

Project Title:

**Voice Recognition System**

Submitted by:

**Ebaa Haq [2021-CE-22]**

**Faiza Riaz [2021-CE-20]**

**Maham Nadeem [2021-CE-10]**

Submitted to:

**Raja Muzammil Munir**

Course:

**CMPE-341L Artificial Intelligence**

Semester:

**Fall 2023 (5th)**

Date of Submission:

**[December 22, 2023]**

**Department of Computer Engineering**

**University of Engineering and Technology, Lahore**

Table of Content

[1. Title of the project 3](#_Toc154252575)

[2. Objectives 3](#_Toc154252576)

[3. System requirements 3](#_Toc154252577)

[4. Project planning 3](#_Toc154252579)

[5. Flowchart 6](#_Toc154252580)

[6. Problems 7](#_Toc154252581)

[7. Voice Recognition System Implementation 7](#_Toc154252582)

[8. Working of Project Code 14](#_Toc154252583)

[9. Output of Project 16](#_Toc154252584)

[10. Applications of the Voice Recognition System 19](#_Toc154252585)

[11. References 20](#_Toc154252586)

1. **Title of the project**

The project work presented in thisreport entitled “**Voice Recognition System**” is submitted to the Department of Computer Engineering to Sir Raja Muzammil as the 5th Semester (FALL 23) final project.

1. **Objectives**

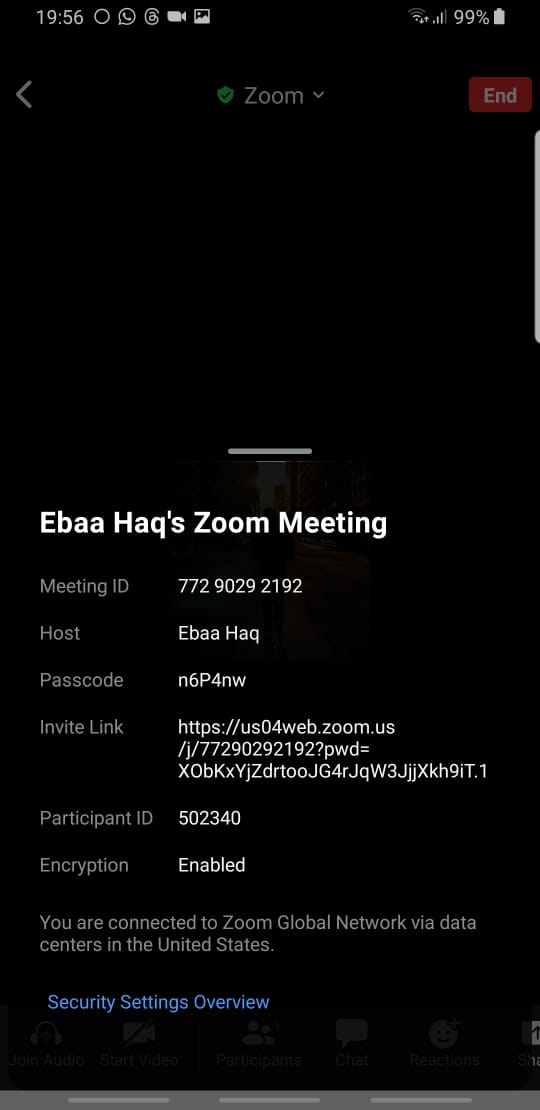
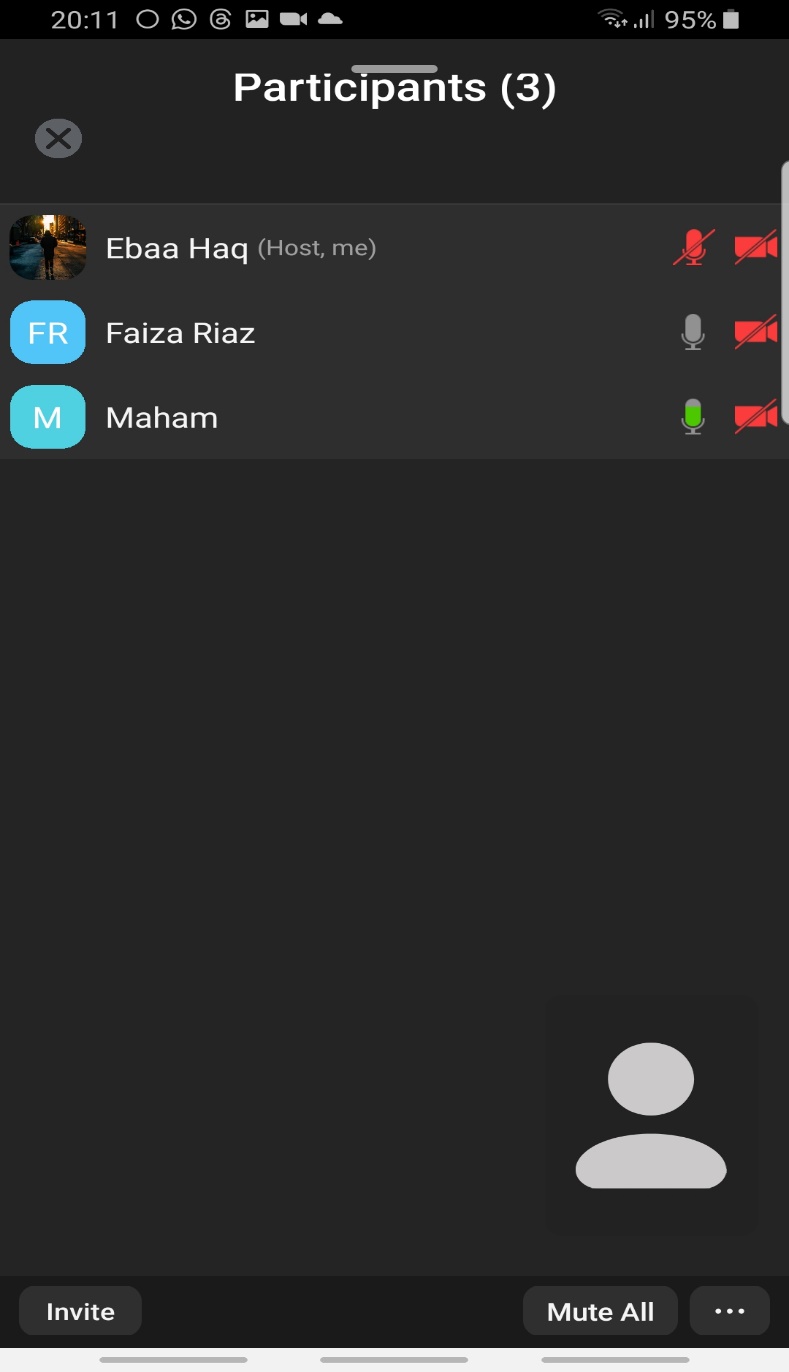
The objective of our artificial intelligence semester project is to create a user-friendly Voice Recognition System using Python. We aim to employ Gaussian Mixture Models (GMMs) for speaker modeling, training the system to distinguish between different speakers based on Mel Frequency Cepstral Coefficients (MFCC) extracted from audio signals. The project includes features like persistent model storage, testing functionality with recorded audio, and speech-to-text conversion. Integration with external libraries such as sounddevice, speech\_recognition, and python\_speech\_features enhances the system's capabilities. The user interface, built with Tkinter, provides buttons for training, recording, browsing files, and testing, offering a responsive and intuitive experience. Through thorough documentation and a detailed report, we plan to showcase the application of artificial intelligence concepts in developing a practical voice recognition system, demonstrating its functionality and significance in real-world scenarios.

