**1. Code Documentation**

import tkinter as tk

from tkinter import ttk, filedialog

from tkinter import messagebox

import speech\_recognition as sr

import pyaudio

import wave

import numpy as np

import scipy.signal

import threading

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import pygame

import os

class Speech\_to\_Text\_Converter\_App(tk.Tk):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.title("Audio Recorder and Transcriber")

        self.geometry("800x600")

        self.filename\_var = tk.StringVar()

        self.duration\_var = tk.IntVar(value=5)

        self.sample\_rate\_var = tk.IntVar(value=44100)

        self.chunk\_size\_var = tk.IntVar(value=1024)

        self.channels\_var = tk.IntVar(value=1)

        self.transcription\_var = tk.StringVar()

        self.fig, self.ax = plt.subplots()

        self.canvas = None

        self.playback\_thread = None

        self.create\_widgets()

    def create\_widgets(self):

        main\_frame = ttk.Frame(self)

        main\_frame.pack(expand=True, fill='both')

        recording\_frame = ttk.LabelFrame(main\_frame, text="Recording Settings")

        recording\_frame.grid(row=0, column=0, padx=10, pady=10, sticky=(tk.W, tk.E))

        filename\_label = ttk.Label(recording\_frame, text="Filename:")

        filename\_label.grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

        filename\_entry = ttk.Entry(recording\_frame, textvariable=self.filename\_var, width=30)

        filename\_entry.grid(row=0, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))

        filename\_button = ttk.Button(recording\_frame, text="Browse", command=self.browse\_file)

        filename\_button.grid(row=0, column=2, padx=5, pady=5)

        duration\_label = ttk.Label(recording\_frame, text="Duration (seconds):")

        duration\_label.grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)

        duration\_scale = ttk.Scale(recording\_frame, from\_=1, to=10, variable=self.duration\_var, orient='horizontal')

        duration\_scale.grid(row=1, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))

        duration\_value\_label = ttk.Label(recording\_frame, textvariable=self.duration\_var)

        duration\_value\_label.grid(row=1, column=2, padx=5, pady=5, sticky=tk.W)

        sample\_rate\_label = ttk.Label(recording\_frame, text="Sample Rate:")

        sample\_rate\_label.grid(row=2, column=0, padx=5, pady=5, sticky=tk.W)

        sample\_rate\_scale = ttk.Scale(recording\_frame, from\_=22050, to=44100, variable=self.sample\_rate\_var, orient='horizontal')

        sample\_rate\_scale.grid(row=2, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))

        sample\_rate\_value\_label = ttk.Label(recording\_frame, textvariable=self.sample\_rate\_var)

        sample\_rate\_value\_label.grid(row=2, column=2, padx=5, pady=5, sticky=tk.W)

        chunk\_size\_label = ttk.Label(recording\_frame, text="Chunk Size:")

        chunk\_size\_label.grid(row=3, column=0, padx=5, pady=5, sticky=tk.W)

        chunk\_size\_scale = ttk.Scale(recording\_frame, from\_=512, to=4096, variable=self.chunk\_size\_var, orient='horizontal')

        chunk\_size\_scale.grid(row=3, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))

        chunk\_size\_value\_label = ttk.Label(recording\_frame, textvariable=self.chunk\_size\_var)

        chunk\_size\_value\_label.grid(row=3, column=2, padx=5, pady=5, sticky=tk.W)

        channels\_label = ttk.Label(recording\_frame, text="Channels (1 for mono, 2 for stereo):")

        channels\_label.grid(row=4, column=0, padx=5, pady=5, sticky=tk.W)

        channels\_scale = ttk.Scale(recording\_frame, from\_=1, to=2, variable=self.channels\_var, orient='horizontal')

        channels\_scale.grid(row=4, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))

        channels\_value\_label = ttk.Label(recording\_frame, textvariable=self.channels\_var)

        channels\_value\_label.grid(row=4, column=2, padx=5, pady=5, sticky=tk.W)

        record\_button = ttk.Button(recording\_frame, text="Start Recording", command=self.start\_recording)

        record\_button.grid(row=5, column=0, columnspan=3, pady=10)

        playback\_frame = ttk.LabelFrame(main\_frame, text="Playback")

        playback\_frame.grid(row=1, column=0, padx=10, pady=10, sticky=(tk.W, tk.E))

        playback\_button = ttk.Button(playback\_frame, text="Play Recording", command=self.playback\_recording)

