

Smart Conversational AI: Voice-Enabled Chatbot Using Gemini Pro

By

Varnika Milind Mulay (MST03-0060)

Submitted to

Meta Scifor Technologies



**Meta
Scifor Technologies**

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 Google's Gemini 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 Gemini 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 Google Text-to-Speech (gTTS). 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 Gemini 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.

Keywords: Voice Chatbot, Gemini API, Streamlit, Speech-to-Text, Generative AI

Abbreviations:

gTTS – Google Text to Speech

UML – Unified Modelling Language

NLP – Natural Language Processing

API – Application Programming Interface

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 Google's 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.

In summary, "Developing a Conversational Chatbot Using Gemini API and Streamlit" is an ambitious project that combines cutting-edge NLP technology with a user-friendly interface to create a sophisticated and interactive voice-based chatbot. By harnessing the capabilities of the Gemini API and Streamlit, this project aims to deliver a chatbot that not only meets the current demands of conversational AI but also sets the stage for future innovations in human-computer interaction.

Technology used

1. Programming Language:

- Python: The primary programming language used for developing the chatbot, handling voice input, and interacting with the Gemini API.

2. Libraries and Frameworks:

- Streamlit: A Python-based web framework used to create the user interface for the chatbot, allowing real-time interaction through a web application.
- Google Generative AI (google.generativeai): Provides the NLP capabilities through the Gemini API, enabling the chatbot to generate and understand human-like responses.
- SpeechRecognition: Used for converting speech to text, allowing users to interact with the chatbot via voice commands.
- Google Text-to-Speech (gTTS): Converts text responses generated by the chatbot into spoken audio, enhancing user interaction with audible responses.
- Audio Recorder Streamlit (audio_recorder): A module integrated into Streamlit to capture audio input from the user's microphone in real-time.
- Dotenv: A library used to load environment variables from a .env file, ensuring secure handling of API keys and other sensitive information.
- io (BytesIO): Used for handling audio data in memory, enabling efficient processing of audio streams without needing to write them to disk.

3. Tools and Environments:

- Visual Studio Code: The development environment used for coding and testing the chatbot application.
- Operating System: The project is developed and tested in a Windows environment.

4. APIs and Services:

- Google Gemini API: Provides the conversational AI capabilities, powering the chatbots ability to understand and generate human-like responses.

5. Data Handling:

- Speech-to-Text Conversion: Custom functions that utilize the SpeechRecognition library to process and convert voice inputs into text.
- Text-to-Speech Conversion: Implemented using gTTS to convert text-based responses into audio, which is then played back to the user.

6. User Interface Design:

- Custom CSS Styling: Applied to the Streamlit app to enhance the visual appeal and user experience by styling the chatbots interface, including background colors, text colors, and message formatting.

Methodology

The methodology followed in the study is explained in this section.