1. **System requirements**

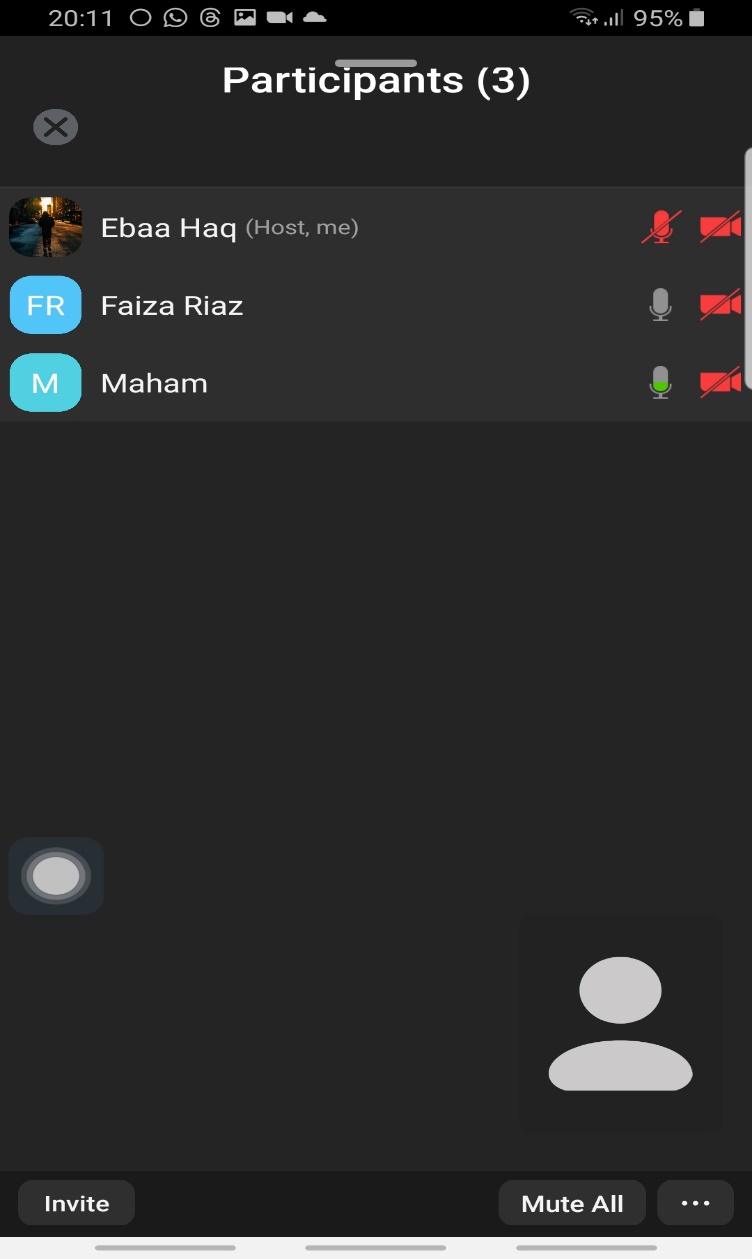
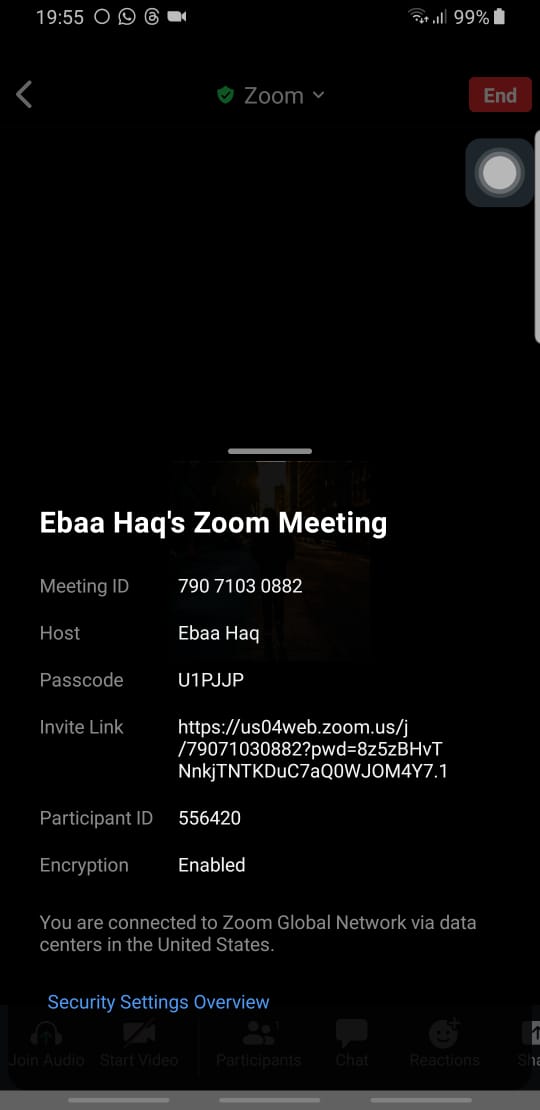
The system requirements for our project comprised both hardware and software components. On the hardware side, a laptop or PC was essential. For the software aspect, we selected Windows 10 as the operating system, providing a stable and widely used platform. Python was chosen as the primary programming language due to its versatility and ease of use, serving as the front end of the project. The Integrated Development Environment (IDE) utilized for coding and development was PyCharm, a robust application offering comprehensive features for software development. Together, these components formed the foundation for building our Voice Recognition System, ensuring compatibility and efficiency in the development process.

