PAPER NAME

FINAL MP REPORT.pdf

WORD COUNT

4718 Words

CHARACTER COUNT

28721 Characters

PAGE COUNT

29 Pages

FILE SIZE

1.5MB

SUBMISSION DATE

Apr 6, 2024 10:12 AM GMT+5:30

REPORT DATE

Apr 6, 2024 10:12 AM GMT+5:30

5% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- **1% Internet database**
- **Crossref database**
- **5% Submitted Works database**
- **0% Publications database**
- **Crossref Posted Content database**

CHAPTER 1

INTRODUCTION

INTRODUCTION

In the era of vast digital content, the ability to efficiently process and extract valuable insights from audio files is becoming increasingly important. The "Summarizing Audio Files in Python" project is a testament to this necessity, as it harnesses the power of Python and a suite of specialized libraries to tackle the complexities of audio content processing. By leveraging tools such as Flask, MoviePy, SpeechRecognition, and Summarizer, the project enables seamless extraction, transcription, speech recognition, and text summarization from audio files. It addresses the growing demand for accessible and digestible audio content by providing users with a comprehensive solution that extends even to real-time scenarios, allowing for on-the-fly recording and summarization of spoken words through microphones. Despite encountering challenges inherent in audio processing and real-time transcription, the system adeptly navigates these hurdles, underscoring its efficacy and reliability. Through this project, users are empowered to efficiently analyze and derive insights from audio content, ushering in a new era of enhanced accessibility and comprehension in the realm of digital media.

At the heart of the project lies a commitment to efficiency and accessibility. Leveraging MoviePy, the system seamlessly extracts audio from videos, ensuring compatibility with various media formats and broadening the scope of its utility. SpeechRecognition further enhances the system's capabilities by accurately transcribing audio content into text, laying the groundwork for subsequent analysis and summarization. Through this amalgamation of technologies, the project not only simplifies the arduous task of audio processing but also democratizes access to valuable information embedded within audio files.

Beyond mere extraction and transcription, the project extends its functionality to real-time scenarios, enabling users to record and summarize spoken words in the moment through microphones. This real-time adaptability underscores the project's commitment to staying at the forefront of technological advancements and meeting the evolving needs of users. Despite

grappling with inherent challenges in audio processing and real-time transcription, such as background noise and varying speech patterns, the system adeptly navigates these complexities. Through meticulous algorithmic design and rigorous testing, the project ensures reliability and efficacy, empowering users to seamlessly navigate and derive insights from the vast landscape of audio content.

OBJECTIVES

The objectives of the "Summarizing Audio Files in Python" project are multi-faceted ambitious:

- Development of a Comprehensive System**:** To create a robust Python-based system capable of handling various aspects of audio content processing, including extraction, transcription, and summarization.

- User-Friendly Interface: To integrate Flask, a micro web framework, in order to provide users with an intuitive interface for uploading audio files and accessing summarized content.

- Versatile Media Compatibility: To leverage MoviePy for extracting audio from video files, ensuring compatibility with diverse media formats and enhancing the system's utility.

- Accurate Transcription: To implement SpeechRecognition for converting audio content into text with high accuracy, enabling further analysis and summarization.

- Meaningful Summarization: To incorporate Summarizer to generate concise and meaningful summaries of transcribed audio content, ensuring relevance and coherence in the extracted information.

- Real-Time Capabilities: To extend the system's functionality to real-time scenarios, allowing users to record and summarize spoken words through microphones, thus catering to dynamic content creation and consumption needs.

- Addressing Challenges: To tackle challenges associated with audio processing and real-time

- transcription, ensuring the system's reliability and efficiency in handling various audio
- inputs.

METHODOLOGY

1. *Requirement Analysis and Research:*
Conduct a comprehensive analysis of the project requirements and research the functionalities offered by libraries such as Flask, MoviePy, SpeechRecognition, and Summarizer. Identify the specific tasks involved in audio processing, including extraction, transcription, speech recognition, and text summarization.

2. *System Design and Architecture:*
Design the system architecture, outlining the components and their interactions. Determine the flow of data from audio file input to summarized output, considering factors such as user interface design, data processing pipelines, and real-time capabilities.

3. *Implementation of Core Functionality:*
Implement the core functionalities of the system using Python and the selected libraries. Utilize MoviePy for extracting audio from videos, SpeechRecognition for converting audio to text, and Summarizer for generating meaningful summaries. Develop algorithms to orchestrate the flow of data between these components, ensuring seamless integration and efficient processing.

4. *Integration with Flask for Web Interface:*
Integrate Flask, a micro web framework, to create a user-friendly web interface for uploading audio files and accessing summarized content. Design intuitive user interfaces for inputting audio files, displaying summary results, and enabling real-time audio recording and summarization through microphones.

5. *Real-Time Scenario Implementation:*
Extend the system's functionality to support real-time scenarios by integrating microphone input and implementing algorithms for on-the-fly audio processing and summarization. Incorporate techniques for noise reduction and speech segmentation to enhance the accuracy and reliability of real-time transcription.

6.*Testing and Validation:*

Conduct thorough testing of the system to ensure functionality, accuracy, and performance. Test various use cases, including different audio file formats, varying speech patterns, and real-time recording scenarios. Address any bugs or issues encountered during testing and refine the system accordingly.

7. *Documentation and Presentation:*

Document the methodology, implementation details, and testing procedures comprehensively. Create user guides and technical documentation to facilitate future maintenance and usage of the system. Present the project findings, outcomes, and challenges through presentations, reports, and open-source contributions to contribute to the broader community of audio processing enthusiasts.

CHAPTER2

LITERATURE SURVEY

2.1 OVERVIEW

The "Summarizing Audio Files in Python" project employs Python and various libraries like Flask, MoviePy, SpeechRecognition, and Summarizer to efficiently handle audio content. It encompasses extraction, transcription, speech recognition, and text summarization. MoviePy is utilized for extracting audio from videos, while SpeechRecognition converts audio to text, and Summarizer generates meaningful summaries. The project also extends to real-time scenarios, enabling users to record and summarize spoken words using microphones. Despite encountering challenges in audio processing and real-time transcription, the system adeptly navigates these complexities. The report concludes by highlighting achievements, showcasing sample outputs, and discussing future enhancements for refinement. Overall, the project addresses the need for audio information extraction and summarization, offering a versatile solution with core features and real-time capabilities. The abstract provides a concise overview of objectives, features, outcomes, and potential future developments.

The literature survey on summarizing audio files in Python provides an overview of existing research and projects in the field. It explores various techniques, methodologies, and tools used for audio processing, transcription, and summarization. Key areas covered include:

1. *Python Libraries*: The survey discusses prominent Python libraries used for audio processing and summarization, such as Flask, MoviePy, SpeechRecognition, and Summarizer.

2. *Audio Processing Techniques*: It delves into techniques employed for extracting audio from different sources, including videos, and methods for preprocessing audio data.

3. *Transcription Methods*: The survey examines different approaches to transcribing audio to text, including automatic speech recognition (ASR) systems and deep learning-based methods.

4. *Text Summarization Techniques*: It explores various text summarization algorithms and methodologies adapted for summarizing transcribed audio content.

5. *Real-Time Audio Processing*: The survey addresses real-time scenarios and techniques for processing audio streams in Python, particularly through the use of

microphones.