        playback\_button.pack(pady=10)

        stop\_button = ttk.Button(playback\_frame, text="Stop Playback", command=self.stop\_playback)

        stop\_button.pack(pady=10)

        upload\_button = ttk.Button(playback\_frame, text="Upload Audio", command=self.upload\_audio)

        upload\_button.pack(pady=10)

        transcription\_frame = ttk.LabelFrame(main\_frame, text="Transcription")

        transcription\_frame.grid(row=0, column=1, rowspan=2, padx=10, pady=10, sticky=(tk.W, tk.E))

        result\_label = ttk.Label(transcription\_frame, text="Transcription:")

        result\_label.grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

        transcription\_text = tk.Text(transcription\_frame, height=10, width=50, wrap=tk.WORD)

        transcription\_text.grid(row=1, column=0, padx=5, pady=5, sticky=(tk.W, tk.E))

        scrollbar = ttk.Scrollbar(transcription\_frame, orient="vertical", command=transcription\_text.yview)

        scrollbar.grid(row=1, column=1, sticky=(tk.NS, tk.E))

        transcription\_text.config(yscrollcommand=scrollbar.set)

        self.transcription\_text = transcription\_text

    def browse\_file(self):

        filename = filedialog.askopenfilename(filetypes=[("Wave files", "\*.wav")])

        if filename:

            self.filename\_var.set(filename)

    def start\_recording(self):

        filename = self.filename\_var.get()

        duration = self.duration\_var.get()

        sample\_rate = self.sample\_rate\_var.get()

        chunk\_size = self.chunk\_size\_var.get()

        channels = self.channels\_var.get()

        record\_thread = threading.Thread(target=self.record\_and\_transcribe, args=(filename, duration, sample\_rate, chunk\_size, channels))

        record\_thread.start()

    def record\_and\_transcribe(self, filename, duration, sample\_rate, chunk\_size, channels):

        record\_audio(filename, duration, sample\_rate, chunk\_size, channels)

        processed\_filename = "processed\_" + filename

        preprocess\_audio(filename, processed\_filename, sample\_rate)

        transcription = analyze\_audio(processed\_filename)

        self.transcription\_var.set(transcription)

        self.transcription\_text.delete(1.0, tk.END)

        self.transcription\_text.insert(tk.END, transcription)

        # Plot the waveform

        self.plot\_waveform(processed\_filename)

    def plot\_waveform(self, filename):

        with wave.open(filename, 'rb') as wf:

            audio\_data = wf.readframes(-1)

            audio\_np = np.frombuffer(audio\_data, dtype=np.int16)

            duration = wf.getnframes() / wf.getframerate()

        self.ax.clear()

        self.ax.plot(np.linspace(0, duration, len(audio\_np)), audio\_np)

        self.ax.set\_title('Waveform')

        self.ax.set\_xlabel('Time (s)')

        self.ax.set\_ylabel('Amplitude')

        self.fig.tight\_layout()

        if self.canvas:

            self.canvas.get\_tk\_widget().destroy()

        self.canvas = FigureCanvasTkAgg(self.fig, master=self)

        self.canvas.draw()

        self.canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

    def playback\_recording(self):

        filename = self.filename\_var.get()

        if os.path.exists(filename):

            pygame.mixer.init()

            pygame.mixer.music.load(filename)

            pygame.mixer.music.play()

        else:

            messagebox.showerror("File Not Found", "The selected audio file does not exist.")

    def stop\_playback(self):

        pygame.mixer.music.stop()

    def upload\_audio(self):

        filename = filedialog.askopenfilename(filetypes=[("Wave files", "\*.wav")])

        if filename:

            self.filename\_var.set(filename)

            # Automatically start playing the uploaded audio

            self.playback\_recording()

def record\_audio(filename, duration=5, sample\_rate=44100, chunk\_size=1024, channels=1, format=pyaudio.paInt16):

    audio = pyaudio.PyAudio()

    stream = audio.open(format=format, channels=channels,

                        rate=sample\_rate, input=True,

                        frames\_per\_buffer=chunk\_size)

    print("Recording...")

    frames = []

    for i in range(0, int(sample\_rate / chunk\_size \* duration)):

        data = stream.read(chunk\_size)

        frames.append(data)

    print("Finished recording.")

