National College of Ireland

Configuration Manual

MSc Research Project
Practicum 2

Forename Surname
Student ID: 22209387

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Sonal Deepak Pardesi …….

Student ID: ……22209387……

Programme: ……MSc. AI for Business…… **Year:** …2023 - 2024

Module: ……Practicum…2……

Lecturer: ……Anderson Simiscuka……

Submission Due Date: ……

Project Title: …Pixelating to the Edge – Generative AI Art on Edge Devices……..

Word Count: …… **Page Count:** ……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Forename Surname
Student ID:

1 Introduction

This is a configuration manual to give you a step by step guide on how can configure, run and observe results for a text to image based application. The code for the application was developed on google collab hence the steps are provided for the same environment. The objective of the research is to calculate and compare inference time, FID and CLIP score. The manual consists of three parts 1. Original Code to create an initial reference result. 2. We make some changes in the architecture and sampling operations for better results along with generating images from prompts 3. Code with the final architectural change for maximum efficiency.

2 Downloading Essential

This application was created using Google Colab and hence an account on google colab is essential. Once the account is created the screen should look like below. Click on new notebook to start coding. Since the text to image application is a GPU intensive application it is recommended to change the run time as shown below to the T4GPU.

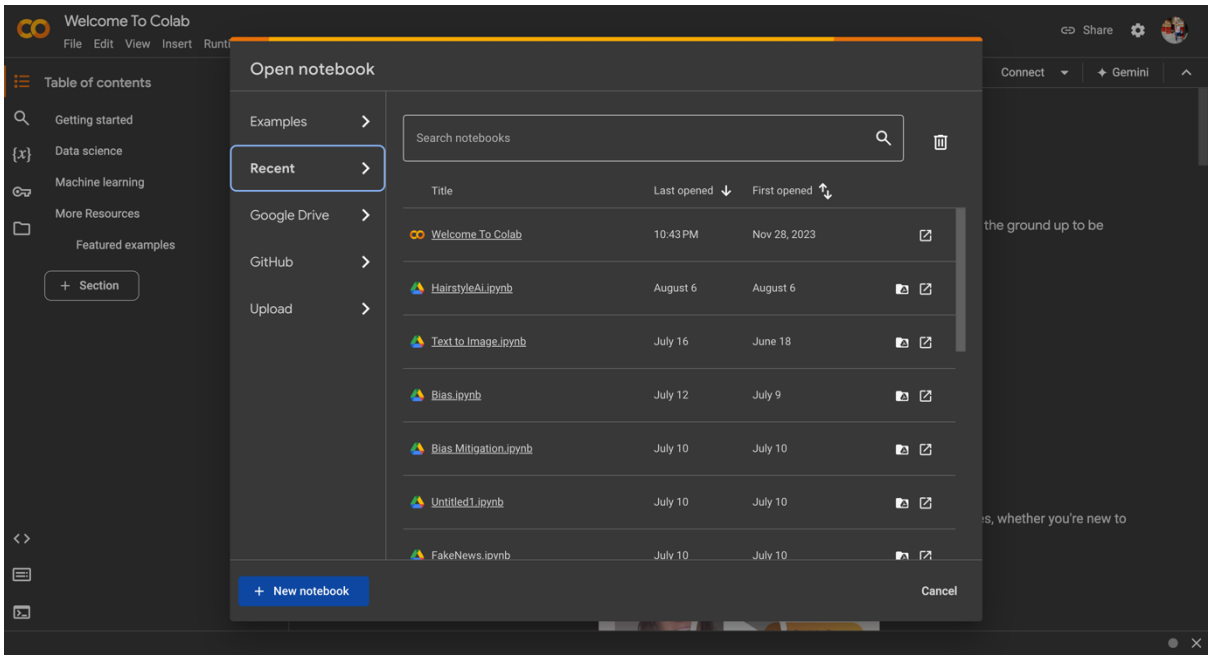

Fig 1. Starting Screen of Google Colab. Click on New Notebook to start coding.

