Image-Text-Image Architecture

The Image-Text-Image Architecture is a powerful paradigm for manipulating and generating visual content based on textual descriptions. It creates a cyclical relationship: first, analyzing an input image to generate a textual description (Image to Text); then, allowing this text to be modified (Text Manipulation/Modification); and finally, using the modified text to generate a new image (Text to Image).

The project involved setting up an environment with key Python libraries like EasyOCR, PyTorch, transformers, and diffusers, requiring sufficient disk space and preferably a GPU. A crucial step was extracting text from images using EasyOCR's optical character recognition capabilities.

To generate textual descriptions from images, I implemented three pretrained image captioning models: Salesforce's BLIP, the ViT-GPT2 model, and Microsoft's GIT model trained on the COCO dataset. Each model offers different strengths in generating natural language descriptions.The project also includes a Visual Question Answering component using the BLIP VQA model for interactive image analysis through natural language queries.

To handle large-scale data, I integrated the COCO dataset with a custom `COCODataset` class and DataLoader for efficient processing.Completing the cycle, I explored two text-to-image generation approaches: Google's Imagen and Stability AI's Stable Diffusion model, both allowing for new image generation from textual input.

The project culminates in an end-to-end pipeline chaining these components: from initial image text extraction and captioning to generating new images based on modified text. The modular design allows for model experimentation, and I've included examples demonstrating the pipeline's effectiveness on various images.

My work on this project has reinforced my understanding that these AI tools augment creators with new capabilities, expanding the horizons of visual content generation. Understanding the underlying technology and practical building steps was key to successfully undertaking this project and harnessing its full potential.

Image-Text-Image Architecture

Installation

Create a new notebook file in Jupyter Notebook or Google Colab

Import required python libraries down below. If you do not have the libraries installed on your machine please install using pip.

```
import easyocr
import cv2
from matplotlib import pyplot as plt
import numpy as np
from google import genai
from google.genai import types
from PIL import Image
from io import BytesIO
import base64
import requests
import os
import time
from PIL import Image as PILImage

import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from transformers import BlipProcessor, BlipForConditionalGeneration,
BlipForQuestionAnswering
from transformers import VisionEncoderDecoderModel, ViTImageProcessor,
AutoTokenizer
from transformers import AutoProcessor, AutoModelForCausalLM

from diffusers import StableDiffusionPipeline

from pycocotools.coco import COCO
```

Text Recognition and Extraction (EasyOCR)

EasyOCR provides powerful optical character recognition capabilities for extracting text from images. The implementation uses the extract_text_from_image function that accepts an image path and returns detected text along with bounding box coordinates.

The results can be visualized using OpenCV to draw rectangles around detected text regions and display the processed image with matplotlib.

Create a function the extracts text from an image using EasyOCR

```
def extract_text_from_image(image_path, languages=['en']):
 """Extracts text from an image using EasyOCR.

 Args:
   image_path: The path to the image file.
   languages: A list of language codes to use for OCR. Defaults to English
('en').

 Returns:
   A list of tuples, where each tuple contains the extracted text, bounding box
coordinates, and confidence score.
 """
 reader = easyocr.Reader(languages)
 result = reader.readtext(image_path)
 return result
```

Define the image path and determine the language that you would like to use and pass in as arguments to the function. In addition, you should save the result to a variable to use for further processing.

```
IMAGE_PATH = 'quote_image.png'
extracted_text = extract_text_from_image(IMAGE_PATH)
print(extracted_text)
```

It is recommended to print the result to see the structure, which should look like the following:

```
[([[np.int32(96), np.int32(381)],
   [np.int32(326), np.int32(381)],
   [np.int32(326), np.int32(472)],
   [np.int32(96), np.int32(472)]],
  '"You',
  np.float64(0.3567149341106415)),
 ([[np.int32(343), np.int32(395)],
   [np.int32(729), np.int32(395)],
   [np.int32(729), np.int32(488)],
   [np.int32(343), np.int32(488)]],
  'are your',
  np.float64(0.9356462700945349)),
 ([[np.int32(200), np.int32(478)],
   [np.int32(726), np.int32(478)],
   [np.int32(726), np.int32(605)],
   [np.int32(200), np.int32(605)]],
  "best thing:'",
  np.float64(0.8184738606310134)),
 ([[np.int32(249), np.int32(689)],
   [np.int32(633), np.int32(689)],
   [np.int32(633), np.int32(733)],
   [np.int32(249), np.int32(733)]],
  '~TONI MORRISON',
  np.float64(0.9974949575745765)),
 ([[np.int32(344), np.int32(736)],
   [np.int32(534), np.int32(736)],
   [np.int32(534), np.int32(778)],
   [np.int32(344), np.int32(778)]],
  'BELOVED',
  np.float64(0.999983734424943))]
```

