1. **Installing Required Packages in Python**

```
!pip install torch torchvision torchaudio transformers opencv-python-headless
   moviepy pillow reportlab yt-dlp google-api-python-client nbformat nbconvert
   psutil
```

This installs the required libraries using pip. These include:

- `torch`, `torchvision`, `torchaudio`: PyTorch and related modules for deep learning and audio processing.
- `transformers`: Hugging Face's library for NLP models (e.g., BERT, GPT).
- `opencv-python-headless`: OpenCV for computer vision tasks (without GUI).
- `moviepy`: For video editing.
- `pillow`: Image manipulation library.
- `reportlab`: For generating PDFs.
- `yt-dlp`: For downloading videos from YouTube.
- `google-api-python-client`: To interact with Google services.
- `nbformat`, `nbconvert`: For Jupyter notebook reading and conversion.
- `psutil`: For system utilities like memory management.

---

2. **Importing Libraries**

```
import yt_dlp

import torch

import librosa

from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC, pipeline

import os

import psutil
```

This imports necessary libraries:

- `yt_dlp`: For downloading audio or video from YouTube.
- `torch`: PyTorch library for deep learning.
- `librosa`: For audio processing.

- `Wav2Vec2Processor`, `Wav2Vec2ForCTC`, `pipeline`: From `transformers` for speech-to-text transcription using Wav2Vec 2.0.
- `os`: For interacting with the operating system.
- `psutil`: For accessing system/process information.

---

3. **Downloading Audio from YouTube**

```
def download_audio_from_youtube(link, output_path):

    ydl_opts = {

        'format': 'bestaudio/best',

        'outtmpl': output_path,

        'quiet': True,

        'postprocessors': [{

            'key': 'FFmpegExtractAudio',

            'preferredcodec': 'wav',

            'preferredquality': '192',

        }],

    }

    with yt_dlp.YoutubeDL(ydl_opts) as ydl:

        ydl.download([link])

    print(f"Audio downloaded and saved at {output_path}")
```

- This function downloads audio from a YouTube link, extracts it as a WAV file, and saves it to the specified path.

---

4. **Printing Memory Usage**

```
def print_memory_usage():

    process = psutil.Process(os.getpid())

    print(f"Memory usage: {process.memory_info().rss / 1024 ** 2:.2f} MB")
```

- This function prints the current memory usage of the Python process in MB.

---

5. **Transcribing Audio with Hugging Face's Wav2Vec2**

```
def transcribe_audio_with_huggingface(audio_path, batch_size=30):

    print_memory_usage()

    print("Loading the model...")

    processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base-960h")

    model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base-960h")

    print_memory_usage()


    print("Processing audio...")

    audio, rate = librosa.load(audio_path, sr=16000)

    total_length = len(audio)

    transcription = ""


    for i in range(0, total_length, batch_size * rate):

        batch_audio = audio[i:i + batch_size * rate]

        if len(batch_audio) == 0:
            break

        input_values = processor(batch_audio, sampling_rate=rate,
return_tensors="pt").input_values

        print_memory_usage()


        print("Transcribing audio batch...")

        logits = model(input_values).logits

        predicted_ids = torch.argmax(logits, dim=-1)

        if predicted_ids.max() >= model.config.vocab_size:

            print(f"Warning: Predicted ID {predicted_ids.max()} is out of range.")

            continue

        batch_transcription = processor.decode(predicted_ids[0])

        transcription += batch_transcription + " "


        # Clear intermediate data

        del batch_audio, input_values, logits, predicted_ids

        torch.cuda.empty_cache()

        print_memory_usage()


    return transcription.strip()
```

- This function transcribes audio to text using Hugging Face's Wav2Vec2 model.
- It processes the audio in batches and clears memory after each batch to prevent overflow.

---

6. **Generating a Text File from the Transcription**

```
def generate_text_file_from_transcription(transcription, output_text_path):
    print("Generating text file...")
    with open(output_text_path, 'w') as file:
        file.write("Audio Transcription Report\n\n")
        file.write(transcription)
    print(f"Text file saved at {output_text_path}")
```

- This function generates a text file with the transcription.

---

7. **Extracting Video Metadata from YouTube**

```
def extract_video_metadata(link):
    ydl_opts = {'quiet': True}
    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        info = ydl.extract_info(link, download=False)
    title = info.get("title", "No title available")
    description = info.get("description", "No description available")
    return title, description
```

- This function extracts metadata (title and description) from a YouTube video.

---

8. **Reading a Transcription File**

```
def read_transcription_file(transcription_path):
    with open(transcription_path, 'r') as file:
        transcription = file.read()
    return transcription
```

- This function reads a transcription file and returns its contents.

---

9. **Summarizing Text with Hugging Face's BART**

```
def summarize_text_chunk(chunk, title, description):
    summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
    text = title + " " + description + " " + chunk
    summary = summarizer(text, max_length=150, min_length=50, do_sample=False)
    return summary[0]['summary_text']
```

- This function uses Hugging Face's BART model to summarize text.

---

10. **Generating a Summary Text File**

```
def generate_text_file_from_summary(summary, output_text_path):
    print("Generating summary text file...")
    with open(output_text_path, 'w') as file:
        file.write("Summary Report\n\n")
        points = summary.split('. ')
        for point in points:
            file.write(f"- {point.strip()}.\n")
    print(f"Summary text file saved at {output_text_path}")
```

- This function generates a summary text file.

---

11. **Main Function for Processing Video**

```
def process_video(link, audio_path, text_path, summary_text_path):

    download_audio_from_youtube(link, audio_path)

    transcription = transcribe_audio_with_huggingface(audio_path)

    generate_text_file_from_transcription(transcription, text_path)

    title, description = extract_video_metadata(link)

    transcription = read_transcription_file(text_path)

    chunks = [transcription[i:i + 1000] for i in range(0, len(transcription), 1000)]

    final_summary = ""

    for chunk in chunks:

        summary = summarize_text_chunk(chunk, title, description)

        final_summary += summary + " "

    generate_text_file_from_summary(final_summary.strip(), summary_text_path)
```

- This function:
    - Downloads audio from YouTube.
    - Transcribes the audio.
    - Generates a text file with the transcription.
    - Extracts video metadata.
    - Summarizes the transcription.
    - Generates a summary text file.

---

12. **Running the Process**

```
‘ ‘ ‘video_link = "https://www.youtube.com/watch?v=0oGJTQCy4cQ"

audio_file_path = "./processed/audio_extracted.wav"

text_file_path = "./processed/audio_transcription.txt"
```

```
summary_text_path = "./processed/summary_report.txt"

os.makedirs("./processed", exist_ok=True)

process_video(video_link, audio_file_path, text_file_path, summary_text_path)’ ‘ ‘
```

- This block defines the video link and file paths for audio, transcription, and summary.
- It runs the `process_video` function to process the video.