    stream.stop\_stream()

    stream.close()

    audio.terminate()

    filepath = os.path.join(os.getcwd(), filename)

    with wave.open(filepath, 'wb') as wf:

        wf.setnchannels(channels)

        wf.setsampwidth(audio.get\_sample\_size(format))

        wf.setframerate(sample\_rate)

        wf.writeframes(b''.join(frames))

def preprocess\_audio(input\_filename, output\_filename, sample\_rate=44100):

    try:

        with wave.open(input\_filename, 'rb') as wf:

            n\_channels = wf.getnchannels()

            sampwidth = wf.getsampwidth()

            n\_frames = wf.getnframes()

            audio\_data = wf.readframes(n\_frames)

        audio\_np = np.frombuffer(audio\_data, dtype=np.int16)

        # Perform noise reduction using a simple high-pass filter

        b, a = scipy.signal.butter(1, 1000 / (0.5 \* sample\_rate), btype='high', analog=False)

        filtered\_audio = scipy.signal.filtfilt(b, a, audio\_np)

        filtered\_audio = np.int16(filtered\_audio)

        with wave.open(output\_filename, 'wb') as wf:

            wf.setnchannels(n\_channels)

            wf.setsampwidth(sampwidth)

            wf.setframerate(sample\_rate)

            wf.writeframes(filtered\_audio.tobytes())

            print(f"Audio processing completed: {output\_filename}")

    except Exception as e:

        print(f"Error in audio processing: {e}")

def analyze\_audio(filename):

    recognizer = sr.Recognizer()

    with sr.AudioFile(filename) as source:

        audio\_data = recognizer.record(source)

        try:

            text = recognizer.recognize\_google(audio\_data)

            print("Transcription:", text)

            return text

        except sr.UnknownValueError:

            print("Could not understand audio")

            return "Could not understand audio"

        except sr.RequestError as e:

            print("Error occurred:", e)

            return f"Error occurred: {e}"

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

    app = Speech\_to\_Text\_Converter\_App()

    app.mainloop()

**1.1. Code Overview**

The Speech\_to\_Text\_Converter\_App is a tkinter-based GUI application that allows users to record audio, preprocess it, transcribe it into text, play it back, and visualize the waveform. It uses several libraries to handle different functionalities, including pyaudio for recording, speech\_recognition for transcription, scipy and numpy for audio processing, and pygame for playback.

**1.1.1. Class and Methods**

**1. Class Speech\_to\_Text\_Converter\_App**

This class inherits from tk.Tk and represents the main application window.

**\_\_init\_\_(self)**

* Initializes the main window with a title and geometry.
* Sets up instance variables using tkinter variables for filename, duration, sample rate, chunk size, channels, and transcription text.
* Initializes a matplotlib figure for plotting.
* Calls the create\_widgets() method to set up the GUI components.

**create\_widgets(self)**

* Creates the main frame and sub-frames for different functionalities (Recording Settings, Playback, Transcription).
* Adds labels, entry fields, scales, and buttons for setting recording parameters.
* Sets up a text widget for displaying the transcription with a scrollbar.

**browse\_file(self)**

* Opens a file dialog to select a .wav file and updates the filename variable with the selected file path.

**start\_recording(self)**

* Retrieves user settings and starts a new thread to handle the recording and transcription process by calling record\_and\_transcribe().

**record\_and\_transcribe(self, filename, duration, sample\_rate, chunk\_size, channels)**

* Calls record\_audio() to record audio based on user settings.
* Calls preprocess\_audio() to preprocess the recorded audio.
* Calls analyze\_audio() to transcribe the audio and updates the transcription text widget.
* Calls plot\_waveform() to plot the waveform of the processed audio.

**plot\_waveform(self, filename)**

* Reads the processed audio file and extracts audio data.
* Plots the waveform using matplotlib and displays it in the Tkinter window.

**playback\_recording(self)**

* Uses pygame to play the selected audio file.
* Displays an error message if the file doesn't exist.

**stop\_playback(self)**

* Stops audio playback using pygame.

**upload\_audio(self)**

* Opens a file dialog to select a .wav file and updates the filename variable.
* Automatically starts playback of the uploaded audio.