1. **Project planning**

Project planning for the Voice Recognition System was meticulously carried out through a series of organized meetings, with a significant session held on 24th November 2023 titled **“Project Progress and Next Steps Discussion.”**



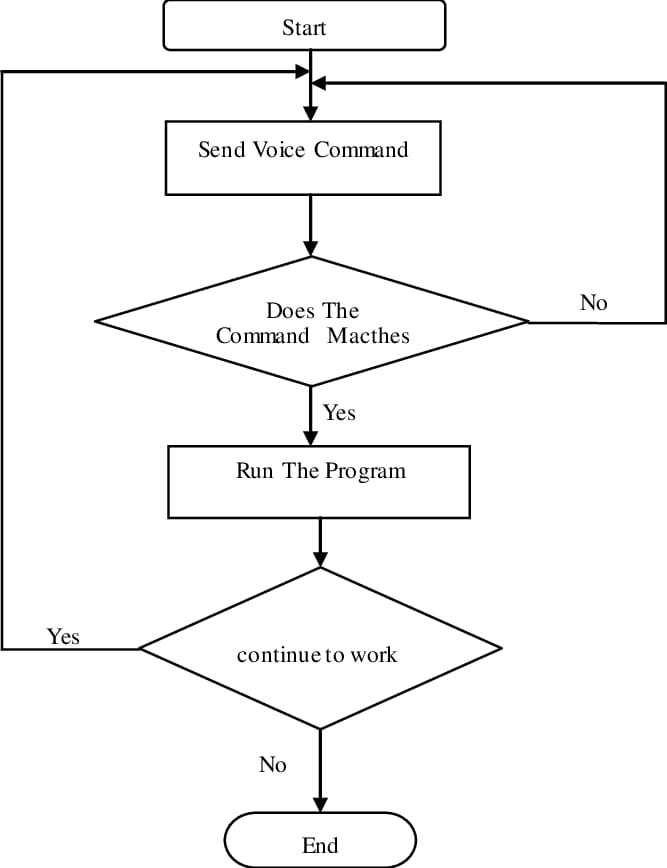
This meeting served as a pivotal point for project stakeholders to engage in detailed discussions, defining core functionalities, desired features, and overall system objectives. Comprehensive analysis of the Voice Recognition processes took place, accompanied by deliberations on the project's timeline, resource allocation, and potential challenges. The outcome of this meeting played a crucial role in shaping the project plan, fostering collaboration, and ensuring alignment among the project team towards the successful implementation of the **Voice Recognition System**.



The second key meeting on 8th December 2023, titled **“Voice to Text Conversion and GUI,”** facilitated productive discussions among project stakeholders, encompassing various crucial aspects. In addressing the Voice to Text Conversion component, we explored techniques and challenges associated with converting audio signals to text, ensuring accurate and reliable transcription. The Graphical User Interface (GUI) development was a central focus, aiming to create an intuitive and user-friendly interface for seamless interaction with the **Voice Recognition System**. Discussions involved the design elements, layout, and features that would enhance the user experience. Brainstorming sessions were conducted to make informed decisions regarding the file handling approach, ensuring efficiency in managing system data. Additionally, tasks and responsibilities related to both file handling and GUI development were meticulously assigned, marking significant progress in the project planning and execution phases. In this session, we also overviewed the report requirements, including structure, content, and formatting guidelines. Assigned specific responsibilities and deadlines for report completion and GitHub uploading. Lastly addressed any other relevant topics or questions regarding the project.

