import streamlit as st

from pytube import YouTube

from moviepy.editor import \*

import speech\_recognition as sr

import requests

import csv

import os

import io

import nltk

from pydub import AudioSegment

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

def download\_audio(youtube\_link):

try:

# Extract the YouTube video's audio

video = YouTube(youtube\_link)

audio = video.streams.filter(only\_audio=True).first()

audio\_streams = video.streams.filter(only\_audio=True)

audio\_streams = sorted(audio\_streams, key=lambda s: s.bitrate)

audio = audio\_streams[0]

# Download the audio

st.write("Downloading audio...")

audio\_filename = "my\_audio.mp3"

audio.download(filename=audio\_filename)

st.success("Audio downloaded successfully!")

return audio\_filename

except Exception as e:

st.error(f"An error occurred during audio download: {str(e)}")

def process\_audio(audio\_filename):

try:

# Convert audio to text

st.write("Converting audio to text...")

audio\_path = f"./{audio\_filename}.wav"

audio\_clip = AudioFileClip(audio\_filename)

audio\_clip.write\_audiofile(audio\_path)

# Perform speech recognition

r = sr.Recognizer()

with sr.AudioFile(audio\_path) as source:

audio\_data = r.record(source)

#text = r.recognize\_google(audio\_data)

text = r.recognize\_sphinx(audio\_data)

st.success("Audio converted to text successfully!")

st.write("Text:")

#st.write(text1)

st.write(text)

# Save the text as a text file

text\_file\_path = "./audio.txt"

with open(text\_file\_path, "w") as text\_file:

text\_file.write(text)

st.success("Text saved successfully as a text file!")

# Perform forced alignment with Gentle

st.write("Performing forced alignment...")

# Prepare the data for the Gentle API request

gentle\_url = "http://localhost:8888/transcriptions?async=false"

files = {"audio": open(audio\_path, "rb")}

data = {"transcript": text}

# Send the request to the Gentle server

response = requests.post(gentle\_url, files=files, data=data)

if response.status\_code == 200:

alignment\_data = response.json()

# Save the alignment result as CSV

csv\_path = "./alignment\_result.csv"

with open(csv\_path, "w", newline="") as csv\_file:

writer = csv.writer(csv\_file)

writer.writerow(["word", "start", "end"])

for word in alignment\_data["words"]:

if "start" in word:

start\_time = word["start"]

elif "startOffset" in word:

start\_time = word["startOffset"] / 1000

else:

start\_time = ""

if "end" in word:

end\_time = word["end"]

elif "endOffset" in word:

end\_time = word["endOffset"] / 1000

else:

end\_time = ""

writer.writerow([word["word"], start\_time, end\_time])

st.write("Forced alignment result saved as CSV.")

else:

st.error("Failed to perform forced alignment.")

except Exception as e:

st.error(f"An error occurred during audio processing: {str(e)}")

stop\_words = set(stopwords.words('english'))

#nltk.download('punkt')

#nltk.download('stopwords')

def search\_word(csv\_file\_path, search\_input):

search\_results = []

temp\_file = 'temp.csv'

with open(csv\_file\_path, 'r') as file\_in, open(temp\_file, 'w', newline='') as file\_out:

reader = csv.reader(file\_in)

writer = csv.writer(file\_out)

for row in reader:

words = word\_tokenize(row[0]) # Tokenize the words in the first column

# Check if any word in the row is a connector word

if any(word in stop\_words for word in words):

continue # Skip the row if it contains a connector word

# Write the row to the temporary CSV file

writer.writerow(row)

# Replace the original CSV file with the modified output

os.replace(temp\_file, csv\_file\_path)

search\_words = search\_input.split()

with open(csv\_file\_path, "r") as csv\_file:

reader =list(csv.reader(csv\_file))