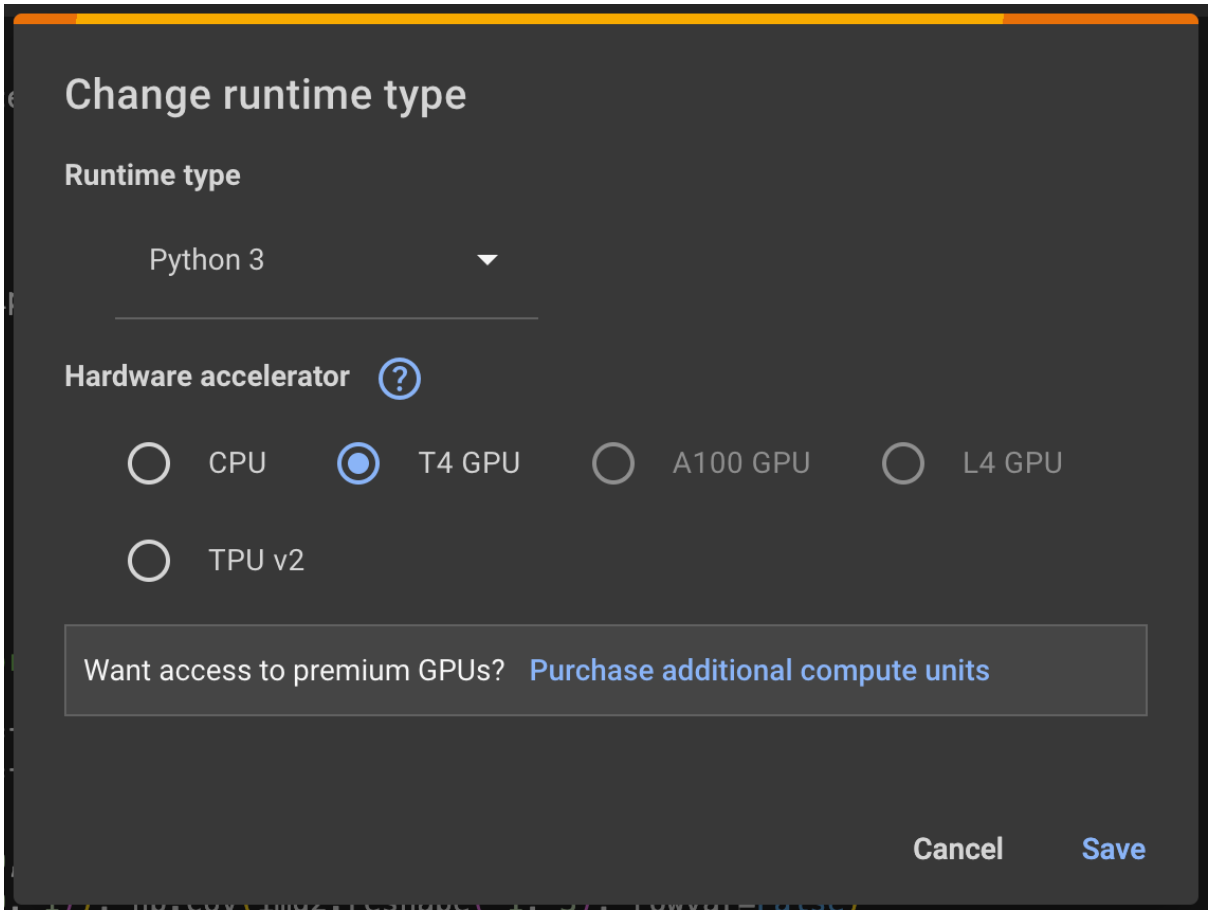

Fig 2. Make sure to enable T4GPU for uninterrupted performance

2.1 Prerequisites

To make sure that we obtain all the functions and they work properly it is essential to download some essential libraries. Below is the list of libraries to be downloaded.