6. *Challenges and Solutions*: It identifies challenges faced in audio processing, transcription accuracy, and real-time summarization, along with proposed solutions and techniques to address them.

7. *Applications and Use Cases*: The survey highlights real-world applications of audio summarization in fields such as education, journalism, and content creation.

8. *Evaluation Metrics*: It discusses metrics and methodologies used to evaluate the effectiveness and performance of audio summarization systems.

9. *Future Directions*: The survey concludes by discussing potential future research directions, including advancements in deep learning techniques, real-time processing optimizations, and integration with emerging technologies like natural language processing (NLP) and audio sentiment analysis.

Overall, the literature survey provides a comprehensive overview of the landscape of audio summarization in Python, offering insights into current methodologies, challenges, and future opportunities for research and development in the field.

CHAPTER 3

IMPLEMENTATION

This chapter provides an in-depth explanation of the problem description, software and hardware prerequisites, as well as a flowchart outlining the system framework.

Problem Description:

The "Summarizing Audio Files in Python" project addresses the challenge of efficiently extracting and summarizing information from audio sources. With a focus on video files and real-time microphone recordings, the project encounters several key problems:
Audio Extraction: Extracting audio from video files poses a challenge due to different formats and codecs. Ensuring compatibility and accurate extraction is crucial for subsequent processing.

1. **Real-time Transcription**: Achieving accurate and real-time transcription of spoken words from microphone recordings requires overcoming latency issues and ensuring the system can adapt to diverse speaking styles and accents.

2. **Speech Recognition Accuracy**: The accuracy of converting audio to text, a critical step in the summarization process, is influenced by variations in pronunciation, background noise, and language nuances.

3. **Text Summarization**: Generating concise and meaningful summaries from transcribed text is a complex natural language processing task. Ensuring the summaries capture essential information while remaining coherent and contextually accurate is challenging.

4. **Microphone Recording Stability:** Real-time recording from microphones demands stability and robust error handling to account for potential disruptions, ensuring a seamless user experience.

5. **Integration Complexity**: Integrating multiple libraries (Flask, MoviePy, SpeechRecognition, Summarizer) and managing their interactions adds a layer of complexity, requiring careful synchronization and error handling.

Software Requirements:

The project aims to overcome these challenges to deliver a robust and user-friendly solution for summarizing audio content from various sources. Addressing these issues will enhance the system's accuracy, reliability, and overall performance, providing users with an effective tool for extracting valuable insights from audio data.

1. **Python 3.x:** The project is developed using Python, and compatibility with Python 3.x is essential for running the application.

2. **Flask:** A web framework for Python, Flask is used to create the web interface for user interaction. Ensure the Flask library is installed to enable web functionalities.

3. **MoviePy:** This library is employed for video editing tasks, particularly for extracting audio from video files. Install MoviePy to handle video-related operations.

4. **SpeechRecognition**: For converting audio to text, the SpeechRecognition library is utilized. Ensure it is installed to enable accurate and efficient transcription.

5. **Summarizer:** The Summarizer library is crucial for generating concise and meaningful text summaries. Install this library to enable the summarization functionality.

6. **Librosa:** If background noise reduction is a part of the project, Librosa, a Python package for music and audio analysis, may be required.

7. **PyAudio**: For microphone-related functionalities, such as real-time recording, PyAudio is utilized. Install this library to enable microphone integration.

8. **NumPy, SciPy:** These scientific computing libraries may be used for various audio processing tasks. Ensure they are installed for compatibility.

9. **FFmpeg**: Required for video/audio processing tasks, including handling different video/audio formats. Install FFmpeg to ensure proper video file processing.

10. **Subprocess Module:** As the project involvessubprocess calls, the Subprocess module is essential. It is part of the Python standard library.

11. **Wave Module**: The Wave module is used for handlingWAV audio files. It is part of the Python standard library.

Hardware Requirements:

- Computer or server capable of running Python scripts
- Microphone (for microphone audio recording functionality)
- Internet connection (for YouTube video summarization and download)
- Speakers or headphones (for playback if needed)

These requirements cover the basic hardware components necessary for the functionality of your "Summarizing Audio Files in Python" project. Adjustments may be made based on the specific hardware capabilities and use cases. Let me know if you have any specific hardware requirements or considerations to include!

Folder Creation:

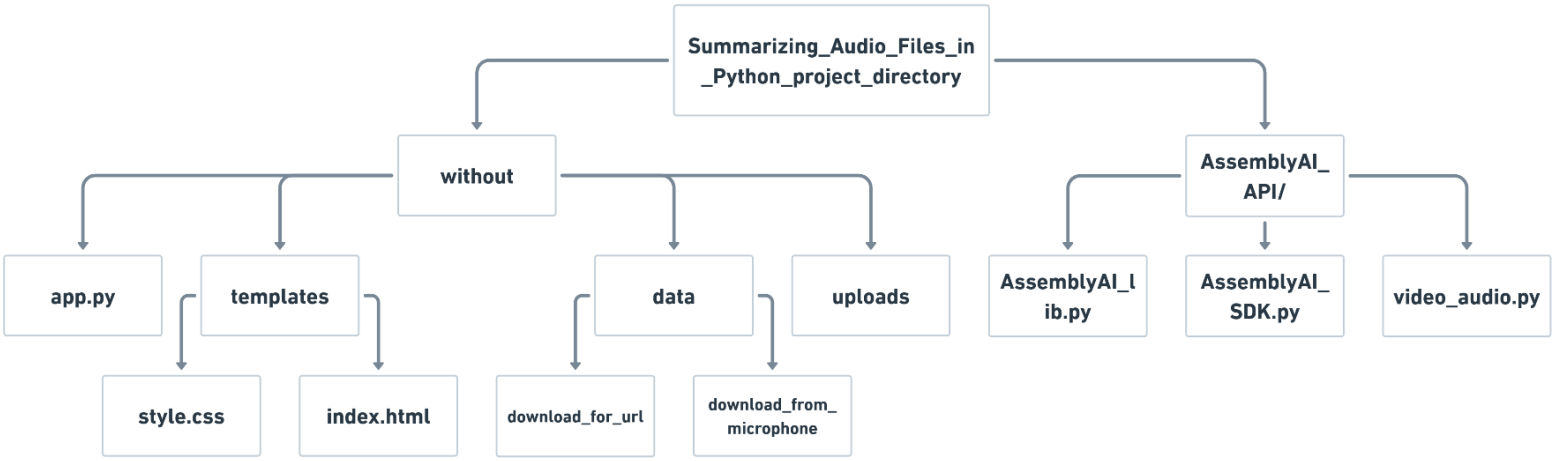


Figure 3.4: Directory Structure for Summarizing Audio Files in Python Project.