Create a function to display the image using the results from EasyOCR

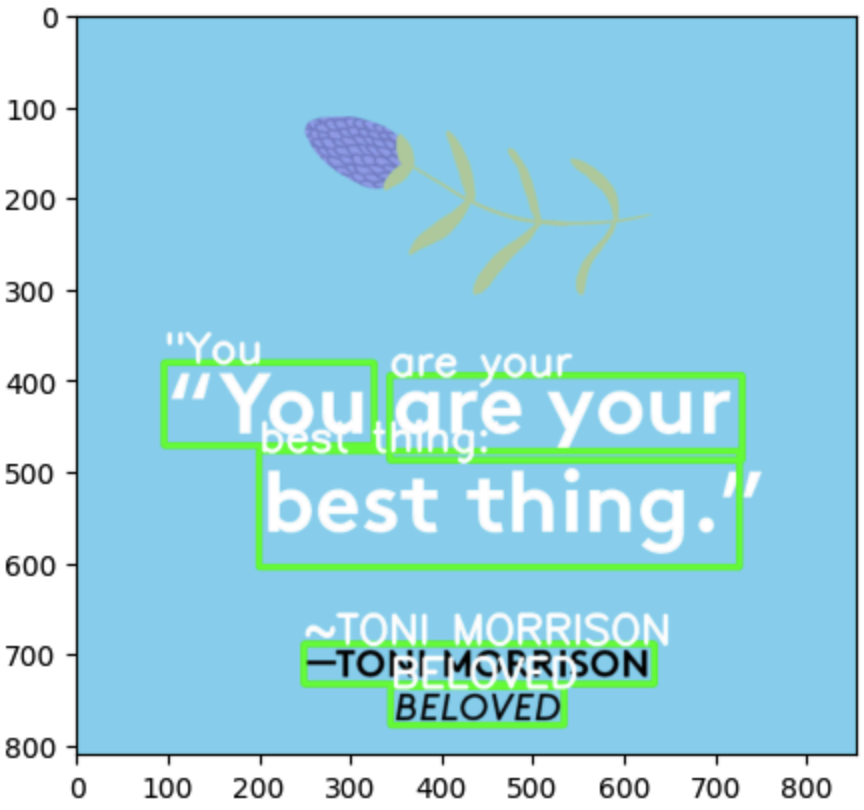
```
def display_image_with_text(image_path, result):
 """Displays the image with detected text and bounding boxes.
```

```
  Args:
    image_path: The path to the image file.
    result: The output from extract_text_from_image function.
  """
  img = cv2.imread(image_path)
  for detection in result:
    top_left = tuple([int(val) for val in detection[0][0]])
    bottom_right = tuple([int(val) for val in detection[0][2]])
    text = detection[1]
    font = cv2.FONT_HERSHEY_SIMPLEX
    img = cv2.rectangle(img, top_left, bottom_right, (0, 255, 0), 5)
    img = cv2.putText(img, text, top_left, font, 1.5, (255, 255, 255), 3,
cv2.LINE_AA)

  plt.imshow(img)
  plt.show()
```

The arguments for this function will be the image_path and the output from the extract_text_from_image function.

```
display_image_with_text(IMAGE_PATH, extracted_text)
```

When you invoke the function, you should see the image you used with bounding boxes and the detected text.



Afterward, you can combine the text to use as the prompt for the text-image models Imagen and Stable Diffusion.

```
combined_text = ' '.join([detection[1] for detection in extracted_text])
print(combined_text)
```

Text-to-Image Generation

Two different approaches are used for generating images from text:

Google's Imagen model through their API and Stability AI's Stable Diffusion model. Both implementations allow for converting textual descriptions back into images, completing the image-text-image cycle.

You will need API keys for both and will need to request access via the following:

Imagen API Access - https://console.cloud.google.com/

Stable Diffusion Access - https://platform.stability.ai/account/keys

After you have obtained access to one or both of the APIs, you can create functions that will generate and display images using Google's Imagen and Stability AI's Stable Diffusion models.

Imagen model Function

```
def generate_and_display_images(api_key, caption, num_images=4):
 """Generates and displays images using Google's Imagen model.

 Args:
   api_key: Your Google Cloud API key.
   caption: The text prompt to use for image generation.
   num_images: The number of images to generate.
 """
 client = genai.Client(api_key=api_key)

 response = client.models.generate_images(
     model='imagen-3.0-generate-002',
     prompt=caption,
     config=types.GenerateImagesConfig(
         number_of_images=num_images,
     )
 )
 for generated_image in response.generated_images:
   image = Image.open(BytesIO(generated_image.image.image_bytes))
   image.show() #To see the output, run the code.
```

```
 return
Image.open(BytesIO(response.generated_images[0].image.image_bytes))
```

