roman-ai

March 25, 2024

[ ]:
```
!pip install SpeechRecognition
!pip install pydub
!apt install ffmpeg
!pip install googletrans==4.0.0-rc1
!pip install gtts
!pip install nltk
!pip install moviepy


import speech_recognition as sr
from pydub import AudioSegment
import nltk
from nltk.tokenize import sent_tokenize
from googletrans import Translator
from gtts import gTTS
from moviepy.editor import VideoFileClip, AudioFileClip
nltk.download('punkt')

#extract the audio from the video
# Import the necessary module
from moviepy.editor import *

# Load the video
video = VideoFileClip("/content/Coding         _ how to learn coding, A2␣
 ↪Motivation, A2 Sir, #shorts.mp4")

# Extract audio
audio = video.audio

# Save the audio
audio.write_audiofile("output_audio.mp3")

#remove the audio from the video

# Import necessary libraries

# Use moviepy to remove audio from the video
```

```
clip = video
video_only = clip.without_audio()
output_video_path = "no_audio.mp4" # replace with desired output path
video_only.write_videofile(output_video_path)

#export the mp3 format to wav format
from pydub import AudioSegment

audio = AudioSegment.from_mp3("output_audio.mp3")
audio.export("output_audio.wav4", format="wav")



def generate_audio_from_english_to_tamil():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='ta').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="ta")
    tts.save("new_audio.mp3")

    #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")



    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_english_to_hindi():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]
```

```python
    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='hi').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="hi")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")
    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_hindi_to_tamil():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='hi-IN')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)
    translator = Translator()
    translated_text = translator.translate(final_text, src='hi', dest='ta').text
    print(translated_text)

    tts= gTTS(translated_text,lang="ta")
    tts.save("new_audio.mp3")



     #sync the audio into the video
    from moviepy.editor import VideoFileClip, AudioFileClip

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")



    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_hindi_to_english():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
```

```
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
             audio_data = recognizer.record(source)
             try:
                 text = recognizer.recognize_google(audio_data, language='hi-IN')

                 # Use Sentence Boundary Detection
                 sentences = sent_tokenize(text)

                 # Combine sentences with commas
                 return ", ".join(sentences)
             except:
                 return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='hi', dest='en').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="en")
    tts.save("new_audio.mp3")



     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")
    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_tamil_to_english():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
             audio_data = recognizer.record(source)
             try:
                 text = recognizer.recognize_google(audio_data, language='ta-IN')

                 # Use Sentence Boundary Detection
                 sentences = sent_tokenize(text)

                 # Combine sentences with commas
                 return ", ".join(sentences)
             except:
                 return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)
    translator = Translator()
    translated_text = translator.translate(final_text, src='ta', dest='en').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="en")
    tts.save("new_audio.mp3")



     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")



    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_tamil_to_hindi():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
```

```
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
             audio_data = recognizer.record(source)
             try:
                 text = recognizer.recognize_google(audio_data, language='ta-IN')

                 # Use Sentence Boundary Detection
                 sentences = sent_tokenize(text)

                 # Combine sentences with commas
                 return ", ".join(sentences)
             except:
                 return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='ta', dest='hi').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="en")
    tts.save("new_audio.mp3")



     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")
    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)



def generate_audio_from_english_to_malayalam():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='ml').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="ml")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_english_to_kannadam():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
```

```
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='kn').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="kn")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)



def generate_audio_from_english_to_telungu():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='te').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="te")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_english_to_bengali():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='bn').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="bn")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)
```

```python
def generate_audio_from_english_to_punjabi():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    print(final_text)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='pa').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="pa")
    tts.save("new_audio.mp3")
```

```
     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_english_to_gujarati():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)
                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='gu').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="gu")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_english_to_urdu():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