1. **Flowchart**

A process flow chart is a logical, relatively easy-to-understand chart, which displays how a process operates by using standard symbols to represent activity. The various activities and decisions are linked together by arrows showing how the process flows between one activity and the next and flows before and after decisions are taken. So, creating a flowchart of **Voice Recognition System** is an excellent way to visually represent the logical sequence of activities and decisions within the system. Here's the flowchart of **Voice Recognition System:**



1. **Problems**

While developing the Voice Recognition System, our team faced various challenges, such as dealing with the complexities of processing audio signals, integrating external libraries for both audio processing and GUI development, training models to identify speakers, creating a user-friendly interface, managing file operations effectively, ensuring accurate speech recognition despite variations, optimizing resource usage, debugging complex interactions between different parts of the system, and thoroughly documenting the code. Additionally, collecting diverse and relevant training data posed a significant challenge. Overcoming these hurdles not only enhanced our technical skills but also sharpened our problem-solving abilities, crucial for success in the fields of artificial intelligence and software development.

1. **Voice Recognition System Implementation**

**7.1 Overview of Functionalities**

The Voice Recognition System implemented in this project encompasses essential features such as training the model, recording and testing audio, and presenting results through a user-friendly graphical interface. The system integrates Mel Frequency Cepstral Coefficients (MFCC) for audio signal processing, Gaussian Mixture Models (GMMs) for speaker identification, and Google's speech recognition service for converting audio to text.

**7.2 Libraries and Technologies**

The project leverages several Python libraries, including Tkinter for GUI development, sounddevice for audio handling, and key audio processing libraries such as scipy.io.wavfile, speech\_recognition, and python\_speech\_features. Additionally, machine learning functionalities are supported by numpy, sklearn, and external tools like Google's speech recognition service.

**7.3 Global Variables**

Global variables models and speakers play a crucial role in storing trained Gaussian Mixture Models and corresponding speaker names, respectively, providing a centralized reference for speaker identification.

**7.4 User Interaction and GUI**

User interaction is facilitated through a Tkinter-based GUI that includes buttons for training the model, recording audio, browsing files, and testing the system. Text widgets display recognized text and detected speakers, enhancing the user experience.

**7.5 Audio Processing and Feature Extraction**

The code employs Mel Frequency Cepstral Coefficients (MFCC) for audio signal processing, extracting features that capture essential information from the recorded audio signals. The process ensures the system's ability to discern distinct speaker characteristics.

**7.6 Model Training and Testing**

Gaussian Mixture Models (GMMs) are utilized for speaker identification, with a specific focus on training the system and subsequently testing its ability to recognize speakers accurately. The system also integrates Google's speech recognition service for converting audio to text during testing.

**7.7 File Handling**

The code efficiently handles file operations, allowing users to browse test files, creating text files containing speaker names, and managing training and testing sets. This ensures seamless data organization and retrieval.

**7.8 Error Handling**

The implementation incorporates robust error handling mechanisms, including the use of try-except blocks, to manage exceptions during the testing phase, contributing to the system's stability and reliability.

**7.9 Documentation Practices**

The code includes comprehensive comments and documentation, providing clarity and context for different sections. This documentation enhances code readability and aids in understanding the implemented functionalities.

*"""  
Created on 25th November 2023 18:56:34  
  
 Project Title:  
 Voice Recognition System  
  
 Submitted by:  
  
 Ebaa Haq 2021-CE-22  
 Faiza Riaz 2021-CE-20  
 Maham Nadeem 2021-CE-10  
  
 Submitted to:  
 Raja Muzammil Munir  
  
 Course:  
 CMPE-341L Artificial Intelligence  
  
 Semester:  
 Fall 2023 (5th)  
  
  
"""*import tkinter as tk  
from tkinter import ttk  
from tkinter import filedialog  
from tkinter import messagebox  
import os  
import time  
import pickle  
import numpy as np  
from scipy.io.wavfile import read, write  
import sounddevice as sd  
import speech\_recognition as sr  
import python\_speech\_features as mfcc  
from sklearn import preprocessing  
from sklearn.mixture import GaussianMixture  
  
# Define models and speakers globally  
models = []  
speakers = []  
  
class VoiceRecognitionApp:  
 def \_\_init\_\_(self, root):  
 # Initialize the GUI window  
 self.root = root  
 self.root.title("Voice Recognition System")  
 self.root.geometry("600x500") # Set the window size  
  
 # Configure a style for colored buttons  
 self.style = ttk.Style()  
 self.style.configure("TButton", background="#4CAF50", foreground="black")  
  
 # Buttons for training, recording, and testing  
 self.train\_button = ttk.Button(root, text="Train Model", command=self.train\_model)  
 self.train\_button.pack(pady=10)  
  
 self.record\_button = ttk.Button(root, text="Record for Test", command=self.record\_for\_test)  
 self.record\_button.pack(pady=10)  
  
 # Entry widget for user to enter the name of the recorded audio file  
 self.recorded\_audio\_name\_var = tk.StringVar()  
 self.recorded\_audio\_name\_entry = ttk.Entry(root, textvariable=self.recorded\_audio\_name\_var)  
 self.recorded\_audio\_name\_entry.pack(pady=5, padx=10)  
  
 # Button to browse for a test audio file  
 self.browse\_button = ttk.Button(root, text="Browse", command=self.browse\_test\_file)  
 self.browse\_button.pack(pady=5)  
  
 # Entry widget to display the selected test file path  
 self.test\_file\_path\_var = tk.StringVar()  
 self.test\_file\_entry = ttk.Entry(root, textvariable=self.test\_file\_path\_var, state="readonly")  
 self.test\_file\_entry.pack(pady=10)  
  