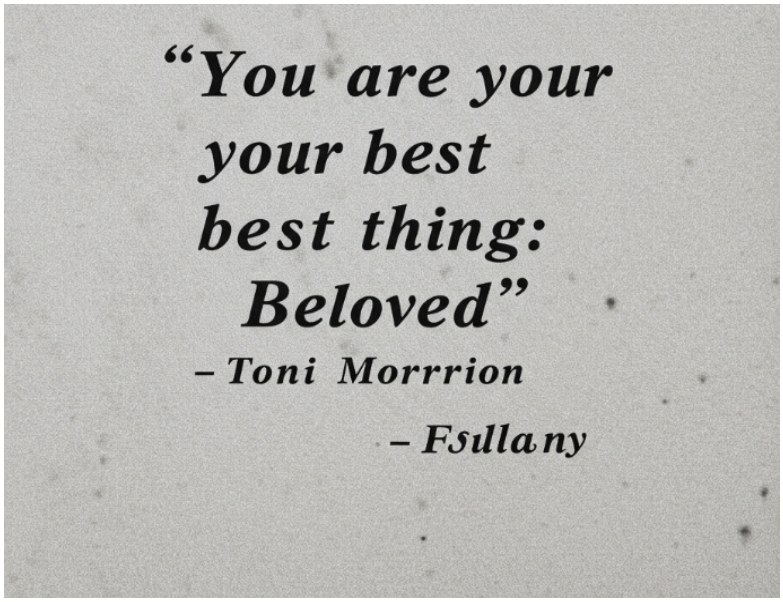
Stable Diffusion Model Function

```
def generate_and_display_image_stability_ai(api_key, combined_text, width=1024,
height=1024, steps=40, cfg_scale=5, seed=0, samples=1):
 """Generates and displays an image using Stability AI's Stable Diffusion API.

 Args:
   api_key: Your Stability AI API key.
   combined_text: The text prompt to use for image generation.
   width: The width of the generated image.
   height: The height of the generated image.
   steps: The number of inference steps.
   cfg_scale: The classifier-free guidance scale.
   seed: The random seed.
   samples: The number of images to generate.
 """
 url =
"https://api.stability.ai/v1/generation/stable-diffusion-xl-1024-v1-0/text-to-i
mage"

 body = {
     "steps": steps,
     "width": width,
     "height": height,
     "seed": seed,
     "cfg_scale": cfg_scale,
     "samples": samples,
     "text_prompts": [
         {
             "text": combined_text,
             "weight": 1
         }
     ],
 }

 headers = {
     "Accept": "application/json",
     "Content-Type": "application/json",
     "Authorization": f"Bearer {api_key}", # Include API key in Authorization
```

```
header
 }

 response = requests.post(url, headers=headers, json=body)

 if response.status_code != 200:
     raise Exception("Non-200 response: " + str(response.text))

 data = response.json()

 # Make sure the out directory exists
 if not os.path.exists("./out"):
     os.makedirs("./out")

 # Generate a unique filename using timestamp
 timestamp = int(time.time())
 filename = f"./out/txt2img_{timestamp}.png"

 for i, image in enumerate(data["artifacts"]):
     image_data = base64.b64decode(image["base64"])
     image = PILImage.open(BytesIO(image_data))
     image.save(filename)
     display(image) #To see the output, run the code.
```

You can use the text that was extracted from the image using EasyOCR or any of the captioning models that will be covered later in this tutorial.

```
generate_and_display_images('Imagen API Key', combined_text, num_images=2)
```




```
generate_and_display_image_stability_ai(
    api_key="Stable Diffusion API Key",
    combined_text=combined_text
)
```

Image Captioning Models

Three different pretrained models are implemented for image captioning:

Several pretrained models excel at generating natural language descriptions from images. These include Salesforce's BLIP model, known for its detailed captions; ViT-GPT2, which integrates vision transformers with language models; and Microsoft's GIT (Generative Image-to-Text) model, trained on the COCO dataset. Each model utilizes a unique approach to effectively describe visual content.

Next steps involve creating functions to encapsulate the functionalities of these pretrained models. This includes steps for using the model's processor, initializing the model, accessing image data from files or URLs, and generating the image captions.

Blip Model Function:

```
def generate_caption_blip(image_url_or_path):
    """Generates a caption for an image using the BLIP model.

    Args:
        image_url_or_path: The URL or local path to the image.

    Returns:
        The generated caption as a string.
    """
    processor =
BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
    model =
BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-
base")

    device = "cuda" if torch.cuda.is_available() else "cpu"
    model.to(device)
    print(f"Using device: {device}")

    try:
```

```
        # Attempt to open as URL first
        raw_image = Image.open(requests.get(image_url_or_path,
stream=True).raw).convert('RGB')
        image_source = image_url_or_path
    except:
        # If URL fails, try opening as local file
        raw_image = Image.open(image_url_or_path).convert('RGB')
        image_source = image_url_or_path

    print(f"Loaded image from: {image_source}")

    inputs = processor(raw_image, return_tensors="pt").to(device)

    print("Generating caption...")
    out = model.generate(**inputs)

    caption = processor.decode(out[0], skip_special_tokens=True)

    print("\nGenerated Description:")
    print(caption)

    return caption  # Return the caption
```