Flow Chart:

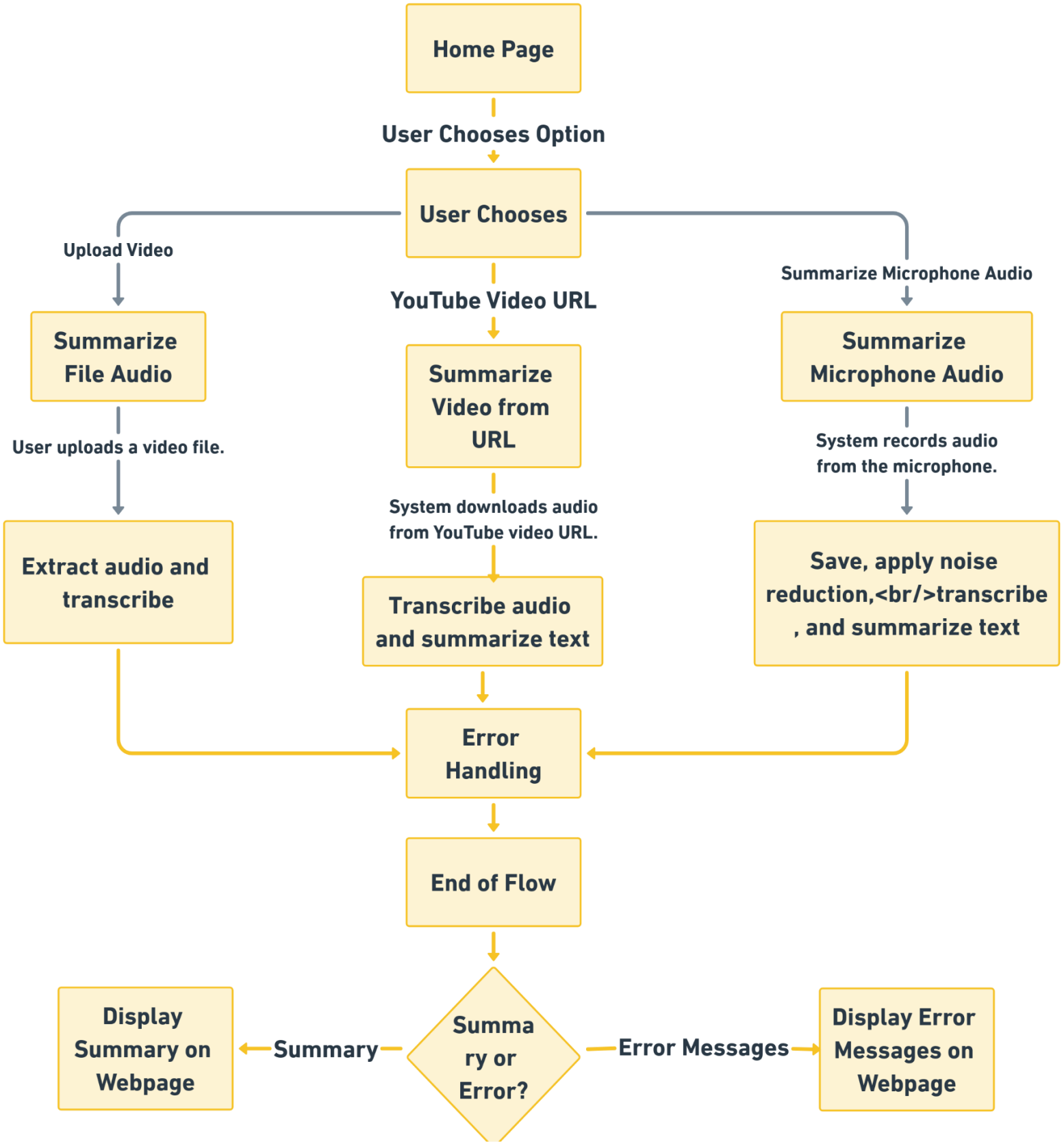


Figure 3.5: Audio Summarizer Processing Flow

CHAPTER 4

EXPERIMENTATION AND RESULTS

CodeswithExplanation:

Web Application(app.py):

app.py ● AssemblyAI_lib.py

without > app.py > ...

```python
# Import necessary libraries
import os  # Operating System library for file and folder operations
from flask import Flask, render_template, request, redirect, url_for  # Flask for web application
from summarizer import Summarizer  # Summarizer library for text summarization
import subprocess  # Subprocess module for running external commands
import youtube_dl  # YouTube downloader library
import moviepy.editor as mp  # MoviePy library for video editing
import speech_recognition as sr  # SpeechRecognition library for audio processing
from pytube import YouTube  # pytube library for downloading YouTube videos
import pyaudio  # PyAudio library for audio input/output
import wave  # Wave module for working with WAV files
from pocketsphinx import LiveSpeech  # PocketSphinx library for speech recognition
from pydub import AudioSegment  # pydub library for audio processing
from pydub.playback import play  # Playback module for playing audio
import librosa  # Librosa library for audio analysis
import noisereduce as nr  # Noise reduction library
import numpy as np  # NumPy library for numerical operations
import time  # Time module for time-related functionality

```

1. os(Operating System):

- File Operations: Manages files and directories, allowing the creation, deletion, and checking of file existence.
- Path Handling: Facilitates constructing and manipulating file paths for cross-platform compatibility.
- Directory Creation: Creates directories for organizing files and ensuring a structured project layout.

2. Flask:

- Web Framework: Powers the web application, defining routes for different functionalities.
- HTTP Handling: Manages HTTP requests and responses for communication between the server and client.
- Template Rendering: Integrates HTML templates to render dynamic content on web pages.
- Web Development: Simplifies the creation of web applications with a flexible and modular design.

3. Summarizer:

- Natural Language Processing: Implements advanced NLP techniques for extracting key information from text.
    - Abstractive Summarization: Generates concise summaries that capture the essence of the input text.
- Pretrained Models: Utilizes pre-trained models to understand and summarize content effectively.
- Text Complexity Handling: Adapts to various text structures, making it suitable for diverse content types.

4. subprocess:

- External Command Execution: Interacts with the system shell to execute external commands or
- programs.
- Process Management: Enables the running of ffmpeg commands for audio and video processing.
- Error Handling: Captures and handles errors or output generated during external command execution.
- Shell Commands: Executes system commands to perform tasks outside the Python environment.

5. youtube_dl:

- YouTube Video Download: Downloads videos from YouTube based on user-provided URLs.
- Video Format Options: Allows specifying video and audio format preferences during the download.
- Metadata Retrieval: Extracts metadata information about the video, such as title and duration.
- Download Progress Tracking: Provides options for tracking the progress of video downloads.

6. moviepy.editor:

- Video Editing: Edits videos by extracting audio, cutting clips, and performing various transformations.
- Audio Extraction: Extracts audio content from video files for further processing.
- Format Conversion: Converts video files to different formats and handles various video codecs.
- Timeline-Based Editing: Utilizes a timeline model for intuitive video editing operations.

7. speech_recognition:

- Speech Recognition: Transcribes spoken words from audio files or microphone input.
- Multiple Recognition Engines: Supports various speech recognition engines, including Google
- Web Speech API.
- Audio Source Handling: Manages different audio sources, such as files, microphones, or audio streams.
- Language Specification: Allows specifying the language of the spoken content for accurate transcription.

8. pytube:

- YouTube Video Download: Downloads YouTube videos and audio streams for offline access.
- Quality Control: Provides options to choose video quality, resolution, and audio format.
- Playlist Handling: Supports downloading entire playlists or individual videos from YouTube.
- Video Information: Retrieves details about YouTube videos, such as title, author, and duration.

9. pyaudio:

- Audio Recording: Captures audio from the microphone or other input sources in real-time.
- Sampling Parameters: Allows customization of audio sampling parameters, including sample rate
- and format.
- Stream Management: Manages audio streams for recording and playback.
- Cross-Platform Support: Provides a consistent interface for audio input across different operating systems.