 # Text widget with vertical scrollbar for displaying detected speaker  
 self.detected\_speaker\_frame = ttk.Frame(root)  
 self.detected\_speaker\_frame.pack(pady=10)  
  
 self.detected\_speaker\_scrollbar = ttk.Scrollbar(self.detected\_speaker\_frame, orient="vertical")  
 self.detected\_speaker\_scrollbar.pack(side="right", fill="y")  
  
 self.detected\_speaker = tk.Text(self.detected\_speaker\_frame, height=1, width=50,  
 yscrollcommand=self.detected\_speaker\_scrollbar.set)  
 self.detected\_speaker.pack(side="left")  
  
 self.detected\_speaker\_scrollbar.config(command=self.detected\_speaker.yview)  
  
 # Text box with vertical scrollbar for displaying recognized text  
 self.text\_display\_frame = ttk.Frame(root)  
 self.text\_display\_frame.pack(pady=10)  
  
 self.text\_display\_scrollbar = ttk.Scrollbar(self.text\_display\_frame, orient="vertical")  
 self.text\_display\_scrollbar.pack(side="right", fill="y")  
  
 self.text\_display = tk.Text(self.text\_display\_frame, height=4, width=50,  
 yscrollcommand=self.text\_display\_scrollbar.set)  
 self.text\_display.pack(side="left")  
  
 self.text\_display\_scrollbar.config(command=self.text\_display.yview)  
  
 # Button to test the model  
 self.test\_button = ttk.Button(root, text="Test Model", command=self.test\_model, style="TButton")  
 self.test\_button.pack(pady=10)  
 def train\_model(self):  
 # Train the voice recognition model  
 write\_names()  
 train\_model()  
 messagebox.showinfo("Training Completed", "Model training completed successfully.")  
  
 def record\_for\_test(self):  
 # Record audio for testing  
 self.record\_button.config(state="disabled")  
 self.test\_button.config(state="disabled")  
  
 fs = 44100  
 duration = 10  
 print("Recording Started...")  
  
 global test\_audio  
 test\_audio = sd.rec(frames=duration \* fs, samplerate=fs, channels=2)  
 sd.wait()  
 print("Recording Ended...")  
  
 # Get the entered name from the Entry widget  
 audio\_name = self.recorded\_audio\_name\_var.get()  
  
 # Check if the name is provided, otherwise use a default name  
 if not audio\_name:  
 audio\_name = "default\_recording"  
  
 # Create the 'recorded\_audio' folder if it does not exist  
 audio\_folder = "recorded\_audio"  
 if not os.path.exists(audio\_folder):  
 os.makedirs(audio\_folder)  
  
 # Save the recorded audio with the entered name  
 audio\_path = os.path.join(audio\_folder, f"{audio\_name}.wav")  
 write(audio\_path, 44100, test\_audio)  
  
 self.record\_button.config(state="normal")  
 self.test\_button.config(state="normal")  
  
 def browse\_test\_file(self):  
 # Browse for a test audio file  
 file\_path = filedialog.askopenfilename(filetypes=[('WAV Files', '\*.wav')])  
 if file\_path:  
 self.test\_file\_path\_var.set(file\_path)  
  
 def test\_model(self):  
 # Test the trained models on the selected test audio file  
 file\_path = self.test\_file\_path\_var.get()  
 if not file\_path:  
 messagebox.showerror("Error", "Please select a test file.")  
 return  
  
 try:  
 # Read the test audio file  
 sr, audio = read(file\_path)  
 vector = extract\_features(audio, sr)  
  
 log\_likelihood = np.zeros(len(models))  
  
 # Calculate the log likelihood for each model  
 for i in range(len(models)):  
 gmm = models[i]  
 scores = np.array(gmm.score(vector))  
 log\_likelihood[i] = scores.sum()  
  
 # Select the speaker with the highest likelihood  
 winner = np.argmax(log\_likelihood)  
 recognized\_speaker = speakers[winner]  
 print("\n>> detected as - ", recognized\_speaker)  
  
 # Display the detected speaker in the text widget  
 self.detected\_speaker.delete(1.0, tk.END) # Clear previous text  
 self.detected\_speaker.insert(tk.END, f"Detected as: {recognized\_speaker}")  
  
 # Convert audio to text using Google Speech Recognition  
 text = convert\_voice\_to\_text(file\_path)  
 print(f"\n>> Recognized text: {text}")  
  
 # Display the recognized text in the text widget  
 self.text\_display.delete(1.0, tk.END) # Clear previous text  
 self.text\_display.insert(tk.END, f"Recognized text: {text}")  
  
 time.sleep(1.0)  
  
 except Exception as e:  
 messagebox.showerror("Error", f"An error occurred during testing: {str(e)}")  
  
def write\_names():  
 # Write names of speakers in the training and testing sets to text files  
 source\_dir = "./training\_set/"  
 train\_file = "./training\_set\_addition.txt"  
 file = open(train\_file, "w")  
 for i in os.listdir(source\_dir):  
 file.writelines(i + '\n')  
  
 source\_dir = "./testing\_set/"  
 test\_file = "./testing\_set\_addition.txt"  
 file = open(test\_file, "w")  
 for i in os.listdir(source\_dir):  
 file.writelines(i + '\n')  
  
def calculate\_delta(array):  
 # Calculate delta coefficients from the MFCC features  
 rows, cols = array.shape  
 deltas = np.zeros((rows, 20))  
 N = 2  
 for i in range(rows):  
 index = []  
 j = 1  
 while j <= N:  
 if i - j < 0:  
 first = 0  
 else:  
 first = i - j  
 if i + j > rows - 1:  
 second = rows - 1  
 else:  
 second = i + j  
 index.append((second, first))  
 j += 1  
 deltas[i] = (array[index[0][0]] - array[index[0][1]] + (2 \* (array[index[1][0]] - array[index[1][1]]))) / 10  
 return deltas  
  