Similar to the EasyOCR implementation, you can specify an `image_url` or `image_path`, call the function, and store the output in a variable. This variable can then be used as a prompt for your text-to-image model(s).

```
image_url =
'https://images.unsplash.com/photo-1511465390398-532913e8328d?q=80&w=2064&auto=
format&fit=crop&ixlib=rb-4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx
8fA%3D%3D'
caption = generate_caption_blip(image_url)
print(caption)  # Print the returned caption
```

The output with the generated caption should look something like this:

```
Using a slow image processor as `use_fast` is unset and a slow processor was saved with
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
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://
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 model
  warnings.warn(
preprocessor_config.json: 100% 287/287 [00:00<00:00, 24.0kB/s]
tokenizer_config.json: 100% 506/506 [00:00<00:00, 28.9kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 1.89MB/s]
tokenizer.json: 100% 711k/711k [00:00<00:00, 10.6MB/s]
special_tokens_map.json: 100% 125/125 [00:00<00:00, 9.57kB/s]
config.json: 100% 4.56k/4.56k [00:00<00:00, 369kB/s]
pytorch_model.bin: 100% 990M/990M [00:07<00:00, 93.7MB/s]
model.safetensors: 100% 990M/990M [00:11<00:00, 49.4MB/s]
Using device: cpu
Loaded image from: https://images.unsplash.com/photo-1511465390398-532913e8328d?q=80&w=2
Generating caption...

Generated Description:
a brick wall with a sign that says i make something special
```

We can now use our text-image models with the generated caption.

Imagen with BLIP Result:

```
generate_and_display_images('Imagen API Key', caption, num_images=2)
```

Stable Diffusion with BLIP Result:

```
generate_and_display_image_stability_ai(
    api_key="Stable Diffusion API Key",
    combined_text=caption
)
```

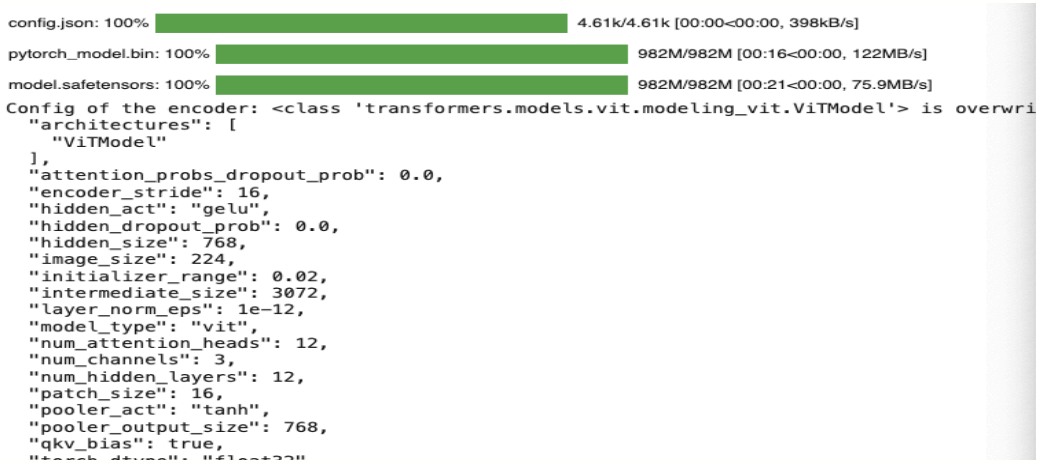
Vit GPT2 Model Function:

```
from PIL import Image
import requests
from transformers import VisionEncoderDecoderModel, ViTImageProcessor,
AutoTokenizer

def generate_caption_vit_gpt2(image_url_or_path):
    """Generates a caption for an image using the
'nlpconnect/vit-gpt2-image-captioning' model.

    Args:
        image_url_or_path: The URL or local path to the image.

    Returns:
        The generated caption as a string.
    """
    # --- Model Loading ---
    model =
VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning
")
    feature_extractor =
ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
    tokenizer =
AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

    # Set model configuration for text generation
    model.config.decoder_start_token_id = tokenizer.cls_token_id
    model.config.pad_token_id = tokenizer.pad_token_id
    model.config.vocab_size = model.config.decoder.vocab_size
    model.config.max_length = 16
    model.config.num_beams = 4
    model.config.early_stopping = True
    model.config.no_repeat_ngram_size = 2
    model.config.length_penalty = 2.0

    # --- Image Loading ---
    try:
        # Attempt to open as URL first
        image = Image.open(requests.get(image_url_or_path,
stream=True).raw).convert("RGB")
        image_source = image_url_or_path
    except:
        # If URL fails, try opening as local file
```

```
        image = Image.open(image_url_or_path).convert("RGB")
        image_source = image_url_or_path

    print(f"Successfully loaded image from {image_source}")

    # --- Caption Generation ---
    pixel_values = feature_extractor(images=image,
return_tensors="pt").pixel_values
    generated_ids = model.generate(pixel_values=pixel_values)
    generated_text = tokenizer.batch_decode(generated_ids,
skip_special_tokens=True)[0]

    print("Generated Caption:", generated_text)
    return generated_text
```

