

**VOICE CHATBOT CREATED USING OPENAI**

By Khushal Singh(MST03-0078)

Under guidance of Urooj khan

Contents

|  |  |  |
| --- | --- | --- |
| Sr.No. | Title | Page Number |
| 1 | Abstract | 3 |
| 2 | Introduction | 4 |
| 3 | Technology used | 5 |
| 4 | Methodology | 6-10 |
| 5 | Code snippet | 11-13 |
| 6 | Results and Discussion | 14-16 |
| 7 | Conclusion | 17 |
| 8 | References | 18 |

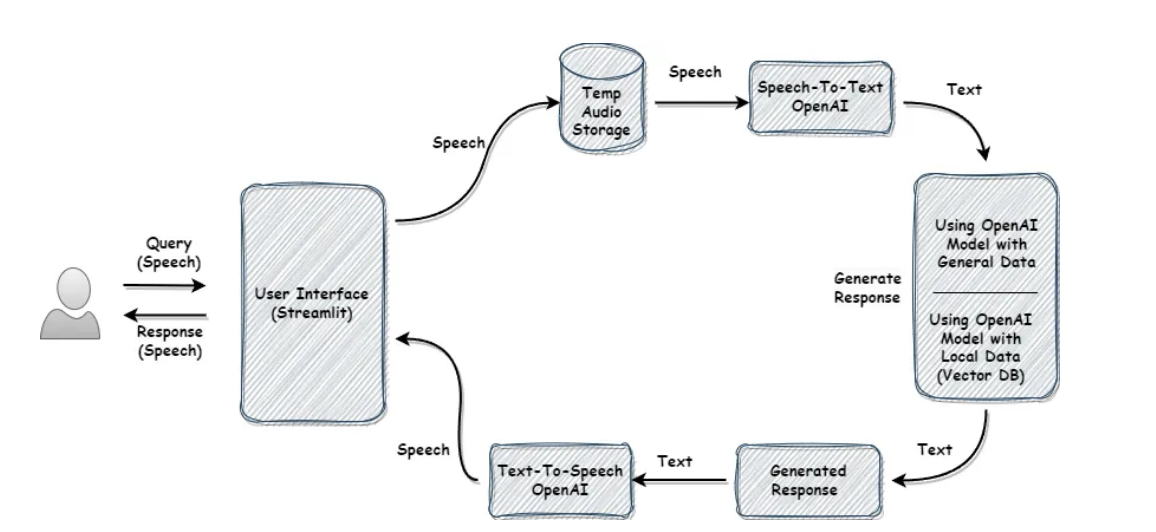
**ABSTRACT**

This project presents the development of a sophisticated conversational voice chatbot using Open AI and Streamlit. The chatbot integrates cutting-edge voice recognition and natural language processing technologies to offer a seamless and interactive user experience. By leveraging Google's Open AI for generative responses and Streamlit for web application development, the chatbot provides a dynamic platform for engaging in voice-based conversations. The project employs several key technologies and methodologies to achieve its objectives. Environment variables are managed securely using a .env file, and custom CSS is applied to ensure a visually appealing and user-friendly interface. Speech-to-text conversion is performed using the SpeechRecognition library, while text-to-speech responses are generated with pre defined open AI structure. These components work in conjunction to create an immersive interaction where users can speak to the chatbot, receive contextually relevant answers, and listen to spoken responses. A crucial feature of the chatbot is its ability to maintain conversation history, which ensures coherent and contextually aware interactions. The integration of a microphone interface allows users to provide voice input directly, enhancing the accessibility and convenience of the chatbot. The project demonstrates how combining advanced AI technologies with real-time voice interaction can create a powerful and engaging conversational agent. Overall, the Open AI Voice Chatbot exemplifies the potential of integrating voice recognition and generative AI to deliver an innovative and user-centric application. It showcases the ability to create intuitive conversational experiences that bridge the gap between human communication and machine intelligence