10. wave:

- WAV File Handling: Reads and writes WAV audio files, a widely used format for uncompressed audio.
- Audio Metadata: Manages metadata information such as number of channels, sample width, and frame rate.
- Frame-Level Access: Allows access to individual audio frames for processing and analysis.
- Compatibility: Ensures compatibility with audio processing libraries that support the WAV format.

11. pocketsphinx:

- Speech Recognition Engine: Implements a lightweight and efficient speech recognition engine.
- Offline Speech Recognition: Enables speech recognition without the need for an internet connection.
- Continuous Speech Recognition: Supports continuous listening for processing longer speech segments.
  Customization: Allows customization of language models for improved recognition accuracy.

12. pydub:

- Audio Manipulation: Provides a high-level interface for manipulating audio data.
- Format Conversion: Converts between different audio formats, allowing seamless integration
- with other libraries.
- Audio Playback: Facilitates playing audio directly within the Python environment.
- Effects and Filters: Applies various effects and filters to modify audio characteristics.

13. librosa:

- Audio Analysis: Analyzes audio data for tasks such as feature extraction and visualization.
- Noise Reduction: Applies algorithms for reducing noise in audio signals.
- Time-Frequency Representation: Generates spectrograms and other time-frequency
- representations.
- Compatibility: Integrates seamlessly with other audio processing libraries.

14. noise reduce:

- Noise Reduction: Implements algorithms for reducing noise in audio signals.
- Adaptive Filtering: Adapts to varying noise conditions for effective noise reduction.
- Signal Processing: Utilizes signal processing techniques to enhance audio quality.
- Real-Time Applications: Applicable to real-time audio processing scenarios.

```python
def download_video_audio(video_url, video_directory):
    try:
        # Download the YouTube video using pytube
        yt = YouTube(video_url)
        yt.streams.filter(only_audio=True).first().download(output_path=video_directory, filename="vedio.mp4")
        return os.path.join(video_directory, "vedio.mp4")
    except Exception as download_error:
        print("Error while downloading the video:", download_error)
        return None
```

This function downloads the audio from a YouTube video using the pytube library.

1. Input:

- video_url: The URL of the YouTube video.
- video_directory: The directory where the downloaded video will be saved

2. Process:

- It creates a YouTube object from the provided video URL using the pytube library.
- Filters the available streams to include only the audio streams (only_audio=True).
- Chooses the first audio stream (assuming it's the best quality) and downloads it.
- The downloaded audio is saved as "vedio.mp4" in the specified video_directory.

3. Output:

- Returns the file path of the downloaded audio ("vedio.mp4") if successful.
- If an error occurs during the download, it prints an error message and returns None.

In essence, this function encapsulates the process of fetching the audio content from a YouTube video, making it ready for further processing in the audio summarization pipeline.

```
def convert_audio_to_wav(audio_file, video_directory):
    try:
        output_audio_file = os.path.join(video_directory, "audio.wav")
        clip = mp.AudioFileClip(audio_file)
        clip.write_audiofile(output_audio_file)
        return output_audio_file
    except Exception as conversion_error:
        print("Error while converting audio:", conversion_error)
        return None
```

This function converts an audio file to the WAV format using the moviepy library.

1. Input:

- audio_file: The path to the input audio file that needs to be converted.
- video_directory: The directory where the converted audio file will be saved.

2. Process:

- It defines the output path for the converted audio file as "audio.wav" in the specified video_directory.
- Utilizes moviepy'sAudioFileClip to create an audio clip from the input audio file.
- Writes the audio clip to the WAV format file specified in the output path.

3. Output:

- Returns the file path of the converted audio ("audio.wav") if successful.
- If an error occurs during the conversion, it prints an error message and returns None.

This function is part of the pipeline to ensure uniformity in the audio file format for further processing, specifically preparing the audio content for transcription and summarization.

```
def extract_audio(video_file, output_path):
    try:
        # Check if the video file exists
        if not os.path.exists(video_file):
            raise FileNotFoundError(f"Video file '{video_file}' not found.")

        # Open the video file
        video = mp.VideoFileClip(video_file)

        # Check if the video has a valid fps
        fps = getattr(video, 'fps', None)
        if fps is None:
            raise ValueError("Invalid video fps")

        # Define the output audio file path
        audio_file = os.path.join(output_path, "audio.wav")

        # Extract and write the audio to a WAV file
        video.audio.write_audiofile(audio_file)

        return audio_file
    except (FileNotFoundError, ValueError) as e:
        print(f"Error: {str(e)}")
    except Exception as e:
        print(f"Error extracting audio: {str(e)}")
    finally:
        # Close the video object in the finally block to ensure it is closed
        if 'video' in locals():
            video.close()

    return None
```

This function extracts the audio from a video file using the moviepy library.

1. Input:

- video_file: The path to the input video file from which audio will be extracted.
- output_path: The directory where the extracted audio file will be saved.

2. Process:

- Checks if the input video file exists; if not, raises a FileNotFoundError.
- Opens the video file using moviepy and checks if it has a valid frames per second (fps) attribute. If not, raises a ValueError.
- Defines the output path for the extracted audio file as "audio.wav" in the specified output_path.
- Extracts and writes the audio from the video t201a WAV file using moviepy's audio writing functional

3. Output:

- Returns the file path of the extracted audio ("audio.wav") if successful.
- If an error occurs during the extraction, it prints an error message and returns None.

This function is a crucial step in the audio processing pipeline, providing the audio content needed for transcription and subsequent summarization. It ensures that the audio is extracted uniformly from various video sources for consistent further processing.

```python
def extract_audio_ffmeg(video_file, output_path):
    try:
        # set the name of the output audio file
        audio_name = f"audio.wav"

        # set the full path of the output audio file
        audio_path = os.path.join(output_path, audio_name)

        # extract the audio from the video by using ffmpeg
        command = ['ffmpeg', '-i', video_file, '-vn', '-ar', '44100', '-ac', '2', '-sample_fmt', 's16', audio_path]
        subprocess.run(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

        return audio_path
    except Exception as e:
        print(f"Error extracting audio: {str(e)}")

    return None
```

This function extracts audio from a video file using the FFmpeg command-line tool.

1. Input:

- video_file: The path to the input video file from which audio will be extracted.
- output_path: The directory where the extracted audio file will be saved.

2. Process:

- Defines the name of the output audio file as "audio.wav."
- Constructs the full path of the output audio file by joining the specified output_path and the predefined audio name.
- Uses the FFmpeg command-line tool to execute a command for audio extraction. The command includes:
  - -i: Input file (the video file).
  - -vn: Disable video recording.
  - -ar 44100: Set audio sample rate to 44100 Hz.
  - -ac 2: Set audio channels to stereo.
  - -sample_fmt s16: Set the audio sample format to 16-bit.
- The output audio file path.

3. Output:

- Returns the file path of the extracted audio ("audio.wav") if the extraction is successful.
- If an error occurs during the extraction, it prints an error message and returns None.

