AI in Industry 4.0: Assignment 1

Harshveer Singh Thind *SE21UARI047*
Siddharth Reddy A *SE21UARI020*
Dhruv Arora *SE21UARI035*

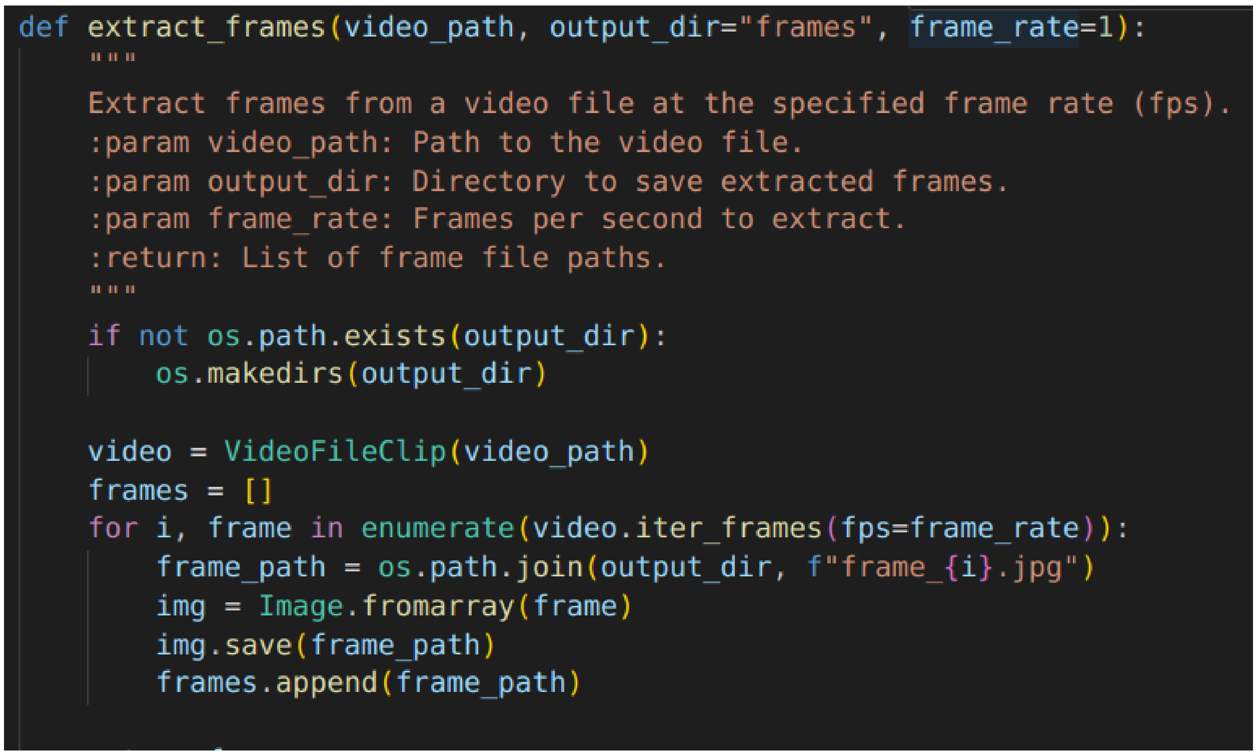
Tools Used

Interface - Gradio
Caption Generation - *moviepy* and *huggingface* transformers library (Blip Model)
Video Generation - huggingface stable diffusion models
Hardware Requirements - T4 gpu from google colab

Step 1

Extract Frames

Used iter_frames method of VideoFileClip to extract frames at 1 fps

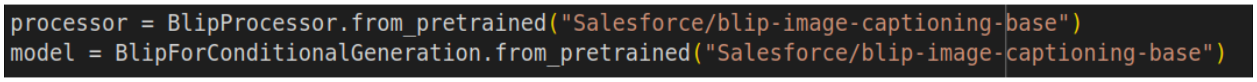


```python
def extract_frames(video_path, output_dir="frames", frame_rate=1):
    """
    Extract frames from a video file at the specified frame rate (fps).
    :param video_path: Path to the video file.
    :param output_dir: Directory to save extracted frames.
    :param frame_rate: Frames per second to extract.
    :return: List of frame file paths.
    """
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    video = VideoFileClip(video_path)
    frames = []
    for i, frame in enumerate(video.iter_frames(fps=frame_rate)):
        frame_path = os.path.join(output_dir, f"frame_{i}.jpg")
        img = Image.fromarray(frame)
        img.save(frame_path)
        frames.append(frame_path)
```