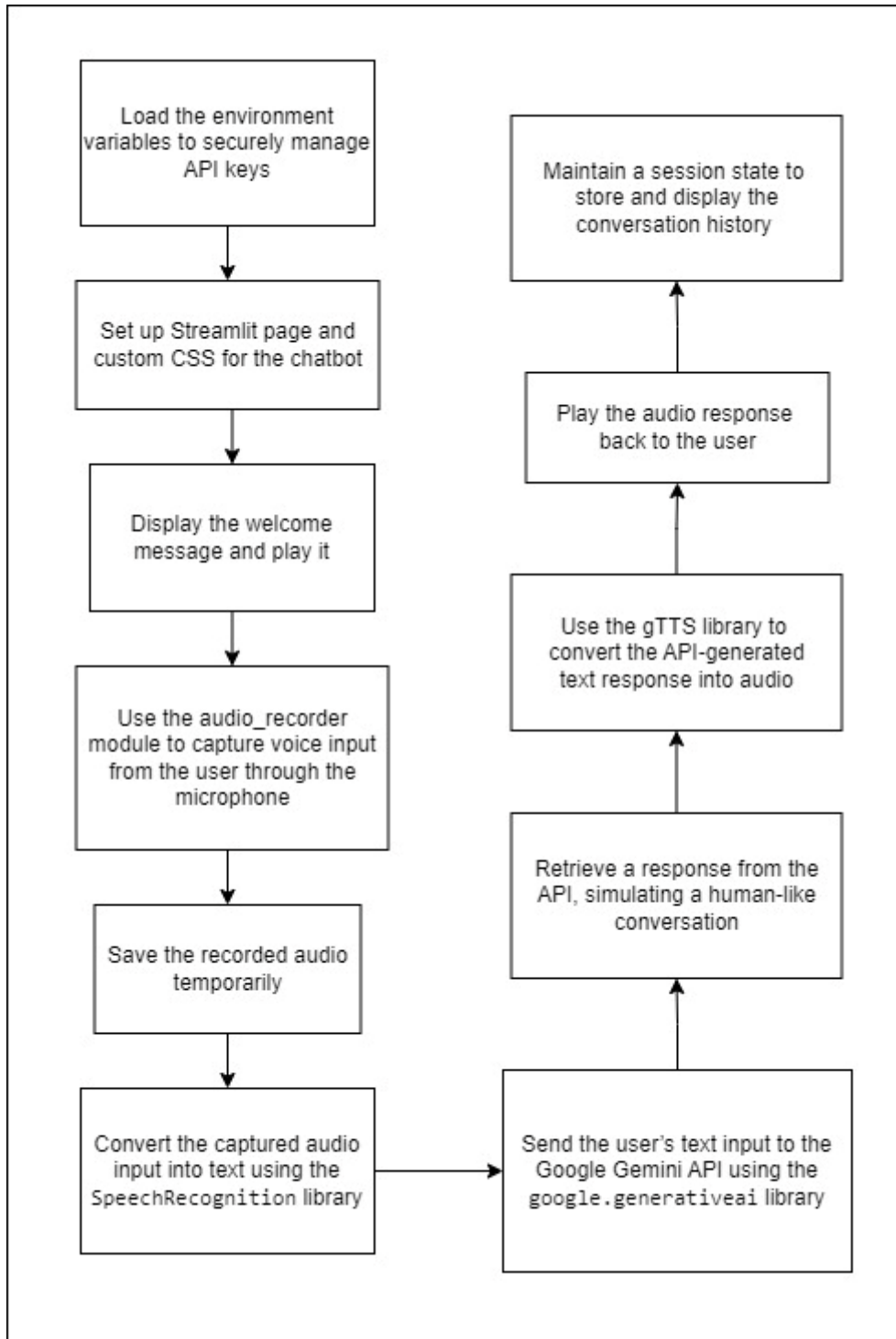


Figure 1. Workflow of the study

This research follows the steps as mentioned in **Figure 1**. The detailed working of the same is explained below:

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.
- Google Generative AI: Used to interact with the Google Gemini 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.
- Google Text-to-Speech: Used for converting text responses generated by the chatbot into spoken audio, allowing the chatbot to respond audibly to the user.
- Audio Recorder Streamlit: A custom module used to capture audio input from the user's microphone within the Streamlit application.
- IO: Specifically, the BytesIO class from the io module is used for handling in-memory audio data streams, allowing efficient audio processing without writing to disk.
- 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 Google API Key from Streamlit Secrets Using Google Generative AI Model

To access Google's 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 Google Gemini AI 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. Generate and Play the Welcome Message Using gTTS and BytesIO

To enhance the user experience, the welcome message is converted to speech using the Google Text-to-Speech (gTTS) service. The audio is then played back to the user using Streamlit's audio function. By using BytesIO, the audio data is handled in memory, allowing for efficient processing without the need to write temporary files to disk. This step makes the interaction more dynamic by adding an auditory element, which can be especially engaging for users who prefer or require auditory feedback.

10. 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.

11. Initialize the Microphone and Create a Sidebar for 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.

12. 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 Gemini AI:** The user's text input, along with the existing conversation history, is sent to the Gemini 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. The response is also converted to speech using gTTS and played back to the user. 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.

The sequential flow of the project is shown in **Figure 2**.

