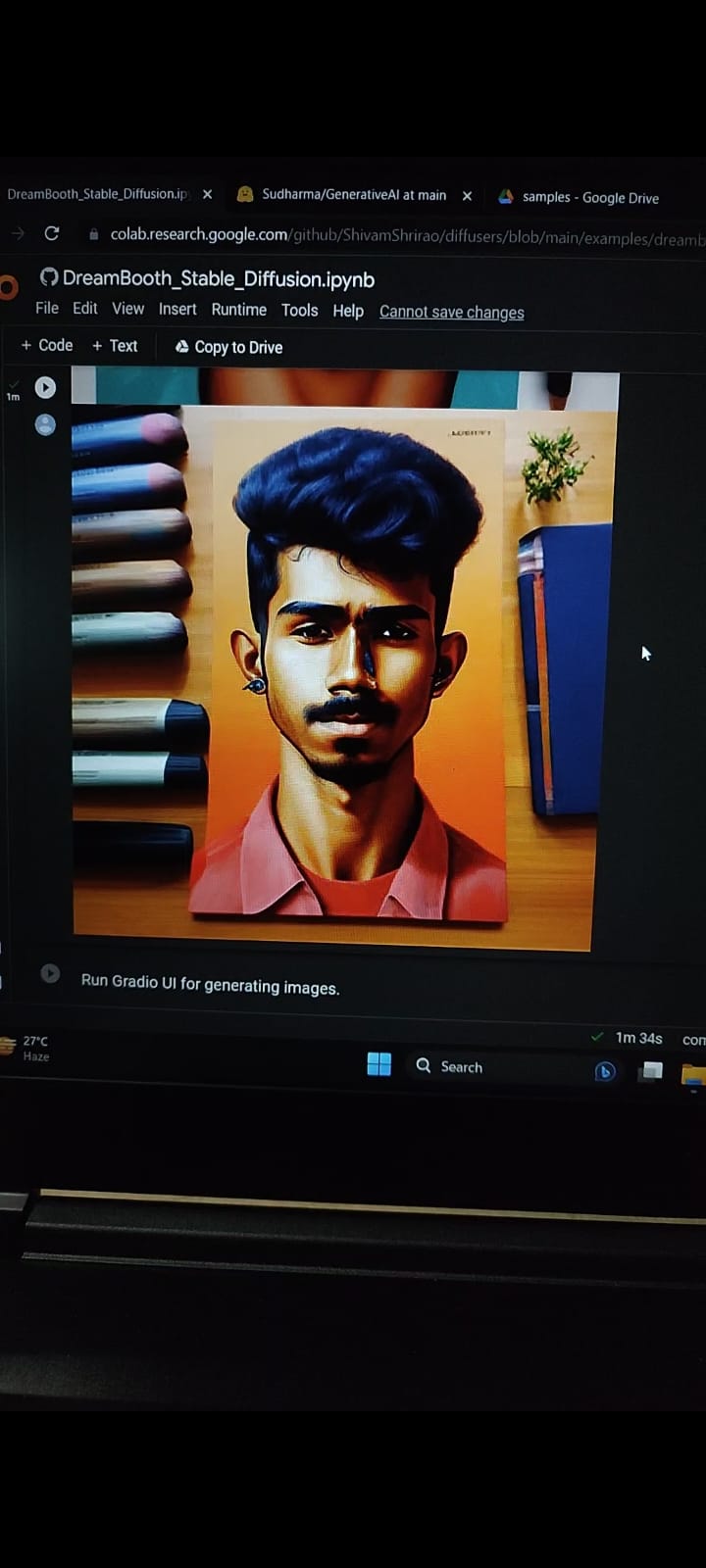
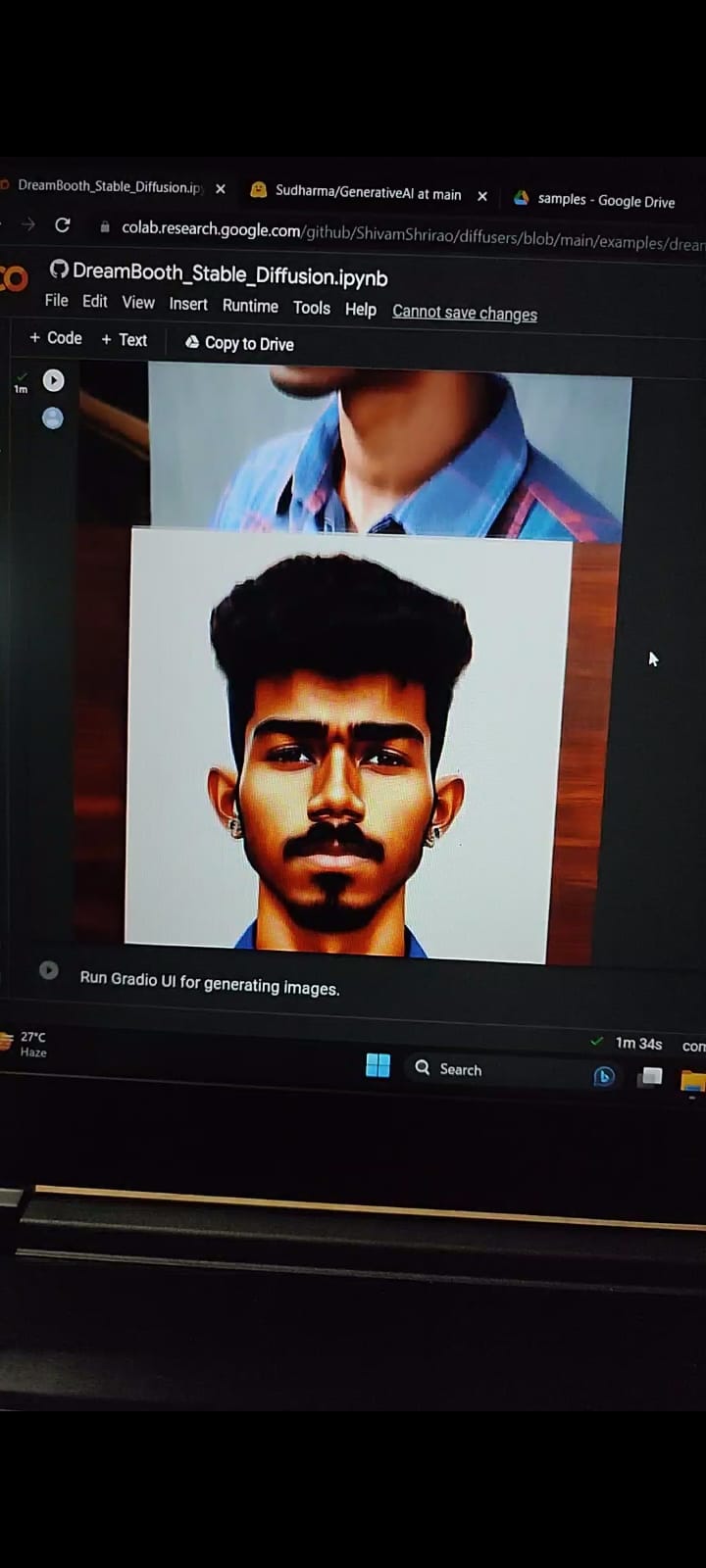
return Response(json.dumps(images\_list), mimetype='application/json')

app.run()

Outputs :







Postman outputs :