       # Save the chunk to a temporary file
       chunk_filename = "temp_chunk.wav"
       chunk.export(chunk_filename, format="wav")

       # Convert audio chunk to text
       with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='ur').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="ur")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)



def generate_audio_from_english_to_sanskrit():


    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, language='en-US')

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)


    translator = Translator()
    translated_text = translator.translate(final_text, src='en', dest='sa').text
    print(translated_text)

    tts= gTTS(text=translated_text,lang="sa")
    tts.save("new_audio.mp3")

     #sync the audio into the video

    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")


    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)




def generate_audio_from_hindi_to_urdu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
```

```python
def process_audio_chunk(chunk):
    # Save the chunk to a temporary file
    chunk_filename = "temp_chunk.wav"
    chunk.export(chunk_filename, format="wav")

    # Convert audio chunk to text
    with sr.AudioFile(chunk_filename) as source:
        audio_data = recognizer.record(source)
        try:
            text = recognizer.recognize_google(audio_data,␣
↪language='hi-IN')  # Adjust language parameter if needed

            # Use Sentence Boundary Detection
            sentences = sent_tokenize(text)

            # Combine sentences with commas
            return ", ".join(sentences)
        except:
            return ""

# Process each chunk
texts = [process_audio_chunk(chunk) for chunk in chunks]

# Combine all texts
final_text = ", ".join(texts)

# Translate text from Hindi to Urdu
translator = Translator()
translated_text = translator.translate(final_text, src='hi', dest='ur').text
print(translated_text)

# Convert translated text to Urdu audio
tts = gTTS(text=translated_text, lang="ur")
tts.save("new_audio.mp3")

# Sync the audio into the video
def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
    # Load audio and video clips
    audio_clip = AudioFileClip(audio_path)
    empty_video_clip = VideoFileClip(empty_video_path)

    # Set the audio of the empty video to the extracted audio
    synced_video_clip = empty_video_clip.set_audio(audio_clip)

    # Write the synchronized video to the output path
    synced_video_clip.write_videofile(output_video_path, codec="libx264")
    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_hindi_to_malayalam():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='hi-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Hindi to Malayalam
    translator = Translator()
    translated_text = translator.translate(final_text, src='hi', dest='ml').text
    print(translated_text)
    # Convert translated text to Malayalam audio
    tts = gTTS(text=translated_text, lang="ml")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_hindi_to_kannada():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='hi-IN')  # Adjust language parameter if needed
```

```
                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Hindi to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='hi', dest='kn').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_hindi_to_telugu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='hi-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Hindi to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='hi', dest='te').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_tamil_to_urdu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ta-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Hindi to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='ta', dest='ur').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_tamil_to_malayalam():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ta-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Tamil to Malayalam
    translator = Translator()
    translated_text = translator.translate(final_text, src='ta', dest='ml').text
    print(translated_text)

    # Convert translated text to Malayalam audio
    tts = gTTS(text=translated_text, lang="ml")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)
def generate_audio_from_tamil_to_kannada():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ta-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Tamil to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='ta', dest='kn').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_tamil_to_telugu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='ta-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""
    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Tamil to Telugu
    translator = Translator()
    translated_text = translator.translate(final_text, src='ta', dest='te').text
    print(translated_text)

    # Convert translated text to Telugu audio
    tts = gTTS(text=translated_text, lang="te")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_malayalam_to_tamil():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ml-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Malayalam to Tamil
    translator = Translator()
    translated_text = translator.translate(final_text, src='ml', dest='ta').text
    print(translated_text)

    # Convert translated text to Tamil audio
    tts = gTTS(text=translated_text, lang="ta")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"
    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_malayalam_to_english():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='ml-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Malayalam to English
    translator = Translator()
    translated_text = translator.translate(final_text, src='ml', dest='en').text
    print(translated_text)

    # Convert translated text to English audio
    tts = gTTS(text=translated_text, lang="en")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_malayalam_to_hindi():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='ml-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Malayalam to Hindi
    translator = Translator()
    translated_text = translator.translate(final_text, src='ml', dest='hi').text
    print(translated_text)

    # Convert translated text to Hindi audio