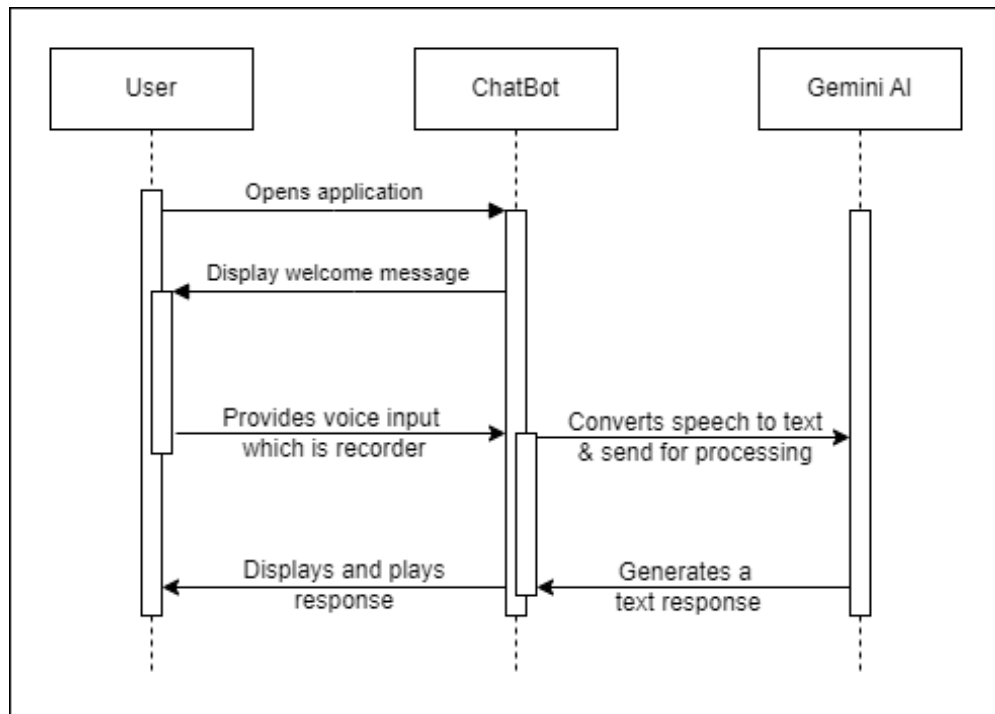
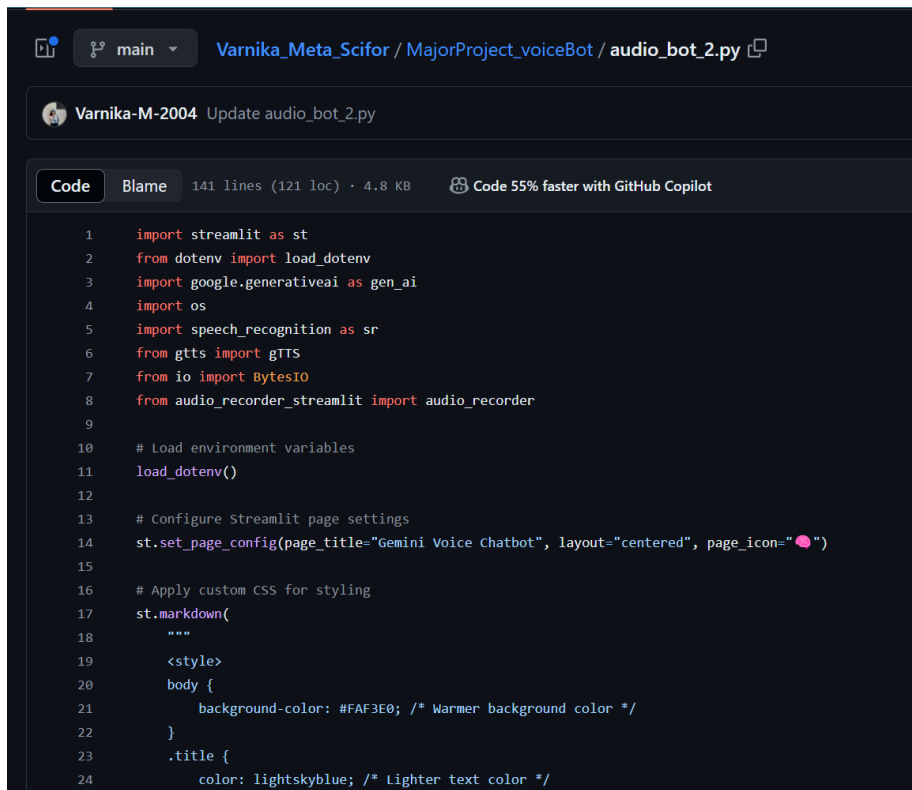


