Here's an explanation of the code line by line:

python

Copy code

import streamlit as st

* This imports the streamlit library and aliases it as st. Streamlit is a Python library for creating interactive web apps quickly. It’s used here for building the user interface.

python

Copy code

import openai

* This imports the openai library, which allows interaction with OpenAI's API, enabling use of models like GPT-3 and Whisper (for transcription).

python

Copy code

import base64

* This imports the base64 module, which provides functions for encoding and decoding data in Base64 format. It is used to embed audio in HTML in a format that the browser can play.

python

Copy code

from st\_audiorec import st\_audiorec

* This imports st\_audiorec from the st\_audiorec package. st\_audiorec is a component for recording audio directly within a Streamlit app.

python

Copy code

import os

* This imports the os module, which allows interaction with the operating system (e.g., file operations such as checking for the existence of files, removing files, etc.).

**Function Definitions**

python

Copy code

def save\_input\_recording(recorded\_audio, file\_path='audio.mp3'):

with open(file\_path, 'wb') as file:

file.write(recorded\_audio)

* This function save\_input\_recording saves the recorded audio (binary data) to a file (audio.mp3 by default). It takes recorded\_audio as input and writes it to the specified file\_path.

python

Copy code

def delete\_audio\_if\_exists(file\_path):

if os.path.exists(file\_path):

os.remove(file\_path)

* This function checks if a file exists at the given file\_path and, if it does, deletes it. This is used to clean up old audio files before saving new ones.

python

Copy code

def transcribe\_audio(client, audio\_path):

with open(audio\_path, 'rb') as audio\_file:

transcript = client.audio.transcriptions.create(

model='whisper-1', file=audio\_file, language='en'

)

return transcript.text

* This function transcribes an audio file using OpenAI's whisper-1 model. It opens the audio file at the specified audio\_path, sends it to OpenAI for transcription, and returns the transcribed text.

python

Copy code

def get\_llm\_response(client, input\_text):

messages = [

{"role":"system", "content":"You are a helpful assistant"},

{"role":"user", "content":input\_text}

]

response = client.chat.completions.create(

model='gpt-3.5-turbo',

messages=messages

)

return response.choices[0].message.content

* This function generates a response from OpenAI’s gpt-3.5-turbo model. It sends a conversation history (system message and user input) and retrieves the assistant's response. The function returns the content of the AI's reply.

python

Copy code

def convert\_text\_to\_audio(client, text, audio\_path):

with client.audio.speech.with\_streaming\_response.create(

model='tts-1', voice='nova', input=text

) as response:

response.stream\_to\_file(audio\_path)

* This function converts a given text (text) to speech using OpenAI's tts-1 model and saves the resulting audio to a file at audio\_path. It streams the audio to the file as it’s being generated.

python

Copy code

def autoplay\_audio(audio\_file):

with open(audio\_file, 'rb') as file:

audio\_bytes = file.read()

base64\_audio = base64.b64encode(audio\_bytes).decode("utf-8")

audio = f'<audio src="data:audio/mp3;base64,{base64\_audio}" controls autoplay>'

st.markdown(audio, unsafe\_allow\_html=True)

* This function plays an audio file in the Streamlit app. It reads the audio file, encodes it in Base64, and then embeds it as an HTML <audio> tag that can autoplay in the app. The unsafe\_allow\_html=True flag allows raw HTML to be rendered in the Streamlit app.

python

Copy code

def create\_card(title, text):

# Define CSS for the card

card\_css = """

<style>

.card {

border-style: solid;

border-radius: 15px;

box-shadow: 0 10px 10px rgba(0, 0, 0, 0.2);

padding: 20px;

margin: 20px 0;

background-color: white;

}

.card-title {

font-size: 24px;

font-weight: bold;

}

.card-text {

font-size: 16px;

color: #555;

}

</style>

"""

# Create the card HTML

card\_html = f"""

<div class="card">

<div class="card-title">{title}</div>

<div class="card-text">{text}</div>

</div>

"""

# Render the card with CSS

st.markdown(card\_css ,unsafe\_allow\_html=True)

st.markdown(card\_html,unsafe\_allow\_html=True)

* This function creates a styled "card" in the Streamlit interface. It defines some custom CSS for the card, then builds the HTML structure with the provided title and text. It renders the CSS and HTML in the Streamlit app.

**Main Application Function**

python

Copy code

def application():

st.set\_page\_config(page\_title="Voice LLM - Manish Singh", page\_icon='🎤')

st.title("Voice LLM - Manish Singh 🤖")

st.subheader("Please enter OpenAI api key in sidebar first")

* The application function sets up the Streamlit page configuration, such as the title and icon. It also displays the main title and a message prompting the user to enter their OpenAI API key in the sidebar.

python

Copy code

# Sidebar

with st.sidebar:

api\_key = st.text\_input("OpenAI API Key", type="password")

if api\_key:

st.success("Done")

* In the sidebar, this block creates a password input field for the user to enter their OpenAI API key. If the key is provided, a success message is displayed.

python

Copy code

# If api key is written

if api\_key:

client = openai.OpenAI(api\_key=api\_key)

st.write("Click Start Recording to Record and Stop once Done🎙️")

recorded\_audio = st\_audiorec()

* If the API key is entered, it initializes the OpenAI client using the provided API key. The user is instructed to click to start recording, and the st\_audiorec() function is called to record audio.

python

Copy code

if recorded\_audio:

with st.spinner("Processing"):

delete\_audio\_if\_exists("audio.mp3")

delete\_audio\_if\_exists("audio\_response.mp3")

save\_input\_recording(recorded\_audio)

transcribed\_text = transcribe\_audio(client, 'audio.mp3')

create\_card("User Input", transcribed\_text)

ai\_response = get\_llm\_response(client, transcribed\_text)

audio\_response\_file = "audio\_response.mp3"

convert\_text\_to\_audio(client, ai\_response, audio\_response\_file)

autoplay\_audio(audio\_response\_file)

create\_card("AI Response", ai\_response)

* If audio is recorded, it processes the audio. First, it cleans up any existing audio files, then saves the new recording. The recording is transcribed to text using OpenAI's Whisper model. A card with the transcribed text is displayed. Then, it generates a response from the LLM (GPT-3.5) and converts it into speech. The resulting audio is played back to the user, and another card with the AI response is shown.

**Running the Application**

python

Copy code

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

application()

* This checks if the script is being run directly (not imported as a module). If so, it calls the application() function to start the app.

**Summary**

This code creates a voice-based interface for interacting with OpenAI’s models using Streamlit. Users can record audio, which is transcribed to text. The transcribed text is sent to GPT-3.5 to generate a response, which is then converted to speech and played back to the user. The app also includes a sidebar for entering the OpenAI API key, and dynamically displays both the user's input and the AI's response in stylized cards.

Top of Form

Bottom of Form