    tts = gTTS(text=translated_text, lang="hi")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_malayalam_to_kannada():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]
```

```
    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ml-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Malayalam to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='ml', dest='kn').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")
    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_malayalam_to_telugu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='ml-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Malayalam to Telugu
    translator = Translator()
    translated_text = translator.translate(final_text, src='ml', dest='te').text
    print(translated_text)

    # Convert translated text to Telugu audio
    tts = gTTS(text=translated_text, lang="te")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_malayalam_to_urdu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ml-IN')  # Adjust language parameter if needed
```

```
                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Malayalam to Urdu
    translator = Translator()
    translated_text = translator.translate(final_text, src='ml', dest='ur').text
    print(translated_text)

    # Convert translated text to Urdu audio
    tts = gTTS(text=translated_text, lang="ur")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_telugu_to_tamil():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='te-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Telugu to Tamil
    translator = Translator()
    translated_text = translator.translate(final_text, src='te', dest='ta').text
    print(translated_text)

    # Convert translated text to Tamil audio
    tts = gTTS(text=translated_text, lang="ta")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_telugu_to_english():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='te-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    # Translate text from Telugu to English
    translator = Translator()
    translated_text = translator.translate(final_text, src='te', dest='en').text
    print(translated_text)

    # Convert translated text to English audio
    tts = gTTS(text=translated_text, lang="en")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_telugu_to_hindi():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='te-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Telugu to Hindi
    translator = Translator()
    translated_text = translator.translate(final_text, src='te', dest='hi').text
    print(translated_text)

    # Convert translated text to Hindi audio
    tts = gTTS(text=translated_text, lang="hi")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_telugu_to_malayalam():
    # Initialize the recognizer
```

```python
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='te-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Telugu to Malayalam
    translator = Translator()
    translated_text = translator.translate(final_text, src='te', dest='ml').text
    print(translated_text)

    # Convert translated text to Malayalam audio
    tts = gTTS(text=translated_text, lang="ml")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_telugu_to_kannada():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio), 
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data, 
 ↪language='te-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Telugu to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='te', dest='kn').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_telugu_to_urdu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='te-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Telugu to Urdu
    translator = Translator()
    translated_text = translator.translate(final_text, src='te', dest='ur').text
    print(translated_text)

    # Convert translated text to Urdu audio
    tts = gTTS(text=translated_text, lang="ur")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"
    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_kannada_to_tamil():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='kn-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Kannada to Tamil
    translator = Translator()
    translated_text = translator.translate(final_text, src='kn', dest='ta').text
    print(translated_text)

    # Convert translated text to Tamil audio
    tts = gTTS(text=translated_text, lang="ta")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_kannada_to_english():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='kn-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Kannada to English (no translation needed)

    # Convert translated text to English audio
    tts = gTTS(text=final_text, lang="en")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_kannada_to_hindi():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='kn-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Kannada to Hindi
    translator = Translator()
    translated_text = translator.translate(final_text, src='kn', dest='hi').text
    print(translated_text)

    # Convert translated text to Hindi audio
    tts = gTTS(text=translated_text, lang="hi")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_kannada_to_malayalam():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='kn-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Kannada to Malayalam
    translator = Translator()
    translated_text = translator.translate(final_text, src='kn', dest='ml').text
    print(translated_text)
    # Convert translated text to Malayalam audio
    tts = gTTS(text=translated_text, lang="ml")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_kannada_to_urdu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='kn-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Kannada to Urdu
    translator = Translator()
    translated_text = translator.translate(final_text, src='kn', dest='ur').text
    print(translated_text)

    # Convert translated text to Urdu audio
    tts = gTTS(text=translated_text, lang="ur")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_kannada_to_telugu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='kn-IN')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Kannada to Urdu
    translator = Translator()
    translated_text = translator.translate(final_text, src='kn', dest='te').text
    print(translated_text)

    # Convert translated text to Urdu audio