1. torch
2. time
3. diffuser (including StableDiffusionPipeline, DDIMScheduler, PNDMScheduler, UNet2DModel)
4. transformers (including CLIPTextModel, CLIPTokenizer, CLIPProcessor, CLIPModel)
5. scipy (linalg)
6. numpy
7. PIL (Image)
8. requests
9. io (BytesIO)
10. matplotlib.pyplot
11. IPython.display
12. Ipywidgets

Below code can be used to install and import libraries.

```
!pip install torch torchvision torchaudio
!pip install diffusers transformers
!pip install scipy numpy pillow requests
!pip install matplotlib ipywidgets
```

Fig 3. To install required libraries

```
import torch
import time
from diffusers import StableDiffusionPipeline, DDIMScheduler
from transformers import CLIPTextModel, CLIPTokenizer, CLIPProcessor, CLIPModel
from scipy import linalg
import numpy as np
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
from IPython.display import display, Image as IPyImage
import ipywidgets as widgets
```

Fig 4. Importing all the libraries

2.2 Starting with the Initial Program

First, we will start with configure for calculation of FID and CLIP score. The following code can be used for it.

```
# Helper functions for FID and CLIP score calculations
def calculate_fid(image1, image2):
    img1 = np.array(image1).astype(np.float32)
    img2 = np.array(image2).astype(np.float32)

    mu1, sigma1 = np.mean(img1, axis=(0, 1)), np.cov(img1.reshape(-1, 3), rowvar=False)
    mu2, sigma2 = np.mean(img2, axis=(0, 1)), np.cov(img2.reshape(-1, 3), rowvar=False)

    ssdiff = np.sum((mu1 - mu2)**2.0)
    covmean = linalg.sqrtm(sigma1.dot(sigma2))
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)

    return fid

def calculate_clip_score(image, text, clip_model, clip_tokenizer):
    inputs = clip_tokenizer(text, return_tensors="pt").to("cuda")
    outputs = clip_model.get_text_features(**inputs)
    img_tensor = pipe.feature_extractor(images=image, return_tensors="pt").pixel_values.to("cuda")
    img_features = clip_model.get_image_features(pixel_values=img_tensor)
    return torch.cosine_similarity(outputs, img_features).item()

# Load CLIP model and tokenizer
clip_model = CLIPTextModel.from_pretrained("openai/clip-vit-base-patch32").to("cuda")
clip_tokenizer = CLIPTokenizer.from_pretrained("openai/clip-vit-base-patch32")
```

Fig. 5 Code FID and CLIP

Second step is to load the pre trained model. Here we are using stable diffusion model with mixed precision.

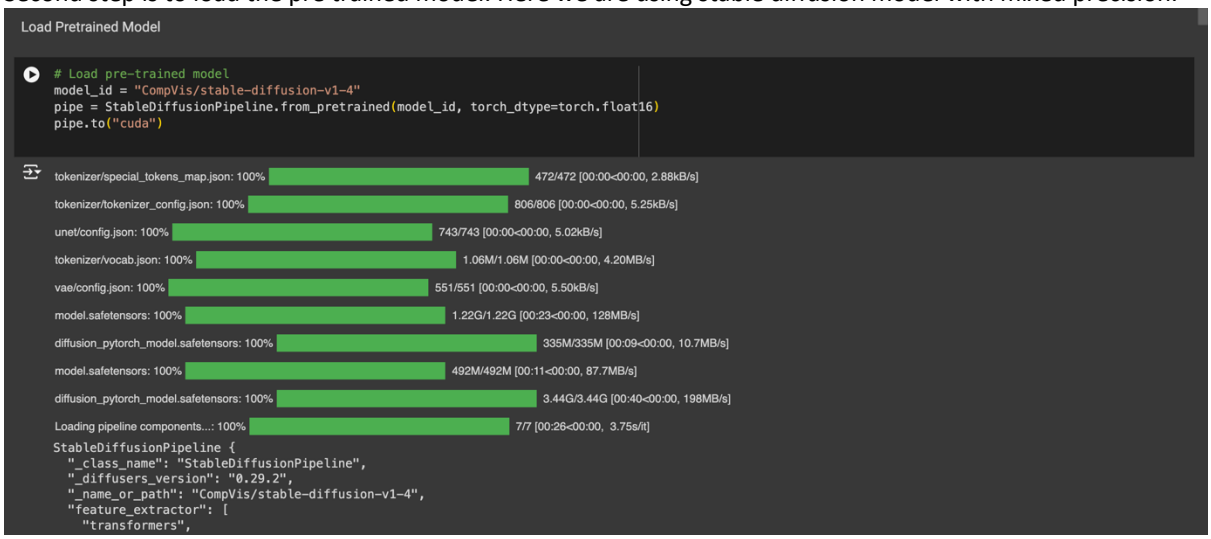

Fig 6. Code for loading Pre trained model.