Define the image URL or path, call the model function, and store the resulting caption in a variable for use with text-image models.

```
image_url = "http://images.cocodataset.org/val2017/000000039769.jpg"
caption = generate_caption_vit_gpt2(image_url)
print(caption)
```



```
preprocessor_config.json: 100% 228/228 [00:00<00:00, 18.6kB/s]
tokenizer_config.json: 100% 241/241 [00:00<00:00, 14.0kB/s]
vocab.json: 100% 798k/798k [00:00<00:00, 6.00MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 1.70MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 11.2MB/s]
special_tokens_map.json: 100% 120/120 [00:00<00:00, 1.90kB/s]
/usr/local/lib/python3.11/dist-packages/transformers/generation/utils.py:1667: UserWarn
  warnings.warn(
The attention mask is not set and cannot be inferred from input because pad token is sa
Successfully loaded image from http://images.cocodataset.org/val2017/000000039769.jpg
We strongly recommend passing in an `attention_mask` since your input_ids may be padded
You may ignore this warning if your `pad_token_id` (50256) is identical to the `bos_tok
Generated Caption: a cat laying on top of a couch next to another cat
```

After the model completes its run, the generated caption will be displayed at the end of the output. For example, in this instance, the output reads: "a cat laying on top of a couch next to another cat."

With the generated caption now available, we can proceed to execute our image-text models.

Imagen with Vit GPT2 Result:

```
generate_and_display_images('Imagen API Key', caption, num_images=2)
```

Stable Diffusion with Vit GPT2 Result:

```
generate_and_display_image_stability_ai(
    api_key="Stable Diffusion API KEY",
    combined_text=caption
)
```

GIT Model Function:

```
def generate_caption_git(image_url_or_path):
    """Generates a caption for an image using the 'microsoft/git-large-coco'
model.

    Args:
        image_url_or_path: The URL or local path to the image.

    Returns:
        The generated caption as a string.
    """
    # Load GIT model and processor
    processor = AutoProcessor.from_pretrained("microsoft/git-large-coco")
    model = AutoModelForCausalLM.from_pretrained("microsoft/git-large-coco")

    device = "cuda" if torch.cuda.is_available() else "cpu"  # Determine device
    model.to(device)

    # Load the image
    try:
```

```
        # Attempt to open as URL first
        image = Image.open(requests.get(image_url_or_path,
stream=True).raw).convert("RGB")
    except:
        # If URL fails, try opening as local file
        image = Image.open(image_url_or_path).convert("RGB")

    # Remove the line causing the error:
    # image = image.to(device) #Move image to device

    # Instead, move the pixel_values tensor to the device after processing:
    pixel_values = processor(images=image,
return_tensors="pt").pixel_values.to(device)


    # Generate caption
    generated_ids = model.generate(
        pixel_values=pixel_values,
        max_length=50,
        num_beams=4
    )

    generated_text = processor.batch_decode(generated_ids,
skip_special_tokens=True, legacy=False)[0]
    print("GIT Caption:", generated_text)
    return generated_text
```

Define the image URL or path, call the model function, and store the resulting caption in a variable for use with text-to-image models.

```
image_url = "http://images.cocodataset.org/val2017/000000039769.jpg"  # Or
local file path
caption = generate_caption_git(image_url)
print(caption)
```

The output for the image that we received is:

```
GIT Caption: two cats sleeping on a couch next to a remote control.
```

Imagen Model with GIT Result:

```
generate_and_display_images('Imagen API Key', caption, num_images=2)
```

Stable Diffusion with GIT Result:

```
generate_and_display_image_stability_ai(
   api_key="Stable Diffusion API Key",
   combined_text=caption
)
```

Visual Question Answering (VQA)

The VQA section demonstrates how to perform question-answering tasks on images using the BLIP VQA model. You can input an image and ask questions about its contents, and the model will generate relevant answers. This provides an interactive way to analyze image content through natural language queries.

```
# Load model and processor
processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")

# Load sample image
img_url = 'https://cdn.pixabay.com/photo/2025/05/04/17/47/dog-9578735_1280.jpg'
raw_image = Image.open(requests.get(img_url, stream=True).raw).convert('RGB')
```

```python
# Define question and process inputs
question = "What color is the woman's jacket?"
inputs = processor(raw_image, question, return_tensors="pt")

# Generate answer
out = model.generate(**inputs)
# Convert the tensor to a list of token IDs
decoded_ids = out.squeeze(0).tolist()  # Remove batch dimension and convert to list
# Decode the token IDs into text
answer = processor.decode(decoded_ids, skip_special_tokens=True)

print(f"Q: {question}")
print(f"A: {answer}")
```