This function provides an alternative method for extracting audio, utilizing FFmpeg's capabilities. It is a robust approach to handling various video formats and extracting high-quality audio for further processing.

```
def transcribe_audio(audio_file):
    if not os.path.exists(audio_file):
        print(f"Audio file '{audio_file}' does not exist.")
        return None

    # Check the format of the audio file and convert it to WAV if needed
    audio_format = audio_file.split('.')[-1]
    if audio_format != 'wav':
        try:
            # Convert the audio file to WAV format
            output_audio_file = audio_file.replace(audio_format, 'wav')
            audio = mp.AudioFileClip(audio_file)
            audio.write_audiofile(output_audio_file)  # Use 'pcm_s16le' codec for WAV format
            audio.close()
            audio_file = output_audio_file
        except Exception as e:
            print(f"Error converting audio to WAV format: {str(e)}")
            return None

    try:
        recognizer = sr.Recognizer()
        with sr.AudioFile(audio_file) as source:
            audio = recognizer.record(source)
        return recognizer.recognize_google(audio, language="en-US")
    except Exception as e:
        print(f"Error transcribing audio: {str(e)}")
        return None
```

This function transcribes the content of an audio file into text using Google's speech recognition service.

1. Input:

- audio_file: The path to the input audio file that needs to be transcribed.

2. Process:

- Checks if the specified audio file exists. If not, it prints an error message and returns None.
- Checks the format of the audio file, and if it's not in WAV format, it attempts to convert it.
- Creates an output audio file with a ".wav" extension using MoviePy library.
- Uses the SpeechRecognition library (recognizer) to transcribe the audio.
- Reads the audio file using sr.AudioFile.
- Records the audio from the file.
- Sends the recorded audio to Google's speech recognition service using recognizer.recognize_google.
- Specifies the language as "en-US" (English, United States).

3. Output:

- Returns the transcribed text if the process is successful.
- If there are errors during the transcription, it prints an error message and returns None.

This function ensures that the input audio file is in the required WAV format for accurate transcription. It leverages the SpeechRecognition library for interfacing with Google's speech recognition service, providing a straightforward way to convert spoken words into text.

```
def transcribe_audio_uncheck(audio_file):
    try:
        recognizer = sr.Recognizer()
        with sr.AudioFile(audio_file) as source:
            audio = recognizer.record(source)
            return recognizer.recognize_google(audio, language="en-US", show_all=True)['alternative'][0]['transcript'].strip
    except Exception as e:
        print(f"Error transcribing audio: {str(e)}")
        return None

def summarize_text(text):
    try:
        model = Summarizer()
        summary = model(text)
        return summary
    except Exception as e:
        print(f"Error summarizing text: {str(e)}")
        return None
```

Transcribe_audio_uncheck Function

This function performs an unchecked transcription of the content of an audio file into text using Google's speech recognition service. It returns the transcribed text without checking for errors.

1. Input:

- audio_file: The path to the input audio file that needs to be transcribed.

2. Process:

- Uses the SpeechRecognition library (recognizer) to transcribe the audio.
- Reads the audio file using sr.AudioFile.
- Records the audio from the file.
- Sends the recorded audio to Google's speech recognition service using recognizer.recognize_google.
- Specifies the language as "en-US" (English, United States).
- Uses show_all=True to retrieve alternative transcriptions.
- Extracts the first alternative's transcript, converts it to lowercase, and removes leading/trailing whitespaces.

3. Output:

- Returns the transcribed text if the process is successful.
- If there are errors during the transcription, it prints an error message and returns None.

summarize_text Function

This function generates a summary of the provided text using the Summarizer model.

1. Input:

- text: The input text that needs to be summarized.

2. Process:

- Uses the Summarizer library (model) to generate a summary of the input text.

3. Output:

- Returns the generated summary if the process is successful.
- If there are errors during the summarization, it prints an error message and returns None.

These functions contribute to the core functionality of summarizing spoken words into concise text summaries. The first function focuses on transcription, while the second one handles the summarization process.

```
def save_transcribed_text(transcribed_text):
    text_file_path = os.path.join(app.config['UPLOAD_FOLDER'], 'transcribed_text.txt')
    with open(text_file_path, 'w') as text_file:
        text_file.write(transcribed_text)
```

This function saves the transcribed text to a text file.

1. Input:

- transcribed_text: The text that needs to be saved.

2. Process:

- Constructs the full path to the text file using the app.config ['UPLOAD_FOLDER'] (upload folder defined in the Flask app).
- Opens the text file in write mode ('w').
- Writes the transcribed text to the file.

3. Output:

- Saves the transcribed text to a text file with the name 'transcribed_text.txt' in the specified upload folder.

This function is crucial for persisting the transcribed text, providing a record of the content that has been transcribed during the application's operation.

```python
# Summarize microphone audio
def summarize_microphone_audio():
    try:
        # Record audio from the microphone
        recognizer = sr.Recognizer()
        with sr.Microphone() as source:
            print("Say something...")
            audio = recognizer.listen(source)

        # Transcribe the recorded audio
        audio_text = recognizer.recognize_google(audio)
        if audio_text:
            summary = summarize_text(audio_text)
            if summary:
                return summary
            else:
                return "Summary generation failed."
        else:
            return "No speech detected."
    except sr.RequestError as e:
        return f"Could not request results: {str(e)}"
    except sr.UnknownValueError:
        return "No speech detected."
```

This function captures audio from the microphone, transcribes the spoken words using Google's Speech Recognition, and then generates a summary of the transcribed text using the summarize_text function.

1. Process:

- Initializes a Recognizer object from the speech_recognition library.
- Opens the microphone as a source for audio input using the with sr.Microphone() as source block.
- Prints a prompt to the user to say something.
- Listens to the audio input and records it.
- Uses Google's Speech Recognition (recognizer.recognize_google(audio)) to convert the recorded audio to text (audio_text).
- If audio text is detected:
- Generates a summary of the transcribed text using the summarize_text function.
- Returns the summary if successful.
- If summary generation fails, returns an appropriate message.
- If no speech is detected, returns a corresponding message.

2. Output:

- Summary of the transcribed audio if successful.
- Messages indicating failure in case of no speech detected or summary generation failure.

This function provides real-time summarization of spoken words from the microphone, making the application interactive and user-friendly.

```python
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/summarize_video_audio', methods=['POST'])
def summarize_video_audio():
    # Return the summary as a string
    data_folder = "without\\data\\download_for_url"
    if not os.path.exists(data_folder):
        os.makedirs(data_folder)

    youtube_url = request.form['youtube_url']
    video_file_path = download_video_audio(youtube_url, data_folder)

    if video_file_path:

        extracted_audio_file = extract_audio_ffmeg(video_file_path, data_folder)

        if extracted_audio_file:
            transcribed_text = transcribe_audio_uncheck(extracted_audio_file)

            if transcribed_text:
                summary = summarize_text(transcribed_text)

                if summary:
                    return summary
                else:
                    return "Summary generation failed."
            else:
                return "Transcription failed. Check the audio file for issues."
        else:
            return "Audio conversion failed."
    else:
        return "Video download failed. Check the video URL and your internet connection."
```

Index.html(For home page)

without > templates > index.html > html > body > div.container > form > div.form-group > input#youtube_url.form-control

```
<!DOCTYPE html>
<html>
<head>
    <title>Audio Summarizer</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    <!-- Add Font Awesome for icons -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
</head>
<body>
    <div class="container">
        <h1>Audio Summarizer</h1>
        <form action="/summarize_file_audio" method="POST" enctype="multipart/form-data">
            <div class="form-group">
                <label for="video_upload">
                    <i class="fas fa-upload"></i> Upload Video:
                </label>
                <input type="file" name="video_upload" id="video_upload" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary">
                Summarize Video
            </button>
        </form>
        <br>
        <form action="/summarize_video_audio" method="POST">
            <div class="form-group">
                <label for="youtube_url">
                    <i class="fab fa-youtube"></i> YouTube Video URL:
                </label>
                <input type="text" name="youtube_url" id="youtube_url" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary">
                Summarize YouTube Video
            </button>
        </form>
        <br>
        <form action="/summarize_microphone_audio" method="POST">
            <button type="submit" class="btn btn-primary">
                <i class="fas fa-microphone"></i> Summarize Microphone Audio
            </button>
        </form>
        <div class="result">
            <h3>Summary:</h3>
            <p>{{ result }}</p>
        </div>
    </div>
</body>
</html>

```

This HTML code defines the structure of a web page for the "Audio Summarizer" application. Here's a breakdown of its components:

1. Document Type Declaration:

- <!DOCTYPE html> specifies the HTML version used in the document.