Next we code to generate image and obtain the initial scores as shown below. These scores will be used as reference score in the project ahead.

07/08/2024, 23:38 Tony Stark - Colab

```
# Generate an image from a prompt
prompt = "A beautiful sunset over the mountains"
start_time = time.time()
generated_image_initial = pipe(prompt).images[0]
initial_inference_time = time.time() - start_time

# Use a placeholder reference image
from PIL import Image
import urllib

# Use a valid URL for a reference image
url = "https://upload.wikimedia.org/wikipedia/commons/4/47/PNG_transparency_demonstration_1.png"
response = requests.get(url)
try:
    reference_image = Image.open(BytesIO(response.content)).convert("RGB")
except UnidentifiedImageError as e:
    print("Error: Cannot identify image file", e)
    raise

# Correcting the CLIP model usage
from transformers import CLIPProcessor, CLIPModel

clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to("cuda")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

def calculate_clip_score(image, text, clip_model, clip_processor):
    inputs = clip_processor(text=[text], images=image, return_tensors="pt", padding=True).to("cuda")
    outputs = clip_model(**inputs)
    return outputs.logits_per_image.item()

# Calculate initial FID and CLIP scores
fid_score_initial = calculate_fid(generated_image_initial, reference_image)
clip_score_initial = calculate_clip_score(generated_image_initial, prompt, clip_model, clip_processor)

print(f"Initial Inference Time: {initial_inference_time} seconds")
print(f"Initial FID Score: {fid_score_initial}")
print(f"Initial CLIP Score: {clip_score_initial}")
```