#next(reader) # Skip the header row

#for i in length(search\_words):

for i, row in enumerate(reader):

if row[0] == search\_words[0]:

if len(search\_words) == 1:

search\_results.append(row)

elif i + 1 < len(reader): # Ensure there is a next row

next\_row = reader[i + 1]

if len(search\_words) > 1 and next\_row[0] == search\_words[1]:

search\_results.append(row)

return search\_results

def delete\_files(file\_paths):

for file\_path in file\_paths:

try:

os.remove(file\_path)

except OSError as e:

st.error(f"Error deleting file '{file\_path}': {str(e)}")

def main():

st.title("VANS AUDIO SYSTEM")

#warm\_up\_speech\_recognition()

#sr.speech\_recognition.start()

r = sr.Recognizer()

# Select audio source option

audio\_source = st.radio("Select audio source", ("Audio Upload", "YouTube Link"))

if audio\_source == "Audio Upload":

# Audio upload option

uploaded\_file = st.file\_uploader("Upload Audio", type=["mp3", "wav"])

# Process uploaded audio if button is clicked

if st.button("Process Audio") and uploaded\_file is not None:

# Save the uploaded audio file with the desired filename

audio\_filename = "my\_audio.mp3" # Rename the file as desired

with open(audio\_filename, "wb") as f:

f.write(uploaded\_file.read())

st.success("Audio uploaded successfully!")

process\_audio(audio\_filename)

elif audio\_source == "YouTube Link":

# YouTube link option

youtube\_link = st.text\_input("Enter YouTube link")

# Process YouTube audio if button is clicked

if st.button("Process Audio"):

audio\_filename = download\_audio(youtube\_link)

# Check if audio download was successful before proceeding

if audio\_filename:

process\_audio(audio\_filename)

text\_file\_path = "./audio.txt"

# Open the file and read its contents

with open(text\_file\_path, "r") as file:

text = file.read()

st.write(text)

# Display the audio bar

audio\_file\_path = "./my\_audio.mp3"

audio\_bytes = open(audio\_file\_path, "rb").read()

st.audio(audio\_bytes, format="audio/mp3")

#generated\_text= process\_audio(audio\_filename)

#st.write(generated\_text)

# Input field for the search word

search\_input = st.text\_input("Enter word to search")

selected\_option = None

csv\_file\_path = "./alignment\_result.csv"

search\_results = search\_word(csv\_file\_path, search\_input)

search\_results\_messages = []

# Search button

if st.button("Search"):

if search\_results:

st.write("Search results:")

search\_results\_messages = []

for i, result in enumerate(search\_results):

#word = result[0]

word = search\_input

start\_time = float(result[1])

option = i + 1

#st.markdown(f"Option {option}: Word '{word}' found at t={start\_time}s")

message= f"Option {option}: Word '{word}' found at t={start\_time}s"

search\_results\_messages.append(message)

else:

st.write("No results found.")

for message in search\_results\_messages:

st.markdown(message)

if search\_results:

max\_value = len(search\_results)

else:

max\_value = 1

selected\_option = st.number\_input("Enter the option to seek", min\_value=1, max\_value=max\_value, step=1, value=1, key='seek\_option')

# Seek functionality

if st.button("Seek"):

if selected\_option:

selected\_result = search\_results[selected\_option - 1]

word = selected\_result[0]

start\_time = float(selected\_result[1])

modify\_audio(start\_time)

if st.button("Done"):

delete\_files(["my\_audio.mp3", "audio.txt", "alignment\_result.csv"])

st.write("All files deleted successfully!")

def modify\_audio(start\_time):

# Convert audio file to audio segment

audio = AudioSegment.from\_file("./my\_audio.mp3")

# Calculate total duration in seconds

total\_duration = len(audio) / 1000

# Seek to the desired time

new\_audio = audio[int(start\_time \* 1000):]

# Convert seeked audio segment to byte array

new\_audio\_bytes = new\_audio.export(format="wav").read()

# Display the audio player with seeked audio

st.audio(io.BytesIO(new\_audio\_bytes), format='audio/wav')

if \_\_name\_\_ == "\_\_main\_\_":

main()