2. HTML Structure:

- The HTML document is wrapped in <html> tags.

3. Head Section:

- Contains metadata and links to external resources.
- Title "Audio Summarizer."
- Stylesheets:
  - "style.css" for custom styles.
  - Bootstrap 4.5.0 CSS.
  - Font Awesome for icons.

4. Body Section:

- <body> contains the main content of the page.

5. Container Division:

- wraps the entire content for styling.

6. Page Title:

- <h1> displays the title "Audio Summarizer."

7. File Upload Form:

- <form> for uploading video files.
- "Upload Video" label and a file input with the ID "video_upload."
- Submit button labeled "Summarize Video."

8. YouTube Video URL Form:

- <form> for summarizing YouTube videos.
- "YouTube Video URL" label and a text input with the ID "youtube_url."
- Submit button labeled "Summarize YouTube Video."

9. Microphone Audio Form:

- <form> for summarizing microphone audio.
- Submit button labeled "Summarize Microphone Audio" with a microphone icon.

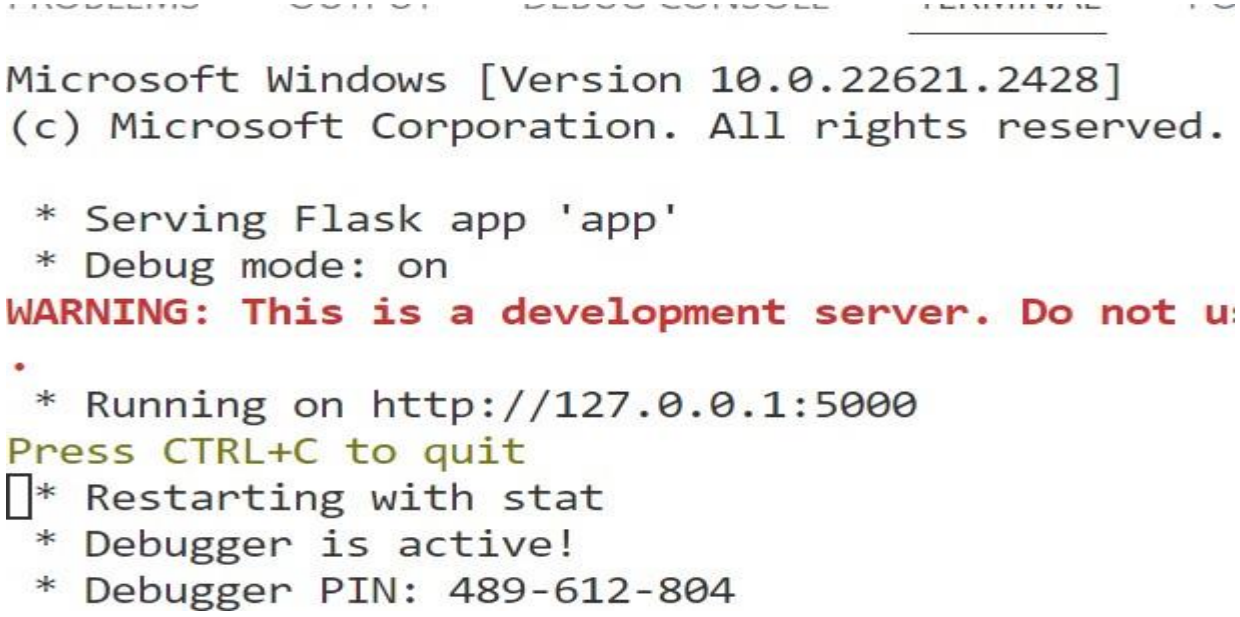
10. Result Display:

- for displaying the summary.
- <h3> for the "Summary" heading.
- <p> to display the actual summary fetched from the Flask application ({{ result }}).

This HTML structure provides a user interface for interacting with different audio summarization features. The forms allow users to upload video files, input YouTube video URLs, and summarize microphone audio, with the results displayed on the page.

Output:

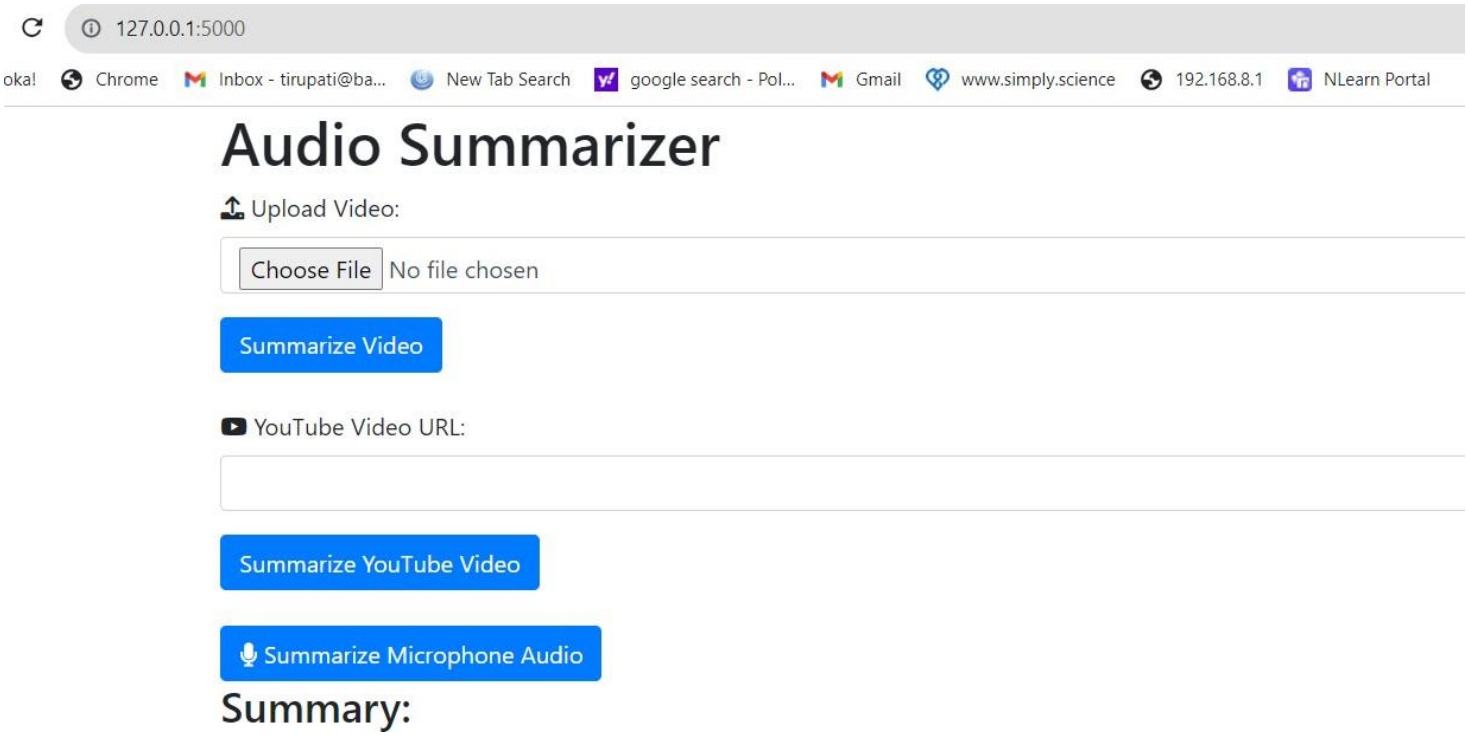
Compile:C:/Users/Sivamani/AppData/Local/Microsoft/WindowsApps/python 3.11.exe "c:/Users/Sivamani/OneDrive/Desktop/Summarizing Audio Files in Python/without/app.py"