Output:

```
Q: What color is the woman's jacket?

A: pink
```

COCO Dataset Integration

The project includes a robust implementation for handling the COCO dataset, one of the largest image-caption paired datasets. A custom COCODataset class is implemented to load and process images and their associated captions. The DataLoader provides efficient batching and iteration over the dataset for training or inference.

```
# Download and setup COCO dataset
!mkdir -p /content/data/coco
!wget http://images.cocodataset.org/zips/val2017.zip -P /content/data/coco/
!wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip -P /content/da

# Unzip the files
!unzip /content/data/coco/val2017.zip -d /content/data/coco/
!unzip /content/data/coco/annotations_trainval2017.zip -d /content/data/coco/

# Clean up zip files to save space
!rm /content/data/coco/*.zip
```

```
# Configure paths for COCO dataset
COCO_DIR = '/content/data/coco/'
IMAGE_DIR = os.path.join(COCO_DIR, 'images')
ANNOTATION_FILE = os.path.join(COCO_DIR, 'annotations/captions_train2017.json')

# Initialize COCO API for caption annotations
try:
    coco = COCO(ANNOTATION_FILE)
    print('COCO dataset loaded successfully')
except Exception as e:
    print(f'Error loading COCO dataset: {e}')
```

Define class for COCO Dataset

```
class COCODataset(Dataset):
    def __init__(self, coco, image_dir, transform=None):
        self.coco = coco
        self.image_dir = image_dir
        self.transform = transform
        self.ids = list(self.coco.imgs.keys())

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

    def __getitem__(self, idx):
        img_id = self.ids[idx]
        img_info = self.coco.loadImgs(img_id)[0]
        image = Image.open(os.path.join(self.image_dir,
img_info['file_name'])).convert('RGB')
```

```
        if self.transform:
            image = self.transform(image)

        # Get annotations (captions)
        ann_ids = self.coco.getAnnIds(imgIds=img_id)
        annotations = self.coco.loadAnns(ann_ids)
        captions = [ann['caption'] for ann in annotations]

        return image, captions
```

Configure paths for COCO dataset, initialize API for caption annotations, define image transformations, create dataset instance and data loader.

```
# Configure paths for COCO dataset
COCO_DIR = '/content/data/coco'
IMAGE_DIR = os.path.join(COCO_DIR, 'val2017')  # Using validation set as
it's smaller
ANNOTATION_FILE = os.path.join(COCO_DIR,
'annotations/captions_val2017.json')

# Initialize COCO API for caption annotations
try:
   coco = COCO(ANNOTATION_FILE)
   print('COCO dataset loaded successfully')
except Exception as e:
   print(f'Error loading COCO dataset: {e}')

# Define image transformations
transform = transforms.Compose([
   transforms.Resize((224, 224)),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])

# Create dataset instance
dataset = COCODataset(coco, IMAGE_DIR, transform=transform)
```

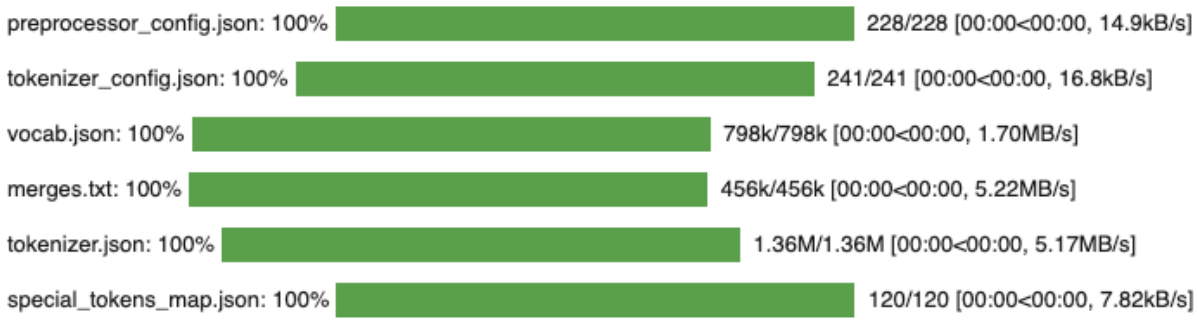
```
# Create data loader
data_loader = DataLoader(dataset, batch_size=1, shuffle=True)
```

Load the pre-trained captioning model

```
# Load pre-trained image captioning model
def load_captioning_model():
    model_name = "nlpconnect/vit-gpt2-image-captioning"
    model = VisionEncoderDecoderModel.from_pretrained(model_name)
    feature_extractor = ViTImageProcessor.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    return model, feature_extractor, tokenizer

# Initialize image captioning models
print('Loading image captioning model...')
caption_model, feature_extractor, tokenizer = load_captioning_model()
print('Model loaded successfully')
```

You should see the model initializing and a printed message that the Model loaded successfully.

```
        "max_length": 50
      }
    },
    "torch_dtype": "float32",
    "transformers_version": "4.51.3",
    "use_cache": true,
    "vocab_size": 50257
  }
```