def extract\_features(audio, rate):  
 # Extract MFCC features from audio  
 mfcc\_feature = mfcc.mfcc(audio, rate, winlen=0.025, winstep=0.01, numcep=20, nfft=1200, appendEnergy=True)  
 mfcc\_feature = preprocessing.scale(mfcc\_feature)  
 delta = calculate\_delta(mfcc\_feature)  
 combined = np.hstack((mfcc\_feature, delta))  
 return combined  
  
def train\_model():  
 # Train the voice recognition model using Gaussian Mixture Models (GMM)  
 write\_names()  
 source = "./training\_set/"  
 dest = "./trained\_models/"  
 train\_file = "./training\_set\_addition.txt"  
  
 file\_paths = open(train\_file, 'r')  
 count = 1  
 features = np.asarray(())  
  
 for path in file\_paths:  
 path = path.strip()  
 sr, audio = read(source + path)  
 vector = extract\_features(audio, sr)  
  
 if features.size == 0:  
 features = vector  
 else:  
 features = np.vstack((features, vector))  
  
 if count == 1:  
 gmm = GaussianMixture(n\_components=7, max\_iter=200, covariance\_type='diag', n\_init=3)  
 gmm.fit(features)  
 picklefile = path.split("-")[0] + ".gmm"  
 pickle.dump(gmm, open(dest + picklefile, 'wb'))  
 print('>> modeling completed for speaker:', picklefile, " with data point = ", features.shape)  
 features = np.asarray(())  
 count = 0  
 count = count + 1  
  
def convert\_voice\_to\_text(audio\_path):  
 # Convert audio to text using Google Speech Recognition  
 recognizer = sr.Recognizer()  
  
 with sr.AudioFile(audio\_path) as source\_audio:  
 audio\_data = recognizer.record(source\_audio)  
  
 try:  
 text = recognizer.recognize\_google(audio\_data)  
 return text  
 except sr.UnknownValueError:  
 print("Speech Recognition could not understand audio")  
 return ""  
 except sr.RequestError as e:  
 print(f"Could not request results from Google Speech Recognition service; {e}")  
 return ""  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = VoiceRecognitionApp(root)  
  
 # Load the models and speakers  
 modelpath = "./trained\_models/"  
 gmm\_files = [os.path.join(modelpath, fname) for fname in os.listdir(modelpath) if fname.endswith('.gmm')]  
 models = [pickle.load(open(fname, 'rb')) for fname in gmm\_files]  
 speakers = [fname.split("\\")[-1].split(".gmm")[0] for fname in gmm\_files]  
  
 root.mainloop()

1. **Working of Project Code**

The Voice Recognition System is designed to recognize speakers and convert audio recordings to text through the integration of audio processing, machine learning, and graphical user interface (GUI) development. The code, written in Python using the Tkinter library for GUI, sound device for audio handling, and various audio processing and machine learning libraries, follows a structured workflow.

**8.1 Initialization and GUI Setup:**

The code initializes by creating a Tkinter application and instantiating the VoiceRecognitionApp class. This sets up the graphical user interface, providing buttons for critical functionalities such as training the model, recording audio, browsing files, and testing the system.

**8.2 Model Training:**

When the "Train Model" button is clicked, the train\_model function is triggered. This function writes speaker names to text files, reads audio files from the training set, and extracts features using Mel Frequency Cepstral Coefficients (MFCC). Gaussian Mixture Models (GMMs) are trained for speaker identification, and the trained models are stored for later use.

**8.3 Recording for Testing:**

The "Record for Test" button initiates the record\_for\_test function. This function records audio for a specified duration, saves the audio file, and plays it back using the sounddevice library. The recorded audio is then used for testing.

**8.4 Testing the Model:**

Clicking the "Test Model" button triggers the test\_model function. This function reads the selected test file, extracts audio features, and applies the trained GMM models to recognize the speaker. The detected speaker is displayed in the GUI text widget. Additionally, the code utilizes Google's speech recognition service to convert the audio to text, displaying the recognized text in the GUI.

**8.5 File Handling:**

File handling is a crucial aspect of the system. The write\_names function creates text files containing speaker names from the training and testing sets. The browse\_test\_file function allows users to select a test file for evaluation.

**8.6 Audio Processing and Feature Extraction:**

The extract\_features function processes audio signals using MFCC, scales the features, and calculates delta features. This comprehensive feature extraction captures relevant characteristics of the audio for both training and testing phases.

**8.7 Speech to Text Conversion:**

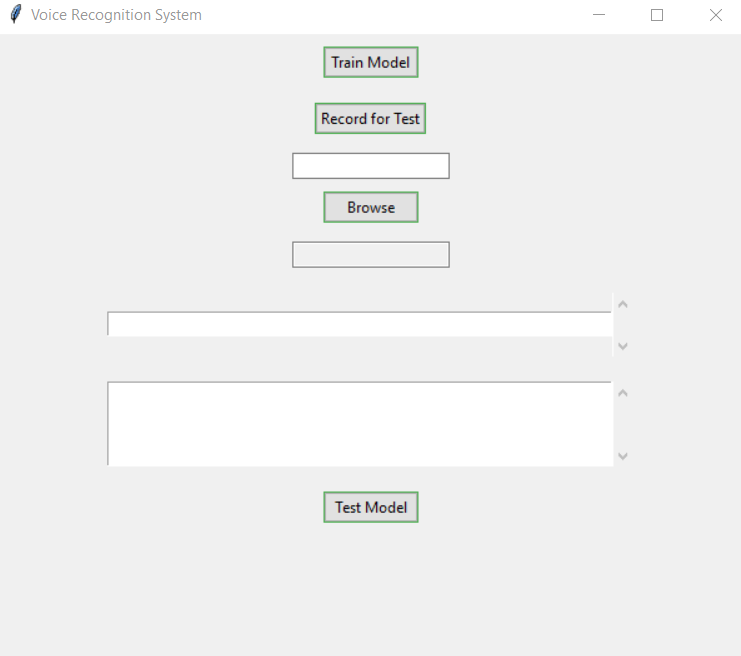
The convert\_voice\_to\_text function utilizes the SpeechRecognition library for converting recorded audio to text. The code handles potential recognition errors and displays the recognized text in the GUI.