Fig 7. Initial Score Calculation

2.3 Modified Code with mixed precision and sampling operations

Follow the below code to create a working application where user can input prompt and generate images and this code will also compare the results to the initial model.

```python
# Install necessary libraries
!pip install diffusers transformers accelerate torch torchvision scipy ipywidgets

# Import libraries
import torch
import time
from diffusers import StableDiffusionPipeline, DDIMScheduler
from transformers import CLIPProcessor, CLIPModel
from scipy import linalg
import numpy as np
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
from IPython.display import display, Image as IPyImage
import ipywidgets as widgets

# Helper functions for FID and CLIP score calculations
def calculate_fid(image1, image2):
    img1 = np.array(image1).astype(np.float32)
    img2 = np.array(image2).astype(np.float32)

    mu1, sigma1 = np.mean(img1, axis=(0, 1)), np.cov(img1.reshape(-1, 3), rowvar=False)
    mu2, sigma2 = np.mean(img2, axis=(0, 1)), np.cov(img2.reshape(-1, 3), rowvar=False)

    ssdiff = np.sum((mu1 - mu2)**2.0)
    covmean = linalg.sqrtm(sigma1.dot(sigma2))
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)

    return fid
```

```python
def calculate_clip_score(image, text, clip_model, clip_processor):
    # Convert PIL image to pixel values
    image = clip_processor(images=image, return_tensors="pt").pixel_values.to("cuda")
    inputs = clip_processor(text=[text], return_tensors="pt", padding=True).to("cuda")
    outputs = clip_model(input_ids=inputs.input_ids, pixel_values=image)
    return outputs.logits_per_image.item()

# Load CLIP model and processor
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to("cuda")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Load pre-trained model with mixed precision
model_id = "CompVis/stable-diffusion-v1-4"
pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
pipe.to("cuda")

# Use DDIM Scheduler with fewer timesteps for efficient sampling
pipe.scheduler = DDIMScheduler.from_config(pipe.scheduler.config)
pipe.scheduler.set_timesteps(3)  # Further reducing the number of timesteps

# Enable mixed precision for speed
pipe = pipe.to(torch.float16)

# Create a text prompt input box
text_prompt = widgets.Text(
    value='A beautiful sunset over the mountains',
    placeholder='Type your text prompt here',
    description='Text Prompt:',
    disabled=False
)
display(text_prompt)
```

```python
# Button to generate image
button = widgets.Button(description="Generate Image")

output = widgets.Output()

def on_button_clicked(b):
    with output:
        output.clear_output()
        # Generate an image from the user-provided prompt
        prompt = text_prompt.value
        start_time = time.time()
        generated_image_modified = pipe(prompt).images[0]
        modified_inference_time = time.time() - start_time

        # Display the generated image
        generated_image_modified.save("generated_image.png")
        display(IPyImage(filename="generated_image.png"))

        # Use a placeholder reference image
        url = "https://upload.wikimedia.org/wikipedia/commons/4/47/PNG_transparency_demonstration_1.png"
        response = requests.get(url)
        try:
            reference_image = Image.open(BytesIO(response.content)).convert("RGB")
        except UnidentifiedImageError as e:
            print("Error: Cannot identify image file", e)
            raise
```

```
        # Calculate modified FID and CLIP scores
        fid_score_modified = calculate_fid(generated_image_modified, reference_image)
        clip_score_modified = calculate_clip_score(generated_image_modified, prompt, clip_model, clip_processor)

        print(f"Modified Inference Time: {modified_inference_time} seconds")
        print(f"Modified FID Score: {fid_score_modified}")
        print(f"Modified CLIP Score: {clip_score_modified}")

        # Initial results for comparison (if needed)
        initial_inference_time = 9.26  # Placeholder value
        fid_score_initial = 11036.34   # Placeholder value
        clip_score_initial = 29.23     # Placeholder value

        # Comparison function
        def compare_results(initial_time, modified_time, initial_fid, modified_fid, initial_clip, modified_clip):
            print("Comparison of Initial and Modified Model Performance:")
            print(f"Initial Inference Time: {initial_time:.2f} seconds")
            print(f"Modified Inference Time: {modified_time:.2f} seconds")
            print(f"Time Reduction: {(initial_time - modified_time) / initial_time * 100:.2f}%")

            print(f"Initial FID Score: {initial_fid:.2f}")
            print(f"Modified FID Score: {modified_fid:.2f}")
            print(f"FID Score Improvement: {(initial_fid - modified_fid):.2f}")

            print(f"Initial CLIP Score: {initial_clip:.2f}")
            print(f"Modified CLIP Score: {modified_clip:.2f}")
            print(f"CLIP Score Improvement: {(modified_clip - initial_clip):.2f}")

            # Plotting the results
            labels = ['Inference Time (s)', 'FID Score', 'CLIP Score']
            initial = [initial_time, initial_fid, initial_clip]
            modified = [modified_time, modified_fid, modified_clip]

            x = np.arange(len(labels))
            width = 0.35

            fig, ax = plt.subplots()
            rects1 = ax.bar(x - width/2, initial, width, label='Initial')
            rects2 = ax.bar(x + width/2, modified, width, label='Modified')

            ax.set_ylabel('Scores')
            ax.set_title('Initial vs Modified Model Performance')
            ax.set_xticks(x)
            ax.set_xticklabels(labels)
            ax.legend()

            fig.tight_layout()
            plt.show()

        # Comparing the results again
        compare_results(initial_inference_time, modified_inference_time, fid_score_initial, fid_score_modified, clip_score_initial, clip_score_modified)

button.on_click(on_button_clicked)

display(button, output)
```

Fig 8. Modified Code

The output should look like this:

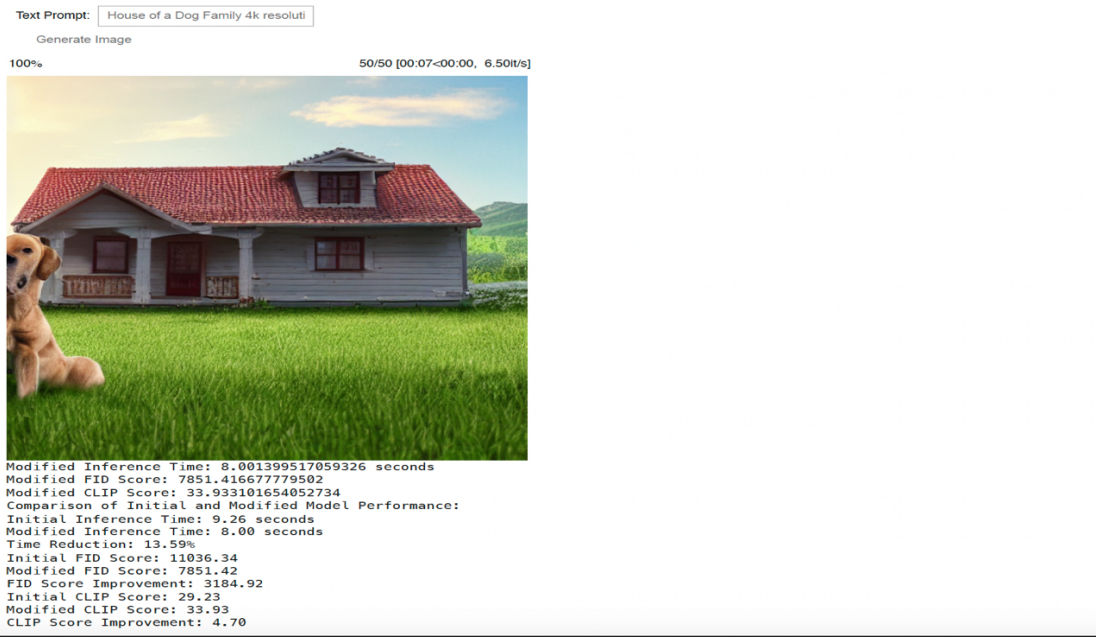

Fig 9. Output for the new code where user can input prompt.

3 Final UNet Modification

In this step we will implement the final code which will involve both modification in the UNet architecture and the sampling operation to increase efficiency. Also same as the previous code user can input prompt, generate image and compare the results.

```python
# Define a simple configuration class
class Config:
    def __init__(self, in_channels, out_channels):
        self.in_channels = in_channels
        self.out_channels = out_channels

# Example Configuration
config = Config(in_channels=3, out_channels=3)

# Define Swish Activation
class Swish(nn.Module):
    def forward(self, x):
        return x * torch.sigmoid(x)

# Define Depthwise Separable Convolution
class SeparableConv2d(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, padding=1):
        super(SeparableConv2d, self).__init__()
        self.depthwise = nn.Conv2d(in_channels, in_channels, kernel_size=kernel_size, padding=padding, groups=in_channels)
        self.pointwise = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        x = self.depthwise(x)
        x = self.pointwise(x)
        return x

# Custom UNet Model
class CustomUNet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(CustomUNet, self).__init__()
        self.encoder = nn.Sequential(
            SeparableConv2d(in_channels, 64),
            Swish(),
            SeparableConv2d(64, 128),
            Swish(),
        )
        self.decoder = nn.Sequential(
            SeparableConv2d(128, 64),
            Swish(),
            nn.Conv2d(64, out_channels, kernel_size=3, padding=1)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

```python
# Integrate with Diffusers
class CustomUNetModel(UNet2DModel):
    def __init__(self, config):
        super().__init__(config)
        self.unet = CustomUNet(config.in_channels, config.out_channels)

    def forward(self, x):
        return self.unet(x)

# Define helper functions for FID and CLIP score calculations
def calculate_fid(image1, image2):
    img1 = np.array(image1).astype(np.float32)
    img2 = np.array(image2).astype(np.float32)

    mu1, sigma1 = np.mean(img1, axis=(0, 1)), np.cov(img1.reshape(-1, 3), rowvar=False)
    mu2, sigma2 = np.mean(img2, axis=(0, 1)), np.cov(img2.reshape(-1, 3), rowvar=False)

    ssdiff = np.sum((mu1 - mu2)**2.0)
    covmean = linalg.sqrtm(sigma1.dot(sigma2))
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)

    return fid

def calculate_clip_score(image, text, clip_model, clip_processor):
    # Convert PIL image to pixel values
    image = clip_processor(images=image, return_tensors="pt").pixel_values.to("cuda")
    inputs = clip_processor(text=[text], return_tensors="pt", padding=True).to("cuda")
    outputs = clip_model(input_ids=inputs.input_ids, pixel_values=image)
    return outputs.logits_per_image.item()
```