**INTRODUCTION**

This project represents an innovative approach to enhancing human-computer interaction through voice-based communication. In an era where conversational AI is becoming increasingly integral to various applications, from customer service to personal assistance, this project seeks to push the boundaries of what is possible by integrating advanced NLP capabilities with an intuitive user interface. At its core, the project leverages Googles Gemini API, a state-of-the-art language model known for its robust ability to comprehend and generate human-like responses. The Gemini API is designed to understand context, nuance, and intent, making it ideal for developing a chatbot that can engage in meaningful and contextually relevant conversations. By utilizing this powerful API, the project aims to create a chatbot that is not only responsive but also capable of providing accurate and insightful answers to user queries. Streamlit, a modern web application framework, is employed in this project to create an interactive and visually appealing user interface. Streamlit's simplicity and flexibility allow for the rapid development of a web-based platform where users can interact with the chatbot through voice commands. The integration of voice functionality enables a more natural and convenient user experience, as users can speak directly to the chatbot instead of relying on traditional text input methods. The development process involves several key components, including speech recognition for capturing and converting spoken language into text, text-to-speech synthesis for generating audible responses, and a conversational engine powered by the Gemini API. These elements work together to create a fluid and dynamic conversation flow, where the chatbot can understand user intent, process information, and provide relevant feedback in real-time. This project also addresses the challenge of maintaining conversation history, ensuring that the chatbot can remember and refer back to previous interactions, thereby offering a more coherent and personalized user experience. Additionally, the design considers scalability, allowing the chatbot to be adapted and expanded for various applications, such as virtual assistants, customer support systems, or educational tools.

**METHODOLOGY**



1. Importing necessary libraries

The first step is to import the necessary libraries required for the implementation of this project. Some of the major libraries are:

Streamlit: Used to build and deploy the web application, providing the user interface for the chatbot.

Dotenv: Used for loading environment variables from a ‘.env' file, ensuring secure

management of API keys and other sensitive information.

Open AI: Used to interact with OPENAI API, enabling the

chatbot to generate and understand human-like responses.

SpeechRecognition: Used for converting speech input from the user into text,

facilitating voice interaction with the chatbot.

Audio Recorder Streamlit: A custom module used to capture audio input from the

user's microphone within the Streamlit application.

OS: Used for interacting with the operating system, primarily to manage file paths

and environment variables.

2. Load the Environment Variables

Environment variables are crucial for securely managing sensitive information, such as API keys, without exposing them directly in the code. In this project, a .env file stores these variables, which are loaded into the application at runtime. This method not only enhances security but also allows for easy configuration changes without modifying the code. For example, if you need to update your API key, you can do so in the .env file without altering the application code itself.

3. Configure Streamlit Page Settings

Streamlit provides the ability to customize the appearance and behavior of your web application. By setting the page title, layout, and icon, you ensure that the application looks professional and is user-friendly. The page title appears in the browser tab, while the layout option controls how the content is arranged on the screen. The page icon adds a visual cue that enhances the branding of the application. These settings contribute to a polished and consistent user experience.

4. Apply Custom CSS for Styling the Page

Custom CSS is used to fine-tune the visual design of the chatbot interface, making it more appealing and user-friendly. By embedding HTML and CSS directly into the Streamlit app, you can control the background color, text color, and specific styles for user and assistant messages. This customization is essential for creating a visually distinct interface that improves readability and user engagement. For example, using different colors for user and assistant messages helps users easily distinguish between their inputs and the chatbot's responses.

5. Create a Function to Convert Speech to Text Using Speech Recognizer  
Speech recognition is a key component of the voice chatbot. The speech-to-text function uses a speech recognition library to convert the user's spoken input into text. This text is then processed by the chatbot to generate a response. The function includes error handling to manage situations where the speech is unclear or the service is unavailable, ensuring that the chatbot can gracefully handle various scenarios. This capability allows users to interact with the chatbot naturally through voice commands.  
6. Set Up OpenAI API Key from Streamlit Secrets Using Microsoft’s OpenAI Model To access Open AI services, you need to authenticate using an API key. This key is securely stored in Streamlit's secrets management system, which protects it from being exposed in the code. The project uses the OpenAI model, which requires the API key for authorization. By configuring the AI model with this key, the chatbot can access advanced language processing capabilities, enabling it to generate human-like responses based on user input.  
7. Initialize Session State for Conversation History  
Streamlit's session state allows the application to maintain data across multiple interactions. In this project, the conversation history is stored in the session state, ensuring that the dialogue between the user and the chatbot is preserved as long as the session is active. This feature is crucial for creating a continuous and coherent conversation, as it allows the chatbot to reference previous exchanges, making the interaction feel more natural and engaging.  
8. Add the Welcome Message to Conversation History

The welcome message serves as an initial greeting when the user first interacts with the chatbot. This message is added to the conversation history only if it hasn't been added already, ensuring that new users are greeted properly while avoiding redundancy for returning users. The welcome message sets a friendly tone and helps users understand that they are interacting with an AI-driven chatbot. It also provides an opportunity to introduce the chatbot's capabilities or provide instructions.