    tts = gTTS(text=translated_text, lang="ur")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)
        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_urdu_to_tamil():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='ur-PK')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)
    # Translate text from Urdu to Tamil
    translator = Translator()
    translated_text = translator.translate(final_text, src='ur', dest='ta').text
    print(translated_text)

    # Convert translated text to Tamil audio
    tts = gTTS(text=translated_text, lang="ta")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_urdu_to_english():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ur-PK')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Urdu to English (no translation needed)

    # Convert translated text to English audio
    tts = gTTS(text=final_text, lang="en")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)

def generate_audio_from_urdu_to_hindi():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ur-PK')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Urdu to Hindi
    translator = Translator()
    translated_text = translator.translate(final_text, src='ur', dest='hi').text
    print(translated_text)

    # Convert translated text to Hindi audio
    tts = gTTS(text=translated_text, lang="hi")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_urdu_to_malayalam():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
 ↪language='ur-PK')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Urdu to Malayalam
    translator = Translator()
    translated_text = translator.translate(final_text, src='ur', dest='ml').text
    print(translated_text)

    # Convert translated text to Malayalam audio
    tts = gTTS(text=translated_text, lang="ml")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


def generate_audio_from_urdu_to_telugu():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
 ↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ur-PK')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Urdu to Telugu
    translator = Translator()
    translated_text = translator.translate(final_text, src='ur', dest='te').text
    print(translated_text)

    # Convert translated text to Telugu audio
    tts = gTTS(text=translated_text, lang="te")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)
```

```
def generate_audio_from_urdu_to_kannada():
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load the audio and split it into 30-second chunks
    audio = AudioSegment.from_wav("output_audio.wav4")
    chunk_length = 30 * 1000  # 30 seconds in milliseconds
    chunks = [audio[i:i + chunk_length] for i in range(0, len(audio),␣
↪chunk_length)]

    # Define function to process audio chunk
    def process_audio_chunk(chunk):
        # Save the chunk to a temporary file
        chunk_filename = "temp_chunk.wav"
        chunk.export(chunk_filename, format="wav")

        # Convert audio chunk to text
        with sr.AudioFile(chunk_filename) as source:
            audio_data = recognizer.record(source)
            try:
                text = recognizer.recognize_google(audio_data,␣
↪language='ur-PK')  # Adjust language parameter if needed

                # Use Sentence Boundary Detection
                sentences = sent_tokenize(text)

                # Combine sentences with commas
                return ", ".join(sentences)
            except:
                return ""

    # Process each chunk
    texts = [process_audio_chunk(chunk) for chunk in chunks]

    # Combine all texts
    final_text = ", ".join(texts)

    # Translate text from Urdu to Kannada
    translator = Translator()
    translated_text = translator.translate(final_text, src='ur', dest='kn').text
    print(translated_text)

    # Convert translated text to Kannada audio
    tts = gTTS(text=translated_text, lang="kn")
    tts.save("new_audio.mp3")

    # Sync the audio into the video
    def sync_audio_to_video(audio_path, empty_video_path, output_video_path):
        # Load audio and video clips
        audio_clip = AudioFileClip(audio_path)
        empty_video_clip = VideoFileClip(empty_video_path)

        # Set the audio of the empty video to the extracted audio
        synced_video_clip = empty_video_clip.set_audio(audio_clip)

        # Write the synchronized video to the output path
        synced_video_clip.write_videofile(output_video_path, codec="libx264")

    audio_path = "new_audio.mp3"
    empty_video_path = "no_audio.mp4"
    output_video_path = "synced_video.mp4"

    sync_audio_to_video(audio_path, empty_video_path, output_video_path)


original_language = input("original_language:")
translated_language = input("translated_language")
if original_language == 'tamil':
    if translated_language == 'english':
        generate_audio_from_tamil_to_english()
    elif translated_language == 'hindi':
        generate_audio_from_tamil_to_hindi()
    elif translated_language == 'malayalam':
        generate_audio_from_tamil_to_malayalam()
    elif translated_language == 'urdu':
        generate_audio_from_tamil_to_urdu()
    elif translated_language == 'kannadam':
        generate_audio_from_tamil_to_kannada()
    elif translated_language == 'telugu':
        generate_audio_from_tamil_to_telugu()
    else:
        print("Translation to", translated_language, "is not supported.")



if original_language == 'english':
    if translated_language == 'tamil':
        generate_audio_from_english_to_tamil()
    elif translated_language == 'hindi':
        generate_audio_from_english_to_hindi()
    elif translated_language == 'malayalam':
        generate_audio_from_english_to_malayalam()
    elif translated_language == 'kannadam':
        generate_audio_from_english_to_kannadam()
    elif translated_language == 'telugu':
        generate_audio_from_english_to_telungu()
    elif translated_language == 'gujarati':
        generate_audio_from_english_to_gujarati()
    elif translated_language == 'punjabi':
        generate_audio_from_english_to_punjabi()
    elif translated_language == 'sanskrit':
        generate_audio_from_english_to_sanskrit()
    elif translated_language == 'urdu':
        generate_audio_from_english_to_urdu()
    elif translated_language == 'bengali':
        generate_audio_from_english_to_bengali()
    else:
        print("Translation to", translated_language, "is not supported.")