```python
def calculate_clip_score(image, text, clip_model, clip_processor):
    # Convert PIL image to pixel values
    image = clip_processor(images=image, return_tensors="pt").pixel_values.to("cuda")
    inputs = clip_processor(text=[text], return_tensors="pt", padding=True).to("cuda")
    outputs = clip_model(input_ids=inputs.input_ids, pixel_values=image)
    return outputs.logits_per_image.item()

# Load CLIP model and processor
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to("cuda")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Dummy dataset for training
inputs = torch.randn(10, 3, 256, 256)
targets = torch.randn(10, 3, 256, 256)
dataset = TensorDataset(inputs, targets)
dataloader = DataLoader(dataset, batch_size=2)
```

```python
# Training loop parameters
num_epochs = 5

# Define optimizer and loss function
optimizer = torch.optim.Adam(custom_unet_model.parameters(), lr=1e-4)
criterion = nn.MSELoss()

# Training loop
for epoch in range(num_epochs):
    for batch in dataloader:
        inputs, targets = batch
        optimizer.zero_grad()
        outputs = custom_unet_model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {loss.item()}")

# Generate and display images, and compare results
pipe = StableDiffusionPipeline.from_pretrained("CompVis/stable-diffusion-v1-4", torch_dtype=torch.float16)
pipe.to("cuda")

pipe.scheduler = DDIMScheduler.from_config(pipe.scheduler.config)
pipe.scheduler.set_timesteps(3)
```

```python
# Create a text prompt input box
text_prompt = widgets.Text(
    value='A beautiful sunset over the mountains',
    placeholder='Type your text prompt here',
    description='Text Prompt:',
    disabled=False
)
display(text_prompt)

# Button to generate image
button = widgets.Button(description="Generate Image")

output = widgets.Output()

def on_button_clicked(b):
    with output:
        output.clear_output()
        # Generate an image from the user-provided prompt
        prompt = text_prompt.value
        start_time = time.time()
        generated_image_modified = pipe(prompt).images[0]
        modified_inference_time = time.time() - start_time

        # Display the generated image
        generated_image_modified.save("generated_image.png")
        display(IPyImage(filename="generated_image.png"))
```

```python
        # Comparison function with twin-axis plot
        def compare_results(initial_time, modified_time, initial_fid, modified_fid, initial_clip, modified_clip):
            print("Comparison of Initial and Modified Model Performance:")
            print(f"Initial Inference Time: {initial_time:.2f} seconds")
            print(f"Modified Inference Time: {modified_time:.2f} seconds")
            print(f"Time Reduction: {(initial_time - modified_time) / initial_time * 100:.2f}%")

            print(f"Initial FID Score: {initial_fid:.2f}")
            print(f"Modified FID Score: {modified_fid:.2f}")
            print(f"FID Score Improvement: {(initial_fid - modified_fid):.2f}")

            print(f"Initial CLIP Score: {initial_clip:.2f}")
            print(f"Modified CLIP Score: {modified_clip:.2f}")
            print(f"CLIP Score Improvement: {(modified_clip - initial_clip):.2f}")
```

```python
            # Plotting the results
            fig, ax1 = plt.subplots()

            labels = ['Initial', 'Modified']
            x = np.arange(len(labels))

            # FID Score Bar
            fid_scores = [initial_fid, modified_fid]
            ax1.bar(x, fid_scores, color='orange', width=0.4)
            ax1.set_ylabel('FID Score')
            ax1.set_xticks(x)
            ax1.set_xticklabels(labels)
            ax1.legend(['FID Score'], loc='upper left')

            # Twin axis for Inference Time and CLIP Score
            ax2 = ax1.twinx()
            inference_times = [initial_time, modified_time]
            clip_scores = [initial_clip, modified_clip]

            ax2.plot(x, inference_times, color='blue', marker='o', label='Inference Time (s)')
            ax2.plot(x, clip_scores, color='green', marker='o', label='CLIP Score')
            ax2.set_ylabel('Inference Time (s) / CLIP Score')
            ax2.legend(loc='upper right')

            fig.tight_layout()
            plt.show()

        # Comparing the results again
        compare_results(initial_inference_time, modified_inference_time, fid_score_initial, fid_score_modified, clip_score_initial, clip_score_modified)

button.on_click(on_button_clicked)

display(button, output)
```

Fig 10 Code for final modification