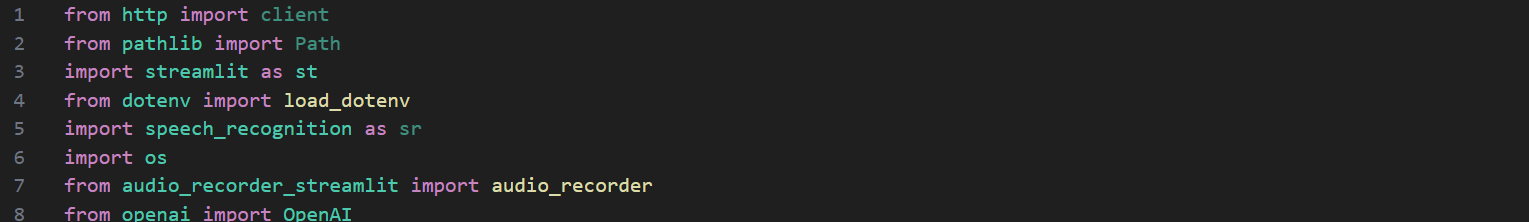
9 Display Conversation History  
Displaying the conversation history is important for maintaining context during the interaction. Each exchange between the user and the chatbot is visually represented on the screen, with different styles applied to user and assistant messages. This visual distinction helps users follow the conversation and keeps the interface organized. By maintaining and displaying the conversation history, users can refer back to previous responses, making the interaction smoother and more intuitive.  
10. Initialize the Microphone and Setting Up the Microphone

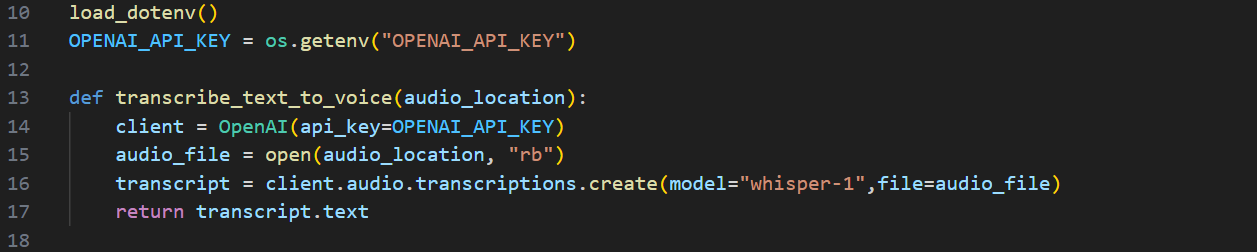
The microphone serves as the primary input device for capturing the user's voice commands. In the Streamlit app, a sidebar is created to allow users to easily access and control the microphone. This setup is critical for enabling voice interaction, as it allows the chatbot to receive audio input from the user. The sidebar provides a convenient and accessible interface for starting or stopping the microphone, ensuring that users can easily interact with the chatbot using voice commands.