Step 2

Generate Captions Using AI Model

Captions_helper function uses the `BlipProcessor`, `BlipForConditionalGeneration` from the huggingface transformers module to generate the captions
The model and processor are defined here as the model and processor from Salesforce



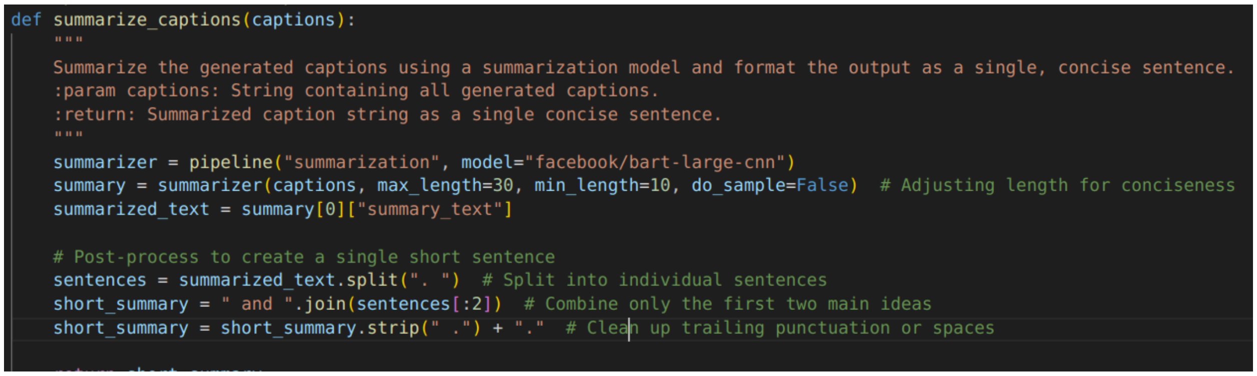
```python
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")
```

The `BlipProcessor` preprocesses the frames, and the `BlipForConditionalGeneration` model generates captions.

Step 3

Summarizing the Captions

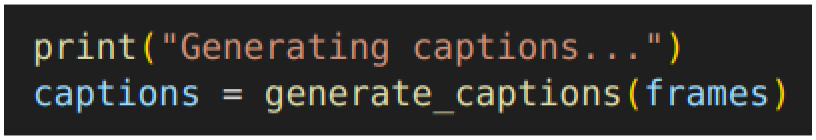
For the easier and quicker execution of the video generation model we summarize the captions generated using bart-large-cnn from facebook



```python
def summarize_captions(captions):
    """
    Summarize the generated captions using a summarization model and format the output as a single, concise sentence.
    :param captions: String containing all generated captions.
    :return: Summarized caption string as a single concise sentence.
    """
    summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
    summary = summarizer(captions, max_length=30, min_length=10, do_sample=False)  # Adjusting length for conciseness
    summarized_text = summary[0]["summary_text"]

    # Post-process to create a single short sentence
    sentences = summarized_text.split(". ")  # Split into individual sentences
    short_summary = " and ".join(sentences[:2])  # Combine only the first two main ideas
    short_summary = short_summary.strip(" .") + "."  # Clean up trailing punctuation or spaces
```

Step 4

generate_captions function will be used to call all the above functions and then will return captions.