if original_language == 'hindi':
    if translated_language == 'tamil':
        generate_audio_from_hindi_to_tamil()
    elif translated_language == 'english':
        generate_audio_from_hindi_to_english()
    elif translated_language == 'kannadam':
        generate_audio_from_hindi_to_kannada()
    elif translated_language == 'malayalam':
        generate_audio_from_hindi_to_malayalam()
    elif translated_language == 'urdu':
        generate_audio_from_hindi_to_urdu()
    elif translated_language == 'telugu':
        generate_audio_from_hindi_to_telugu()
    else:
        print("Translation to", translated_language, "is not supported.")



if original_language == 'malayalam':
        if translated_language == "english":
            generate_audio_from_malayalam_to_english()
        elif translated_language == 'hindi':
            generate_audio_from_malayalam_to_hindi()
        elif translated_language == 'tamil':
            generate_audio_from_malayalam_to_tamil()
        elif translated_language == 'kannadam':
            generate_audio_from_malayalam_to_kannada()
        elif translated_language == 'telugu':
            generate_audio_from_malayalam_to_telugu()
        elif translated_language == 'urdu':
            generate_audio_from_malayalam_to_urdu()
        else:
          print("Translation to", translated_language, "is not supported.")

if original_language == 'kannadam':
        if translated_language == "english":
            generate_audio_from_kannada_to_english()
        elif translated_language == 'hindi':
            generate_audio_from_kannada_to_hindi()
        elif translated_language == 'tamil':
            generate_audio_from_kannada_to_tamil()
        elif translated_language == 'malayalam':
            generate_audio_from_kannada_to_malayalam()
        elif translated_language == 'telugu':
            generate_audio_from_kannada_to_telugu()
        elif translated_language == 'urdu':
            generate_audio_from_kannada_to_urdu()
        else:
          print("Translation to", translated_language, "is not supported.")

if original_language == 'telugu':
        if translated_language == "english":
            generate_audio_from_telugu_to_english()
        elif translated_language == 'hindi':
            generate_audio_from_telugu_to_hindi()
        elif translated_language == 'tamil':
            generate_audio_from_telugu_to_tamil()
        elif translated_language == 'malayalam':
            generate_audio_from_telugu_to_malayalam()
        elif translated_language == 'kannadam':
            generate_audio_from_telugu_to_kannada()
        elif translated_language == 'urdu':
            generate_audio_from_telugu_to_urdu()
        else:
         print("Translation to", translated_language, "is not supported.")

if original_language == 'urdu':
        if translated_language == "english":
            generate_audio_from_urdu_to_english()
        elif translated_language == 'hindi':
            generate_audio_from_urdu_to_hindi()
        elif translated_language == 'tamil':
            generate_audio_from_urdu_to_tamil()
        elif translated_language == 'malayalam':
            generate_audio_from_urdu_to_malayalam()
        elif translated_language == 'kannadam':
            generate_audio_from_urdu_to_kannada()
        elif translated_language == 'telugu':
            generate_audio_from_urdu_to_telugu()
        else:
          print("Translation to", translated_language, "is not supported.")