**Helper Functions**

**record\_audio(filename, duration=5, sample\_rate=44100, chunk\_size=1024, channels=1, format=pyaudio.paInt16)**

* **Description**: Records audio from the microphone and saves it to a .wav file.
* **Parameters**:
  + filename: The name of the file to save the recording.
  + duration: Duration of the recording in seconds.
  + sample\_rate: Sampling rate of the recording.
  + chunk\_size: Number of frames per buffer.
  + channels: Number of audio channels (1 for mono, 2 for stereo).
  + format: Audio format (default is 16-bit PCM).
* **Process**:
  + Initializes pyaudio and opens a stream for recording.
  + Reads audio data in chunks and stores it in a list.
  + Writes the recorded audio data to a .wav file.

**preprocess\_audio(input\_filename, output\_filename, sample\_rate=44100)**

* **Description**: Preprocesses the recorded audio to reduce noise using a high-pass filter.
* **Parameters**:
  + input\_filename: The name of the input audio file.
  + output\_filename: The name of the output processed audio file.
  + sample\_rate: Sampling rate of the audio.
* **Process**:
  + Reads the input audio file and converts the audio data to a NumPy array.
  + Applies a high-pass filter using scipy.signal to reduce noise.
  + Writes the filtered audio data to a new .wav file.

**analyze\_audio(filename)**

* **Description**: Transcribes the audio file to text using Google's speech recognition API.
* **Parameters**:
  + filename: The name of the audio file to be transcribed.
* **Process**:
  + Uses speech\_recognition to read the audio file.
  + Transcribes the audio using Google's speech recognition and returns the transcription text.
  + Handles errors if the transcription fails.

**1.1.2. GUI Components**

* **Main Frame (main\_frame)**: Container for all other frames.
* **Recording Frame (recording\_frame)**: Contains controls for recording settings:
  + Filename entry and browse button.
  + Duration scale and label.
  + Sample rate scale and label.
  + Chunk size scale and label.
  + Channels scale and label.
  + Start recording button.
* **Playback Frame (playback\_frame)**: Contains playback controls:
  + Play recording button.
  + Stop playback button.
  + Upload audio button.
* **Transcription Frame (transcription\_frame)**: Displays the transcription text:
  + Text widget with vertical scrollbar.

**1.1.3. Workflow**

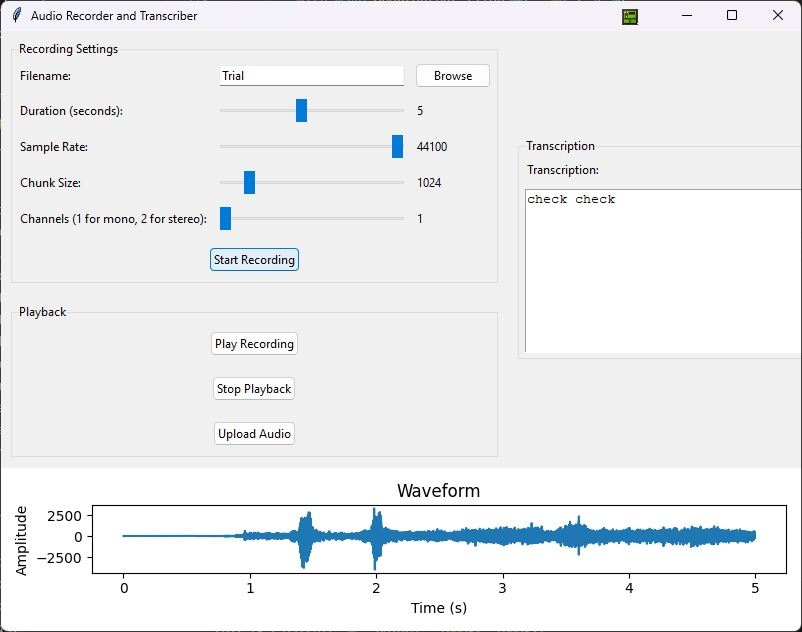
1. **Recording**:
   * User sets the desired recording settings and clicks "Start Recording".
   * The application records audio, processes it to reduce noise, transcribes it, and displays the transcription.
   * The waveform of the processed audio is plotted and displayed.
2. **Playback**:
   * User can play the recorded or uploaded audio using the playback buttons.
3. **Upload Audio**:
   * User can upload an existing .wav file, which is then automatically played.