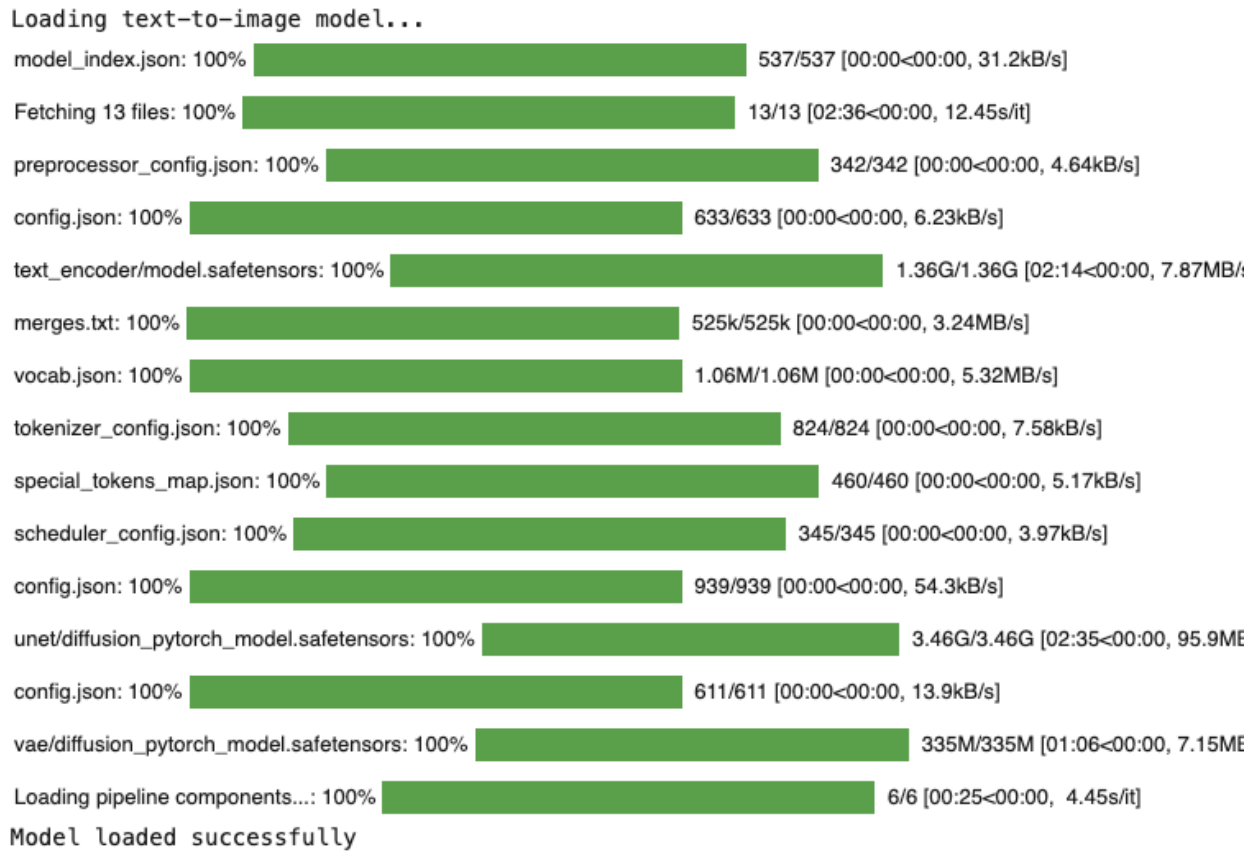

```
Model loaded successfully
```

Next we have to load the Stable Diffusion pipeline:

```python
# Load Stable Diffusion pipeline
def load_text2img_model():
    model_id = "stabilityai/stable-diffusion-2-1"
    pipe = StableDiffusionPipeline.from_pretrained(
        model_id,
        torch_dtype=torch.float16,
        use_safetensors=True
    )
    # Check if GPU is available, otherwise default to CPU
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    pipe.to(device)

    # Only enable offload strategies if on CUDA
    if device.type == "cuda":
        pipe.enable_attention_slicing()
        pipe.enable_model_cpu_offload()  # Offload to CPU if on a CUDA device
        pipe.enable_sequential_cpu_offload()

    return pipe
```

```
print('Loading text-to-image model...')
text2img_pipe = load_text2img_model()
print('Model loaded successfully')
```

You should see the model initializing and a printed message that the Model loaded successfully.