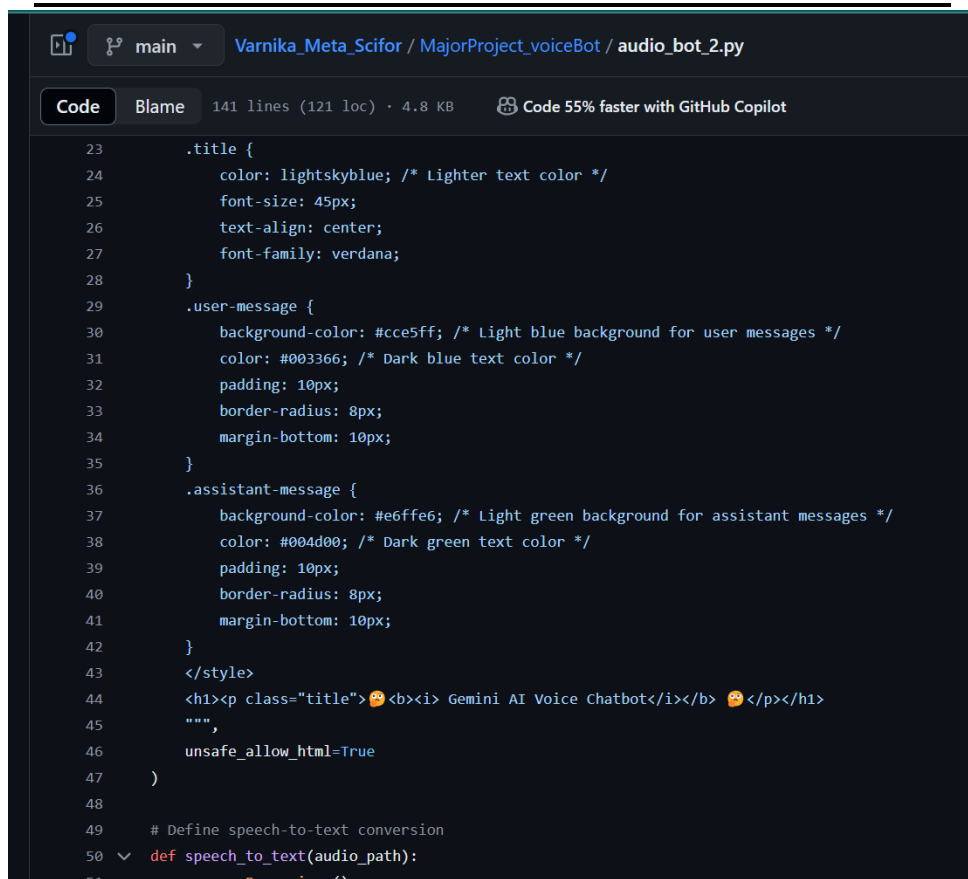
Figure 2. Sequence diagram of study

Figure 2 is a UML diagram representing the sequence of the working of the voice chatbot. It showcases the contribution of each actor (the user, chatbot, and Gemini model) and the interactions between them. The process continues till the user stops asking questions.


Code Snippet



```
1 import streamlit as st
2 from dotenv import load_dotenv
3 import google.generativeai as gen_ai
4 import os
5 import speech_recognition as sr
6 from gtts import gTTS
7 from io import BytesIO
8 from audio_recorder_streamlit import audio_recorder
9
10 # Load environment variables
11 load_dotenv()
12
13 # Configure Streamlit page settings
14 st.set_page_config(page_title="Gemini Voice Chatbot", layout="centered", page_icon="🗣️")
15
16 # Apply custom CSS for styling
17 st.markdown(
18     """
19     <style>
20     body {
21         background-color: #FAF3E0; /* Warmer background color */
22     }
23     .title {
24         color: lightskyblue; /* Lighter text color */
```



```
23     .title {
24         color: lightskyblue; /* Lighter text color */
25         font-size: 45px;
26         text-align: center;
27         font-family: verdana;
28     }
29     .user-message {
30         background-color: #cce5ff; /* Light blue background for user messages */
31         color: #003366; /* Dark blue text color */
32         padding: 10px;
33         border-radius: 8px;
34         margin-bottom: 10px;
35     }
36     .assistant-message {
37         background-color: #e6ffe6; /* Light green background for assistant messages */
38         color: #004d00; /* Dark green text color */
39         padding: 10px;
40         border-radius: 8px;
41         margin-bottom: 10px;
42     }
43     </style>
44     <h1><p class="title">🗣️ <b><i> Gemini AI Voice Chatbot</i></b> 🗣️ </p></h1>
45     """,
46     unsafe_allow_html=True
47 )
48
49 # Define speech-to-text conversion
50 def speech_to_text(audio_path):
51     sr = sr.Recognizer()
```