```

```
Requirement already satisfied: SpeechRecognition in
/usr/local/lib/python3.10/dist-packages (3.10.1)
Requirement already satisfied: requests>=2.26.0 in
/usr/local/lib/python3.10/dist-packages (from SpeechRecognition) (2.31.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from SpeechRecognition) (4.9.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests>=2.26.0->SpeechRecognition) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.26.0->SpeechRecognition) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests>=2.26.0->SpeechRecognition) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests>=2.26.0->SpeechRecognition) (2024.2.2)
Requirement already satisfied: pydub in /usr/local/lib/python3.10/dist-packages
(0.25.1)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
ffmpeg is already the newest version (7:4.4.2-0ubuntu0.22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Requirement already satisfied: googletrans==4.0.0-rc1 in
/usr/local/lib/python3.10/dist-packages (4.0.0rc1)
Requirement already satisfied: httpx==0.13.3 in /usr/local/lib/python3.10/dist-
packages (from googletrans==4.0.0-rc1) (0.13.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2024.2.2)
Requirement already satisfied: hstspreload in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2024.2.1)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (1.3.0)
Requirement already satisfied: chardet==3.* in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (3.0.4)
Requirement already satisfied: idna==2.* in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2.10)
Requirement already satisfied: rfc3986<2,>=1.3 in
/usr/local/lib/python3.10/dist-packages (from
httpx==0.13.3->googletrans==4.0.0-rc1) (1.5.0)
Requirement already satisfied: httpcore==0.9.* in
/usr/local/lib/python3.10/dist-packages (from
httpx==0.13.3->googletrans==4.0.0-rc1) (0.9.1)
Requirement already satisfied: h11<0.10,>=0.8 in /usr/local/lib/python3.10/dist-
packages (from httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1) (0.9.0)
Requirement already satisfied: h2==3.* in /usr/local/lib/python3.10/dist-
packages (from httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1) (3.2.0)
Requirement already satisfied: hyperframe<6,>=5.2.0 in
/usr/local/lib/python3.10/dist-packages (from
h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1) (5.2.0)
Requirement already satisfied: hpack<4,>=3.0 in /usr/local/lib/python3.10/dist-
packages (from h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1)
(3.0.0)
Requirement already satisfied: gtts in /usr/local/lib/python3.10/dist-packages
(2.5.1)
Requirement already satisfied: requests<3,>=2.27 in
/usr/local/lib/python3.10/dist-packages (from gtts) (2.31.0)
Requirement already satisfied: click<8.2,>=7.1 in
/usr/local/lib/python3.10/dist-packages (from gtts) (8.1.7)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gtts) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.27->gtts) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gtts) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gtts)
(2024.2.2)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.2)
Requirement already satisfied: moviepy in /usr/local/lib/python3.10/dist-
packages (1.0.3)
Requirement already satisfied: decorator<5.0,>=4.0.2 in
/usr/local/lib/python3.10/dist-packages (from moviepy) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in
/usr/local/lib/python3.10/dist-packages (from moviepy) (4.66.2)
Requirement already satisfied: requests<3.0,>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from moviepy) (2.31.0)
Requirement already satisfied: proglog<=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from moviepy) (0.1.10)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-
packages (from moviepy) (1.25.2)
```

```
Requirement already satisfied: imageio<3.0,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from moviepy) (2.31.6)
Requirement already satisfied: imageio-ffmpeg>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from moviepy) (0.4.9)
Requirement already satisfied: pillow<10.1.0,>=8.3.2 in
/usr/local/lib/python3.10/dist-packages (from imageio<3.0,>=2.5->moviepy)
(9.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from imageio-ffmpeg>=0.2.0->moviepy) (67.7.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3.0,>=2.8.1->moviepy) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy)
(2024.2.2)

[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!

MoviePy - Writing audio in output_audio.mp3



MoviePy - Done.
Moviepy - Building video no_audio.mp4.
Moviepy - Writing video no_audio.mp4


t: 100%|      | 1426/1433 [00:11<00:00, 101.86it/s,
now=None]WARNING:py.warnings:/usr/local/lib/python3.10/dist-
packages/moviepy/video/io/ffmpeg_reader.py:123: UserWarning: Warning: in file
/content/Coding         _ how to learn coding, A2 Motivation, A2 Sir,
#shorts.mp4, 388800 bytes wanted but 0 bytes read,at frame 1432/1433, at time
47.73/47.74 sec. Using the last valid frame instead.
  warnings.warn("Warning: in file %s, "%(self.filename)+


Moviepy - Done !
Moviepy - video ready no_audio.mp4
original_language:hindi
translated_languagemalayalam
              ,             ,
             ,
                          ,
        ,              .
```

```
Moviepy - Building video synced_video.mp4.
MoviePy - Writing audio in synced_videoTEMP_MPY_wvf_snd.mp3


MoviePy - Done.
Moviepy - Writing video synced_video.mp4


t: 100%|      | 1432/1434 [00:09<00:00, 170.67it/s,
now=None]WARNING:py.warnings:/usr/local/lib/python3.10/dist-
packages/moviepy/video/io/ffmpeg_reader.py:123: UserWarning: Warning: in file
no_audio.mp4, 388800 bytes wanted but 0 bytes read,at frame 1433/1434, at time
47.77/47.77 sec. Using the last valid frame instead.
  warnings.warn("Warning: in file %s, "%(self.filename)+


Moviepy - Done !
Moviepy - video ready synced_video.mp4
```

```
[ ]:
```