This function is really important and brings image-text and text-image together while generating the images that we will see.

```
def process_image_to_image(image, caption_model, feature_extractor,
tokenizer, text2img_pipe):
    # 1. Generate caption
    with torch.no_grad():  # Reduce memory usage
        # Move the image to the same device as the model
        device = caption_model.device  # Get the model's device
        image = image.to(device)
```

```
        # Convert the normalized tensor back to PIL Image for the
feature extractor
        # First denormalize
        image_denorm = image * torch.tensor([0.229, 0.224,
0.225]).to(device)[:, None, None] + \
                       torch.tensor([0.485, 0.456,
0.406]).to(device)[:, None, None]
        # Clip values to be in the range [0, 1]
        image_denorm = torch.clamp(image_denorm, 0, 1)

        # Convert to PIL Image
        image_cpu = image_denorm.cpu()
        image_pil = transforms.ToPILImage()(image_cpu)

        print("Processing image with shape:", image.shape)

        # Process with feature extractor (it will handle the
normalization internally)
        pixel_values = feature_extractor(images=image_pil,
return_tensors="pt").pixel_values.to(device)
        print("Pixel values shape:", pixel_values.shape)
        print("Pixel values range:", pixel_values.min().item(), "to",
pixel_values.max().item())

        generated_ids = caption_model.generate(
            pixel_values,
            max_length=30,
            num_beams=4,
            early_stopping=True,
            repetition_penalty=1.5,  # Penalize repetition more
strongly
            length_penalty=1.0,
            num_return_sequences=1,
            bad_words_ids=[[tokenizer.encode("black and white",
add_special_tokens=False)[0]]],  # Prevent "black and white" from
appearing
        )
        generated_caption = tokenizer.batch_decode(generated_ids,
```

```
skip_special_tokens=True)[0]
        print("Generated caption:", generated_caption)

    # Generate new image from caption
    generated_image = text2img_pipe(
        generated_caption,
        num_inference_steps=20,      # Reduced from 30
        guidance_scale=7.0,
        height=512,                  # Reduced from default 768
        width=512,
        use_memory_efficient_attention=True
    ).images[0]

    return generated_caption, generated_image
```

Testing the pipeline for a single image:

```
for images, _ in data_loader:
   # Process first image
   image = images[0]
   caption, new_image = process_image_to_image(
       image,
       caption_model,
       feature_extractor,
       tokenizer,
       text2img_pipe
   )

   # Display results
   plt.figure(figsize=(15, 5))

   plt.subplot(1, 3, 1)
   plt.imshow(image.permute(1, 2, 0))  # Permute to (H, W, C) for display
   plt.title('Original Image')
   plt.axis('off')

   plt.subplot(1, 3, 2)
   plt.text(0.5, 0.5, caption, ha='center', va='center', wrap=True)
   plt.title('Generated Caption')
   plt.axis('off')
```

```
    plt.subplot(1, 3, 3)
    plt.imshow(new_image)
    plt.title('Generated Image')
    plt.axis('off')

    plt.show()
    break  # Process only one image for testing
```

Image-Text-Image Model results for a single image:

Testing the pipeline for 5 images:

```
# Test the pipeline for 5 images
for i, (images, _) in enumerate(data_loader):
    if i >= 5:  # Stop after processing 5 images
        break

    # Process first image
    image = images[0]
    caption, new_image = process_image_to_image(
        image,
        caption_model,
        feature_extractor,
```

```
        tokenizer,
        text2img_pipe
    )

    # Display results
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(image.permute(1, 2, 0))  # Permute to (H, W, C) for
display
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.text(0.5, 0.5, caption, ha='center', va='center', wrap=True)
    plt.title('Generated Caption')
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(new_image)
    plt.title('Generated Image')
    plt.axis('off')

    plt.show()
```

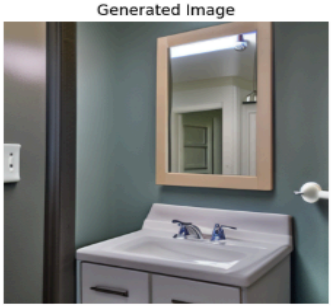
Image-Text-Image Model results for 5 images:

Generated Caption

Original Image

a bathroom with a toilet and a sink

Generated Image



```
Processing image with shape: torch.Size([3, 224, 224])
Pixel values shape: torch.Size([1, 3, 224, 224])
Pixel values range: -1.0 to 1.0
Generated caption: a red brick building with a neon sign on it
```

100% 20/20 [00:18<00:00, 1.06it/s]

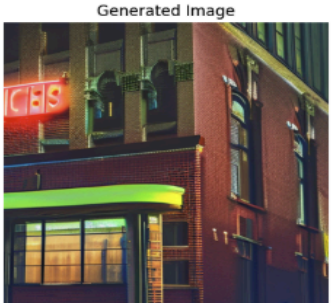
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([
```

Generated Caption

Original Image

a red brick building with a neon sign on it

Generated Image



Generated Caption

Original Image

a man in a baseball uniform holding a baseball bat

Generated Image

Generated Caption

Original Image

a beautiful young woman holding a tennis racquet

Generated Image

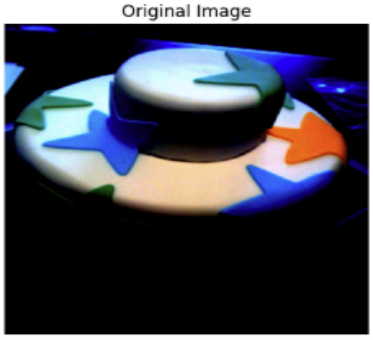
```
Processing image with shape: torch.Size([3, 224, 224])
Pixel values shape: torch.Size([1, 3, 224, 224])
Pixel values range: -1.0 to 0.9058823585510254
Generated caption: a cake shaped like a train on top of a table
```

100% 20/20 [00:19<00:00, 1.06it/s]

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([
```

Generated Caption

Original Image



a cake shaped like a train on top of a table

Generated Image