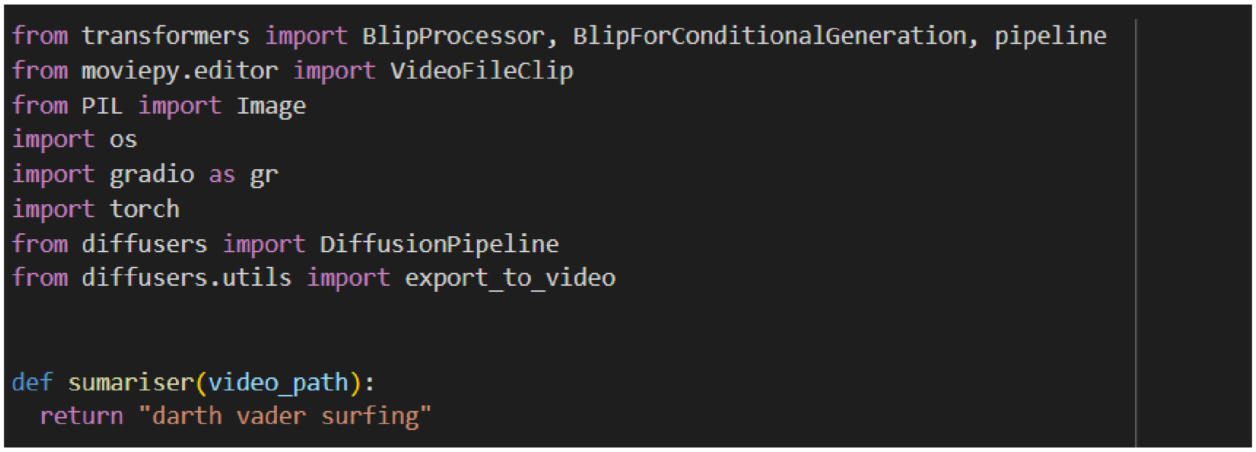


```python
print("Generating captions...")
captions = generate_captions(frames)
```

Step 5

The first part defines a function `summariser(video_path)` that takes the path to a video as input and returns a descriptive caption (e.g., "Darth Vader surfing"). While the function currently returns a static example caption, the intent is to use this function to analyze the content of the input video and generate meaningful text descriptions automatically.

Key purpose: This step can act as a preprocessing step if the user uploads a video, summarizing its content into a caption that can later be used to generate an AI video.



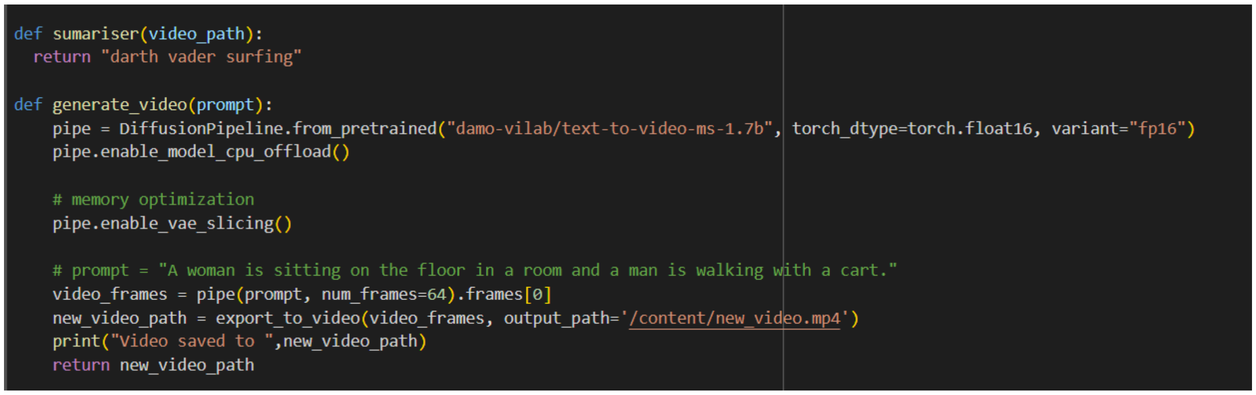
```
from transformers import BlipProcessor, BlipForConditionalGeneration, pipeline
from moviepy.editor import VideoFileClip
from PIL import Image
import os
import gradio as gr
import torch
from diffusers import DiffusionPipeline
from diffusers.utils import export_to_video


def sumariser(video_path):
  return "darth vader surfing"
```

Step 6

The `generate_video(prompt)` function uses a **text-to-video generation pipeline** powered by the `diffusers` library.

- **Pipeline Setup**: It loads a pretrained model, `damo-vilab/text-to-video-ms-1.7b`, optimized with:
  - **Model offloading**: Reduces GPU memory usage by moving unused parts of the model to the CPU.
  - **VAE slicing**: Splits the Variational Autoencoder (VAE) for memory-efficient processing.
- **Prompt-driven generation**: Based on a text prompt (e.g., "Surfing Darth Vader"), the model creates a video consisting of 64 frames.
- **Exporting to video**: The generated video frames are saved to `/content/new_video.mp4` for further use.