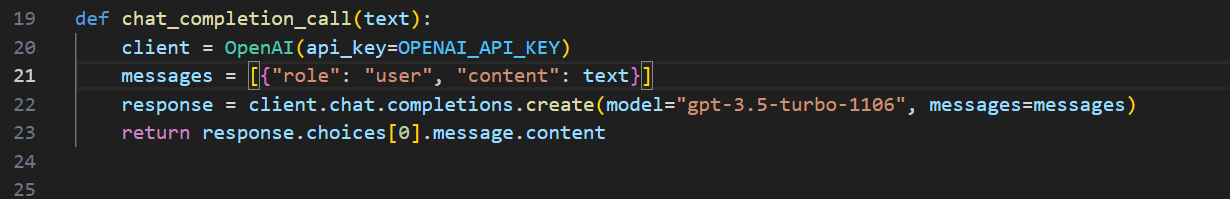
11. Process the Audio Data  
This step is the core of the chatbot's functionality, where the captured audio data is processed and responded to. The workflow is as follows:  
Audio Data Processing: Once the user speaks, the audio is captured and temporarily stored. This allows for further processing without the need to continually record audio in real-time, which can be resource-intensive.  
Speech-to-Text Conversion: The stored audio is converted into text using the speech recognition function. This text represents the user's query or command, which is then used as input for the chatbot. The accuracy of this conversion is crucial for the chatbot to understand and respond appropriately to the user's intent. Display User Prompt: The text generated from the user's speech is added to the conversation history and displayed on the screen. This step ensures that the user can see their input and verify that it was correctly understood by the system.  
Send to Open AI: The user's text input, along with the existing conversation history, is sent to the Open AI model. This model processes the input and generates a relevant response, leveraging advanced natural language understanding to produce a coherent and contextually appropriate answer.  
Display and Voice Response: The AI model's response is added to the conversation history, ensuring that the dialogue is maintained. This creates a seamless interaction, where the chatbot not only responds with text but also provides auditory feedback, making the experience more immersive and accessible.

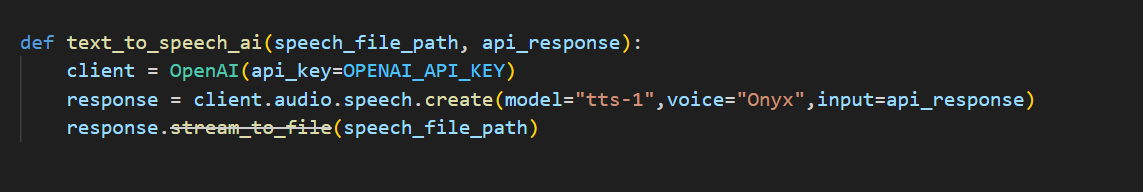
This process integrates various components of the chatbot to create a smooth and interactive user experience. By handling audio input, converting it to text, processing it through an AI model, and providing both visual and auditory responses, the chatbot offers a dynamic and engaging interface for users.