**8.8 Displaying Results in GUI:**

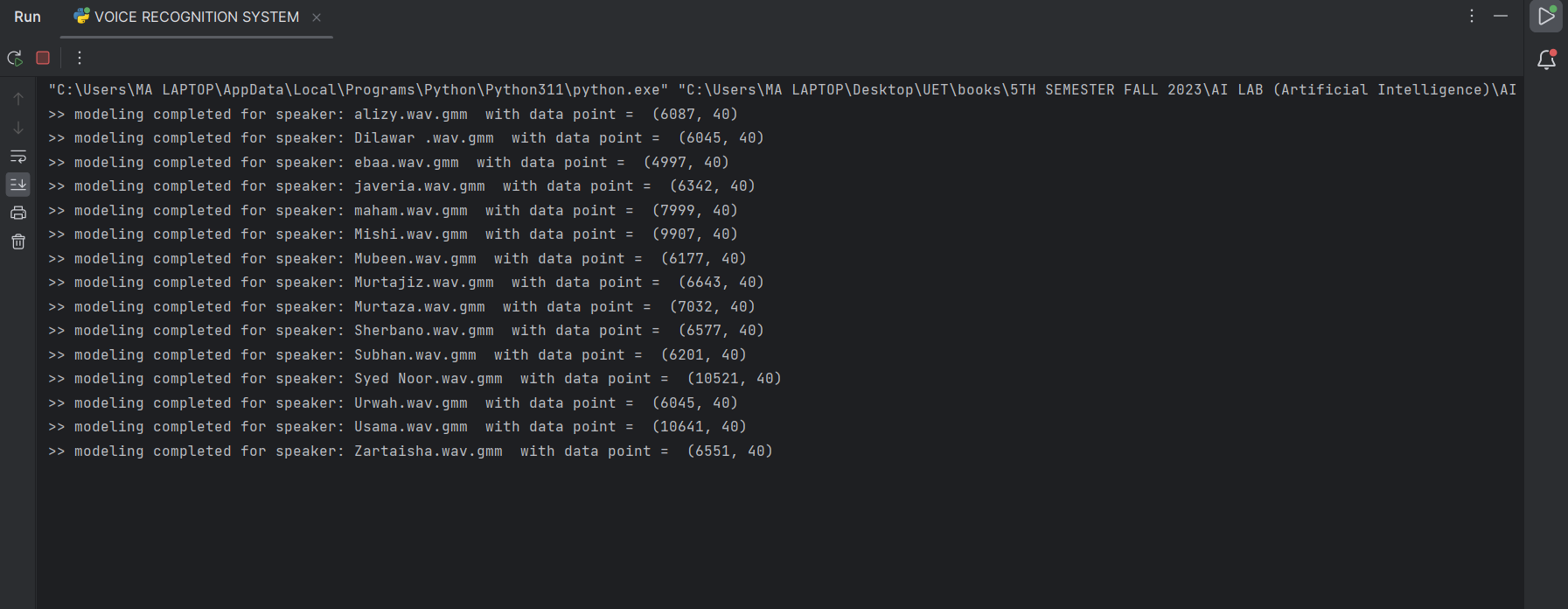
Results of the speaker recognition and the recognized text are prominently displayed in the Tkinter GUI. Two text widgets showcase the detected speaker and the converted text, providing a user-friendly interface for interaction.

1. **Output of Project**

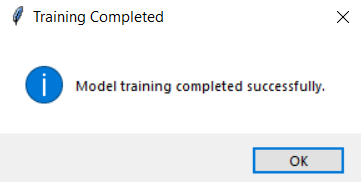
The Output of the project is:



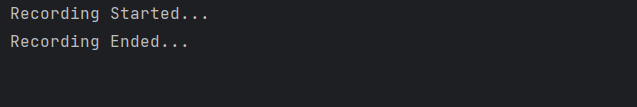
By clicking Train Model Button:



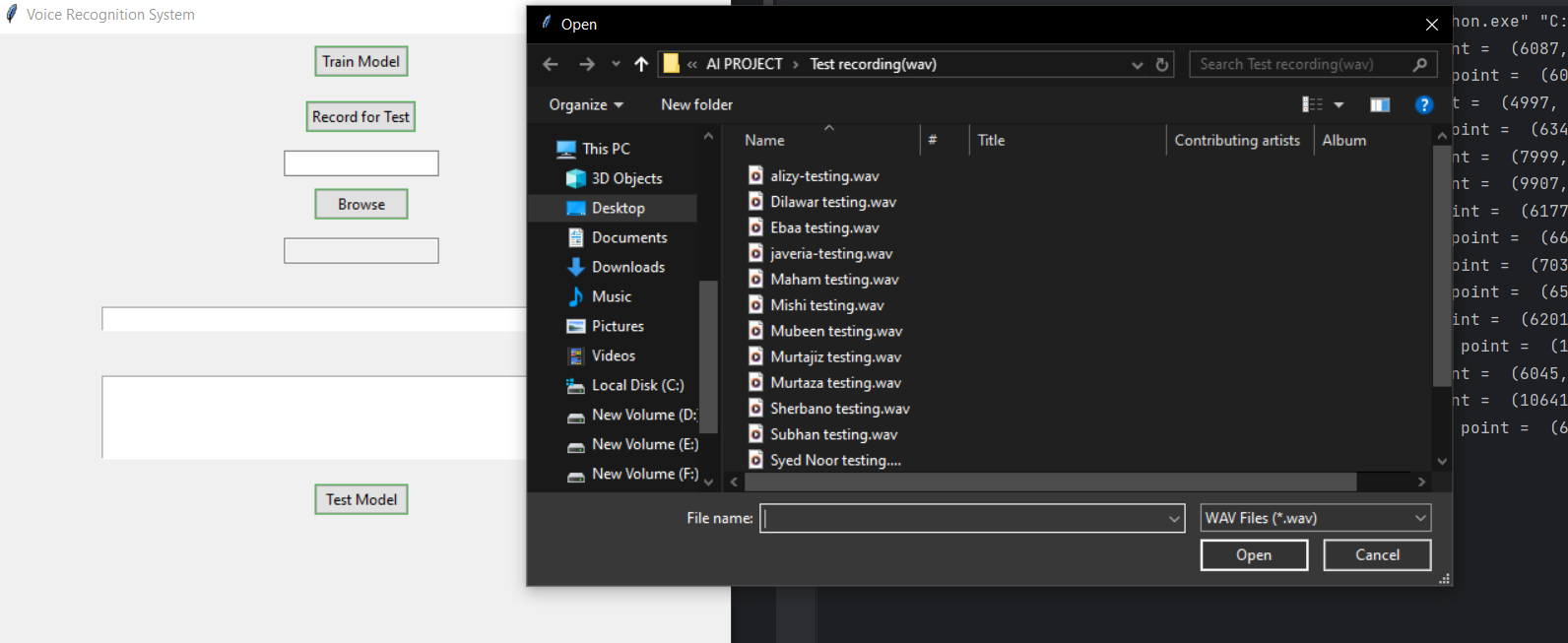
After completing the process of training models:



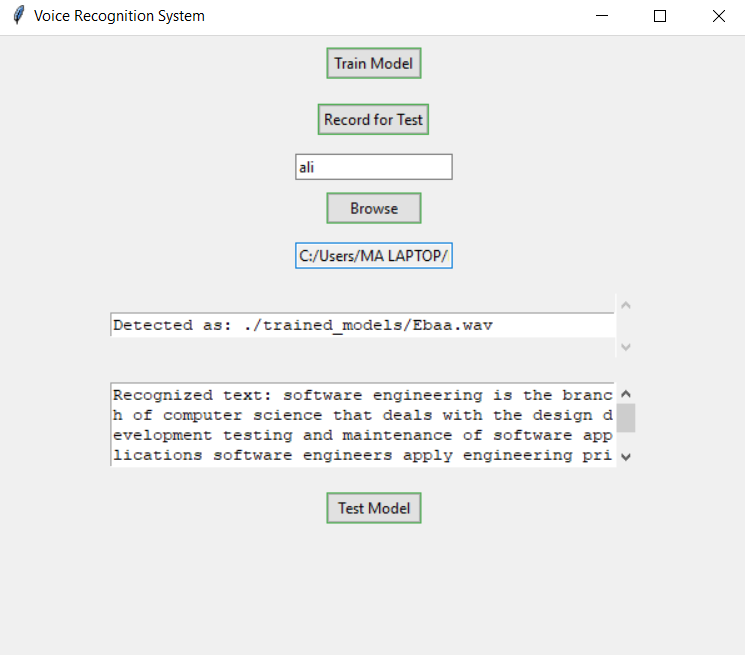
By clicking the Record for Test Button:



By Clicking the Browse Button:



By clicking the Test Model Button:



1. **Applications of the Voice Recognition System**

The Voice Recognition System developed in this project offers a versatile set of applications across various domains, leveraging its capabilities in speaker identification and speech-to-text conversion. The following are key applications of this system:

**10.1 Biometric Security Systems:**

The Voice Recognition System finds practical use in biometric security applications. Its ability to accurately identify and verify individuals based on their unique vocal characteristics enhances security measures. This application can be employed in secure access control systems, identity verification processes, and secure facilities where biometric authentication is crucial.

**10.2 Voice-Controlled Assistants:**

Integration with voice-controlled assistants represents another valuable application. The system's speech-to-text conversion capability enables seamless communication with virtual assistants. Users can dictate commands, compose text, or interact with devices using voice commands, enhancing the accessibility and convenience of voice-controlled interfaces.

**10.3 Call Center Automation:**

In the context of call centers and customer service, the Voice Recognition System can automate the process of transcribing customer calls. The system's ability to convert spoken words to text facilitates efficient handling and analysis of customer interactions, aiding in quality assurance, compliance monitoring, and customer relationship management.

**10.4 Accessibility Features:**

The system can contribute significantly to accessibility features for individuals with disabilities. Speech-to-text conversion allows users with hearing impairments to comprehend spoken information, and speaker identification can be utilized in personalized user interfaces tailored to specific individuals.

**10.5 Educational Tools:**

Voice recognition technology can be integrated into educational tools, providing a novel approach to language learning and pronunciation assessment. The system's ability to identify speakers can personalize learning experiences, and speech-to-text conversion can assist in transcribing lectures or facilitating language learning exercises.

**10.6 Voice-Enabled Smart Home Systems:**

The system can serve as a core component in voice-enabled smart home systems. Users can control smart devices, set reminders, and perform various tasks using voice commands. Speaker identification ensures that the system responds accurately to specific users, customizing the smart home experience.

1. **References**

* https://jonathan-hui.medium.com/speech-recognition-feature-extraction-mfcc-plp 2.5455f5a69dd9
* https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95
* https://www.nist.gov/itl/iad/mig/nist-2021-speaker-recognition-evaluation-sre21