Key purpose: This is the core step where a natural language prompt is converted into a synthetic video using state-of-the-art diffusion technology.



```python
def sumariser(video_path):
  return "darth vader surfing"

def generate_video(prompt):
    pipe = DiffusionPipeline.from_pretrained("damo-vilab/text-to-video-ms-1.7b", torch_dtype=torch.float16, variant="fp16")
    pipe.enable_model_cpu_offload()

    # memory optimization
    pipe.enable_vae_slicing()

    # prompt = "A woman is sitting on the floor in a room and a man is walking with a cart."
    video_frames = pipe(prompt, num_frames=64).frames[0]
    new_video_path = export_to_video(video_frames, output_path='/content/new_video.mp4')
    print("Video saved to ",new_video_path)
    return new_video_path
```

Step 7

The `move_and_rename_file()` function organizes the generated output by:

- Moving the generated video file (`/content/new_video.mp4`) to a specific directory (`/tmp/gradio/...`).
- Renaming it to a standard name (`new_video.mp4`).

This ensures that the generated video is easily accessible, consistently named, and stored in a structured location.

Key purpose: Organize the output video into a pre-defined directory for seamless integration with the Gradio interface or other modules.

```python
import shutil
import os

def move_and_rename_file(src_path, dest_dir, new_name):

    # Ensure the destination directory exists
    if not os.path.exists(dest_dir):
        os.makedirs(dest_dir)

    # Construct the destination path with the new name
    dest_path = os.path.join(dest_dir, new_name)

    # Move and rename the file
    shutil.move(src_path, dest_path)
    print(f"File moved and renamed to: {dest_path}")

    return dest_path
```

Step 8

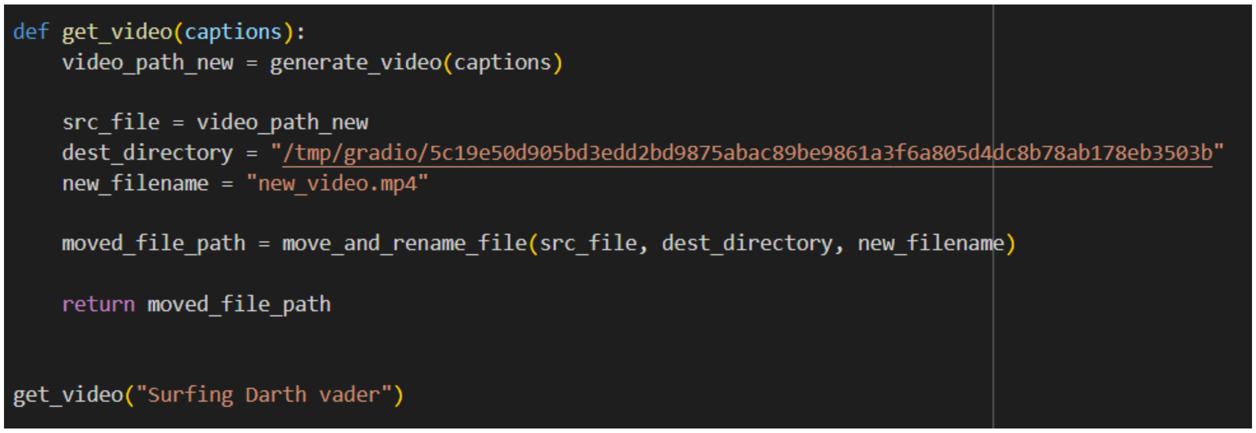
The `get_video(captions)` function ties the previous steps together:

- It generates a video from the provided text caption by calling `generate_video(captions)`.
- It then moves and renames the video for consistent handling.

In a broader context, the commented-out code hints at further functionality using Gradio, where:

- Users could upload an input video.
- The `summariser()` function would extract captions describing the video.
- The `generate_video()` and `move_and_rename_file()` steps would use the captions to produce an AI-generated video.
- Outputs like the original video, generated video, and caption would be displayed interactively.