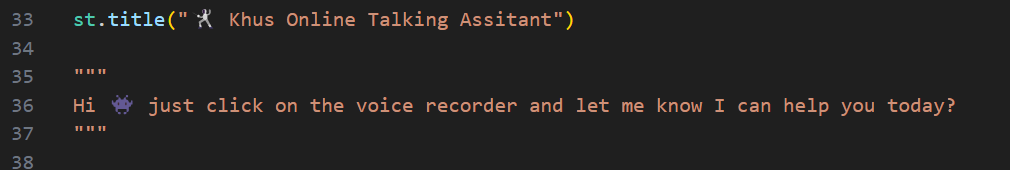
**CODE SNIPPET**

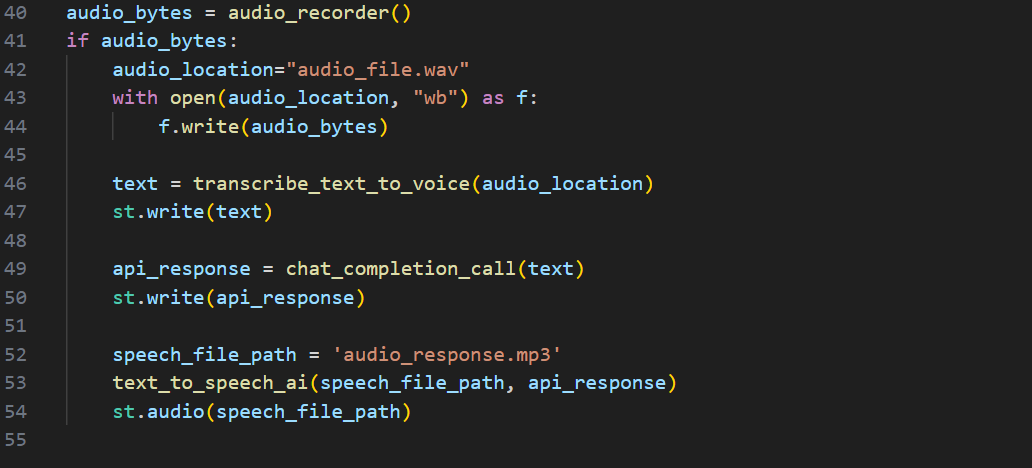
****

****

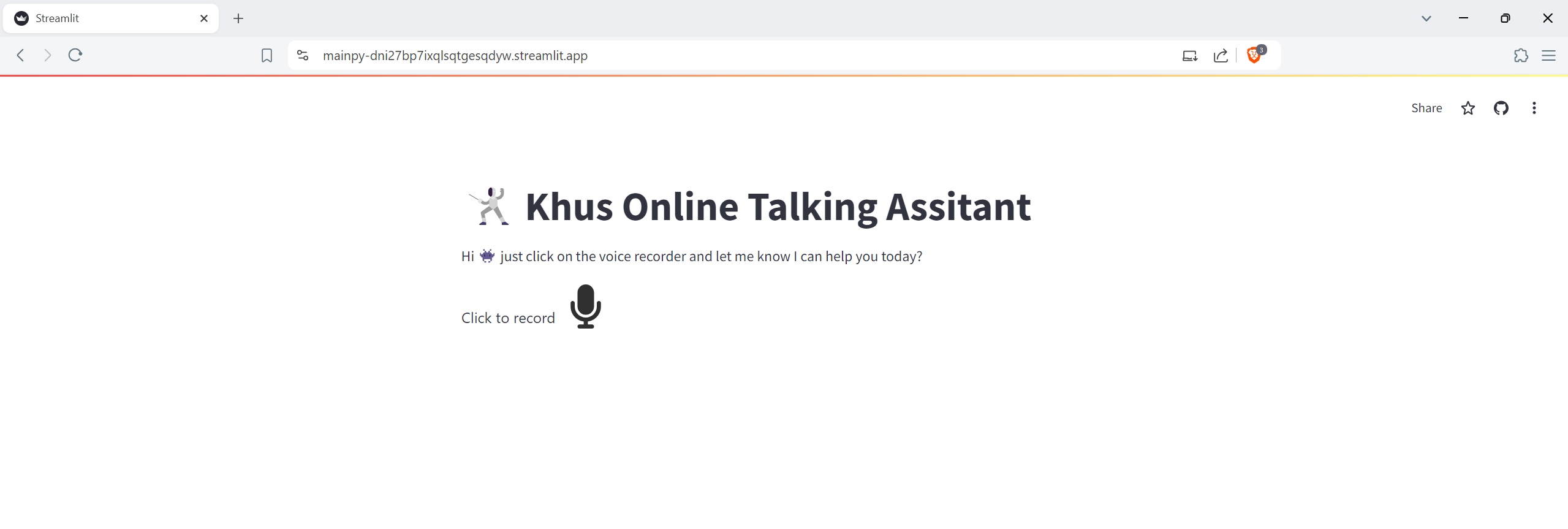
****







**Results and discussion**

****

**Conclusion**The development of the OpenAI Voice Chatbot represents a significant advancement in creating intuitive and interactive conversational agents. By leveraging advanced technologies such as Microsoft OpenAI, Streamlit for building dynamic web applications, and various audio processing libraries, this project demonstrates a sophisticated approach to integrating voice recognition and generative AI.  
Throughout this project, we have seamlessly combined multiple components to deliver a comprehensive voice chatbot experience. From setting up environment variables and configuring page settings to applying custom styles and handling user input, each step has been carefully designed to enhance user engagement and interaction. The chatbot's ability to process speech input, generate contextual responses, and provide both text and voice feedback showcases its versatility and effectiveness.  
This project not only highlights the potential of combining generative AI with real-time voice interaction but also sets the stage for future enhancements. Potential improvements could include expanding language support, refining speech recognition accuracy, and integrating additional features to further personalize user interactions.  
In conclusion, the OpenAI Voice Chatbot exemplifies the power of modern AI technologies to create innovative solutions that improve human-computer interactions. It stands as a testament to the capabilities of integrating voice recognition and AI-driven responses, paving the way for more advanced and user-friendly conversational agents in the future.

**REFRENCES**

<https://medium.com/@sudha_41820/building-a-voice-agent-with-streamlit-a-beginners-guide-f929d0e250c1>

<https://medium.com/@dhruvsawhney04/building-an-intelligent-chatbot-with-openai-and-streamlit-2fb968e5bf94> <https://blog.futuresmart.ai/building-a-conversational-voice-chatbot-integrating-openais-speech-to-text-text-to-speech>