Open your web browser and type "http://127.0.0.1:5000" in the address bar. Press Enter, and it will redirect you to the "index.html" page, prompting you to enter an

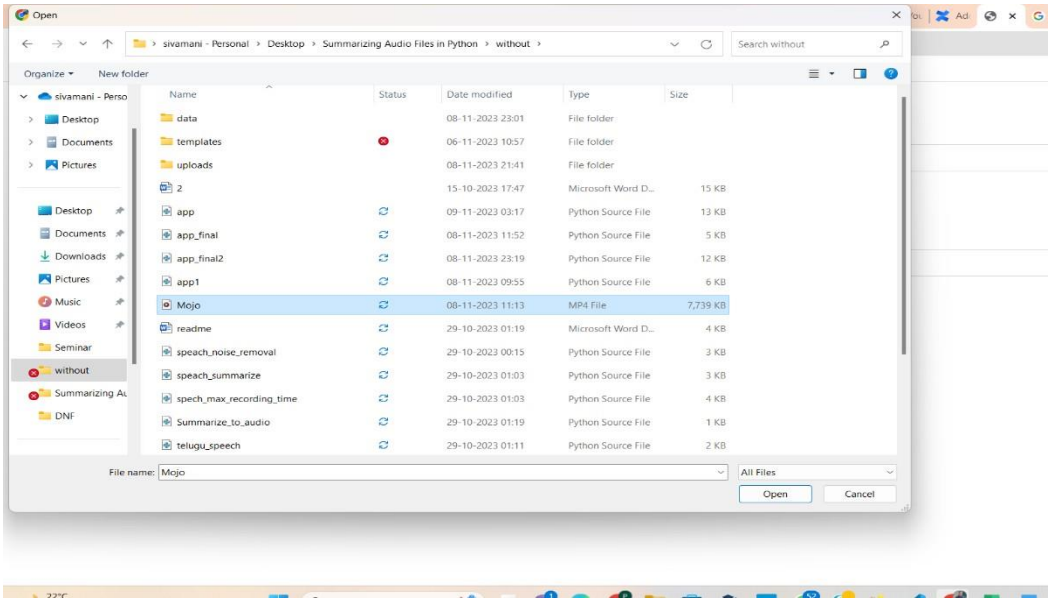
RUN:

Redirectstoindex.html

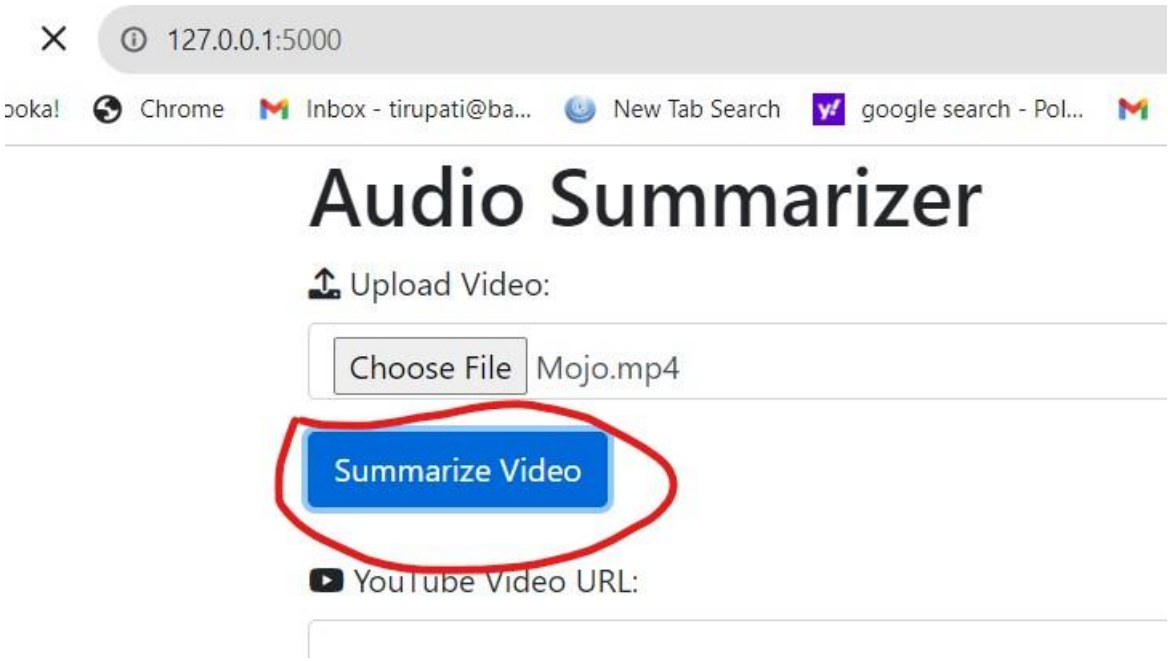


Next You can upload a file by selecting "Choose File" and selecting the audio or video file you want to summarize

Choose a video file from your device by clicking "Choose File" and selecting the desired video file from your directory.