Key purpose: Build a pipeline to seamlessly integrate video summarization, caption-based video generation, and an interactive interface.



```python
def get_video(captions):
    video_path_new = generate_video(captions)

    src_file = video_path_new
    dest_directory = "/tmp/gradio/5c19e50d905bd3edd2bd9875abac89be9861a3f6a805d4dc8b78ab178eb3503b"
    new_filename = "new_video.mp4"

    moved_file_path = move_and_rename_file(src_file, dest_directory, new_filename)

    return moved_file_path


get_video("Surfing Darth vader")
```

Step 9

Preprocess_video function will be calling the generate_captions and generate_video functions and return the captions, the generated video in mp4 format and the original video path.
This functions will be given to gradio as fn parameter

Step 10

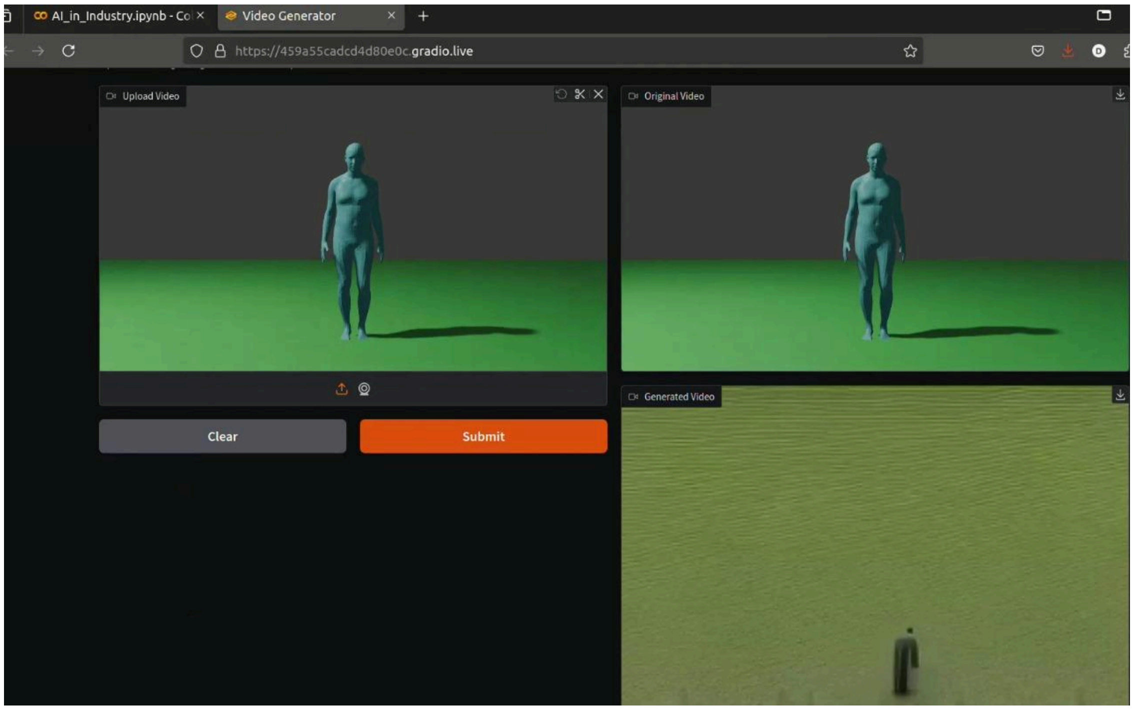
Interface
Input will be an object of gr.Videowhich will allow to upload a video by drag and drop or click and upload the same

There are 3 outputs of 2 types, 2 of gr.Video and one of gr.Textbox. The two video outputs would be the original video and the generated video and the Textbox output will be the captions generated for the original video.

How to Use

First run the code, then you'll get the interface of gradio. If you are using colab, it will run directly or you can open it in another window. Drag and drop a file, or click to upload. Captions will be generated which will be used to generate an AI video. The interface will show the original video, generated video as well as the captions that were generated for the original video.

Outputs to Expect


AI_in_Industry.ipynb - Co
Video Generator
https://459a55cadcd4d80e0c.gradio.live
Upload Video
Original Video
Generated Video
Clear
Submit