**1.1.4. Libraries Used**

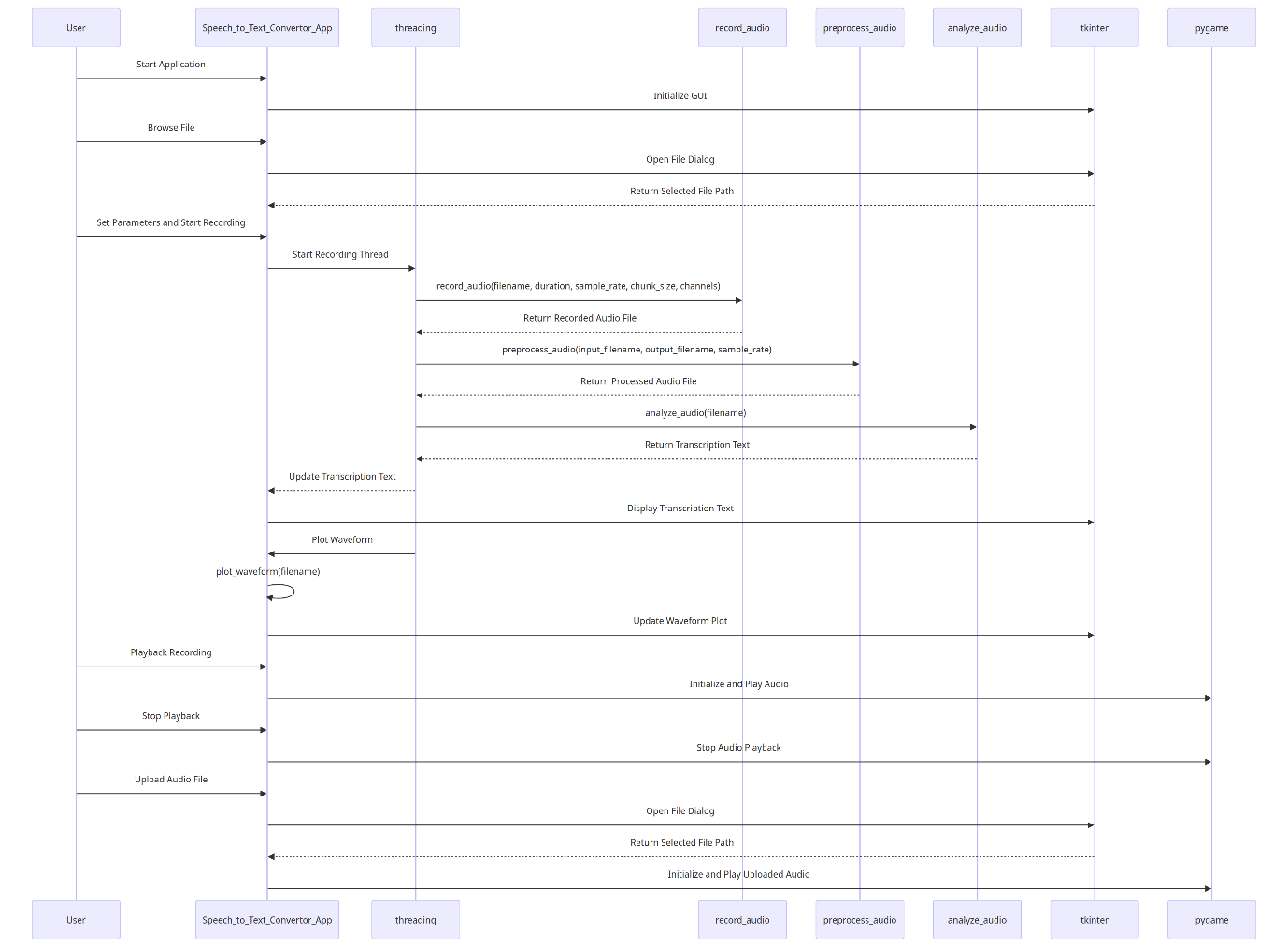
* **tkinter**: For creating the GUI.
* **speech\_recognition**: For transcribing audio.
* **pyaudio**: For recording audio.
* **wave**: For reading and writing .wav files.
* **numpy**: For numerical operations on audio data.
* **scipy.signal**: For applying filters to audio data.
* **threading**: For handling recording and transcription in a separate thread.
* **matplotlib**: For plotting the waveform.
* **pygame**: For audio playback.
* **os**: For file path operations.

This detailed documentation provides a comprehensive understanding of the workings of the Speech\_to\_Text\_Converter\_App code, covering its structure, methods, helper functions, GUI components, workflow, and the libraries used.

**1.2. Output**

****

**1.2. Sequence Diagram**

****