I. Click on Summarize Video



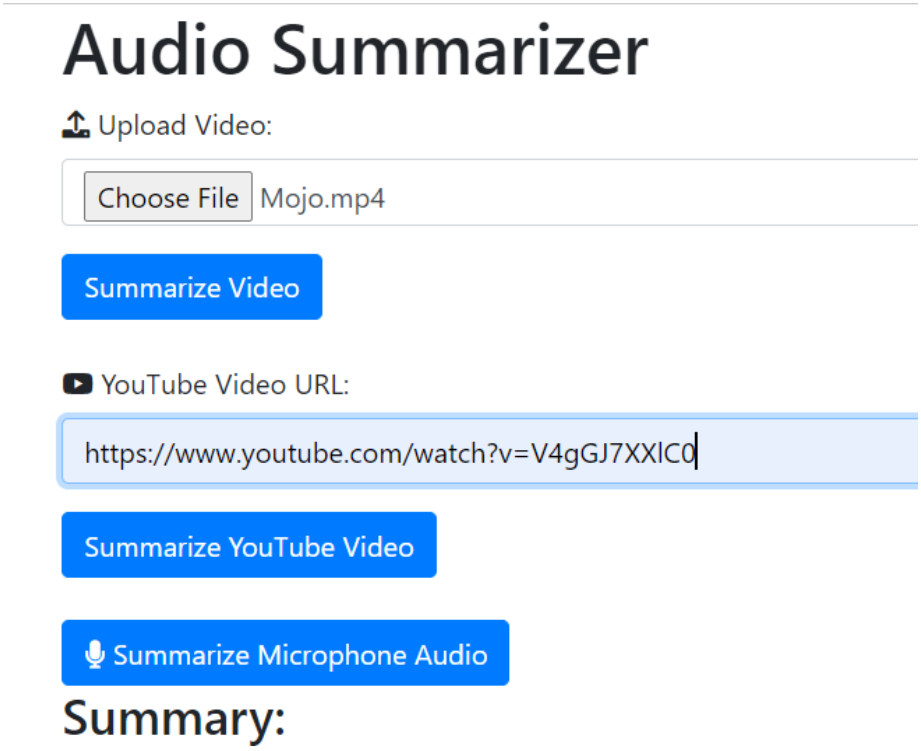
II. Output: The summarize of the upload file will be display on the new website page

erase may 4:23 and you are watching the covid Python is a wonderful language for productive program in but has one big problem is too slow and going slow means you get made fun of breast in C plus plus chats of the world but the tables are about to turn thanks to a brand new programming language called mojo a superset of python is not just two times faster not 10 times faster but up to 35 thousand times faster than the dominant language for artificial intelligence but behind the current company founded by Chris programming language to take advantage of language

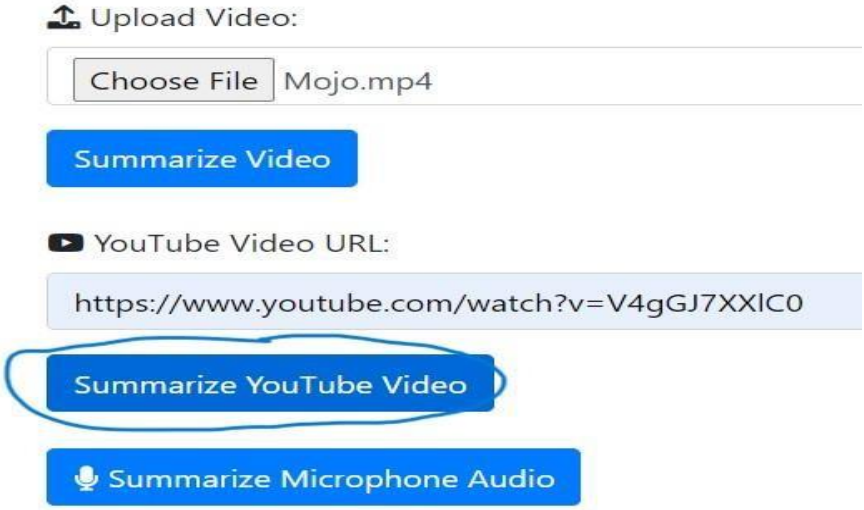
III. Then Return to the Home Page

Paste a YouTube video URL into the provided field and click "Summarize YouTube Video" to generate a summary.

1. Paste the url of any vedio



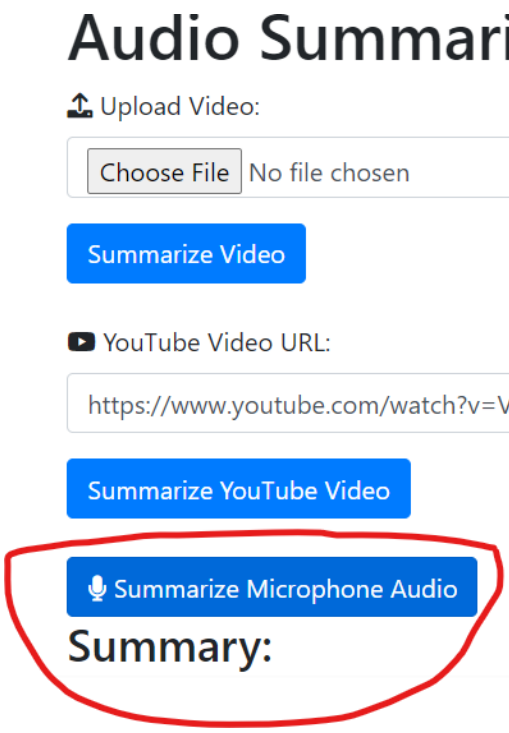
2. Click on the Summarize the youTube video



3. Output: The summarize of the upload file will be display on the new website page

4. Then Return to the Home Page

i. Click "Summarize Microphone Audio" and speak into your microphone. The system will transcribe and summarize your spoken words.



ii.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

In this project, developed an Audio Summarizer web application using Flask, providing users with a convenient platform to summarize audio content from various sources. The application seamlessly integrates functionalities like video file uploads, YouTube video URL summarization, and real-time microphone audio summarization.

By leveraging powerful libraries such as pytube, moviepy, speech_recognition, and noisereduce, our application achieves robust audio processing, including extraction, transcription, and noise reduction. The clean and intuitive user interface, designed with HTML and CSS, enhances user experience with responsive design and consistent styling.

In summary, our Audio Summarizer demonstrates the effective fusion of web development and audio processing technologies, offering a versatile tool for users to extract insights from diverse audio formats. This project serves as an excellent showcase of Python's capabilities in creating user-friendly applications for audio content analysis.

5.2 FUTURE SCOPE

The future scope for the "Summarizing Audio Files in Python" project entails several avenues for advancement and refinement. This includes enhancing audio processing accuracy, integrating advanced speech recognition models, optimizing real-time performance, exploring multimodal summarization techniques, personalizing summaries, integrating with NLP technologies, developing interactive interfaces, tailoring summarization for specific domains, optimizing scalability and deployment, and conducting thorough evaluation and benchmarking studies. By focusing on these areas, the project can evolve to meet the growing demand for efficient and versatile audio summarization solutions, catering to diverse needs and applications effectively.

5% Overall Similarity

Top sources found in the following databases:

- **1% Internet database**
- **Crossref database**
- **5% Submitted Works database**
- **0% Publications database**
- **Crossref Posted Content database**

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Southampton Solent University on 2023-09-25 Submitted works	1%
2	University of Hertfordshire on 2023-01-16 Submitted works	<1%
3	Babes-Bolyai University on 2023-06-17 Submitted works	<1%
4	Letterkenny Institute of Technology on 2023-08-31 Submitted works	<1%
5	University of Wales, Bangor on 2023-10-21 Submitted works	<1%
6	coddyschool.com Internet	<1%
7	Manipal University on 2024-02-13 Submitted works	<1%
8	British University in Egypt on 2022-12-22 Submitted works	<1%

9	ethesis.nitrkl.ac.in Internet	<1%
10	University of East London on 2023-05-11 Submitted works	<1%
11	Queen Mary and Westfield College on 2023-10-15 Submitted works	<1%
12	University of Liverpool on 2023-12-01 Submitted works	<1%