 main

Varnika_Meta_Scifor / MajorProject_voiceBot / audio_bot_2.py


Code

Blame

141 lines (121 loc) · 4.8 KB

Code 55% faster with GitHub Copilot

```
49 # Define speech-to-text conversion
50 def speech_to_text(audio_path):
51     r = sr.Recognizer()
52     with sr.AudioFile(audio_path) as source:
53         audio_data = r.record(source)
54         try:
55             return r.recognize_google(audio_data)
56         except sr.UnknownValueError:
57             return "Sorry, I did not understand that."
58         except sr.RequestError:
59             return "Sorry, the service is unavailable at the moment."
60
61 # Set up Google API key from Streamlit secrets
62 GOOGLE_API_KEY = st.secrets["google"]["api_key"]
63 gen_ai.configure(api_key=GOOGLE_API_KEY)
64 model = gen_ai.GenerativeModel('gemini-pro')
65
66 # Initialize session state for conversation history
67 if "conversation" not in st.session_state:
68     st.session_state.conversation = []
69     st.session_state.first_interaction = True
70
71 # Convert conversation history format
72 def convert_history(history):
73     converted = []
74     for role, text in history:
75         if role == "user":
76             converted.append({"parts": [{"text": text}], "role": "user"})
```

 main

Varnika_Meta_Scifor / MajorProject_voiceBot / audio_bot_2.py

Code

Blame

141 lines (121 loc) · 4.8 KB

Code 55% faster with GitHub Copilot

```
50 def speech_to_text(audio_path):
71 # Convert conversation history format
72 def convert_history(history):
73     converted = []
74     for role, text in history:
75         if role == "user":
76             converted.append({"parts": [{"text": text}], "role": "user"})
77         elif role == "model":
78             converted.append({"parts": [{"text": text}], "role": "model"})
79     return converted
80
81 # Add the welcome message to conversation history if not already added
82 if st.session_state.first_interaction:
83     welcome_message = "Hello! Welcome to Gemini Chatbot! How may I help you today?"
84     st.session_state.conversation.append(("model", welcome_message))
85
86 # Generate and play the welcome message
87 tts_welcome = gTTS(text=welcome_message, lang='en')
88 welcome_audio = BytesIO()
89 tts_welcome.write_to_fp(welcome_audio)
90 welcome_audio.seek(0)
91
92 # Play the welcome audio
93 st.audio(welcome_audio, format="audio/mp3")
94
95 st.session_state.first_interaction = False
96
97 # Display conversation history
```

```

Varnika_Meta_Scifer / MajorProject_voiceBot / audio_bot_2.py
Code Blame 141 lines (121 loc) · 4.8 KB Code 55% faster with GitHub Copilot

72 def convert_history(history):
97 # Display conversation history
98 for role, text in st.session_state.conversation:
99     if role == "user":
100         st.markdown(f'<div class="user-message">{text}</div>', unsafe_allow_html=True)
101     else:
102         st.markdown(f'<div class="assistant-message">{text}</div>', unsafe_allow_html=True)
103
104 # Sidebar for microphone button
105 with st.sidebar:
106     audio_data = audio_recorder()
107
108 # Process the audio data
109 if audio_data:
110     st.write("Processing audio...")
111
112     with BytesIO(audio_data) as audio_file:
113         # Save audio data to a temporary file
114         temp_audio_path = "temp_audio.wav"
115         with open(temp_audio_path, "wb") as f:
116             f.write(audio_file.read())
117
118         user_prompt = speech_to_text(temp_audio_path)
119
120         # Add user's question to chat and display it
121         st.session_state.conversation.append(("user", user_prompt))
122         st.markdown(f'<div class="user-message">{user_prompt}</div>', unsafe_allow_html=True)
123
124         # Send user's question to Gemini-Pro and get answer

```

```

Varnika_Meta_Scifer / MajorProject_voiceBot / audio_bot_2.py
Code Blame 141 lines (121 loc) · 4.8 KB Code 55% faster with GitHub Copilot

72 def convert_history(history):
117
118     user_prompt = speech_to_text(temp_audio_path)
119
120     # Add user's question to chat and display it
121     st.session_state.conversation.append(("user", user_prompt))
122     st.markdown(f'<div class="user-message">{user_prompt}</div>', unsafe_allow_html=True)
123
124     # Send user's question to Gemini-Pro and get answer
125     conversation_history = convert_history(st.session_state.conversation)
126     try:
127         chat_session = model.start_chat(history=conversation_history)
128         gemini_answer = chat_session.send_message(user_prompt)
129
130         # Display and voice the answer
131         st.session_state.conversation.append(("model", gemini_answer.text))
132         st.markdown(f'<div class="assistant-message">{gemini_answer.text}</div>', unsafe_allow_html=True)
133
134         # Generate and play TTS response
135         tts_response = gTTS(text=gemini_answer.text, lang='en')
136         response_audio = BytesIO()
137         tts_response.write_to_fp(response_audio)
138         response_audio.seek(0)
139         st.audio(response_audio, format="audio/mp3")
140     except Exception as e:
141         st.error(f"An error occurred: {e}")

```

Results and Discussion

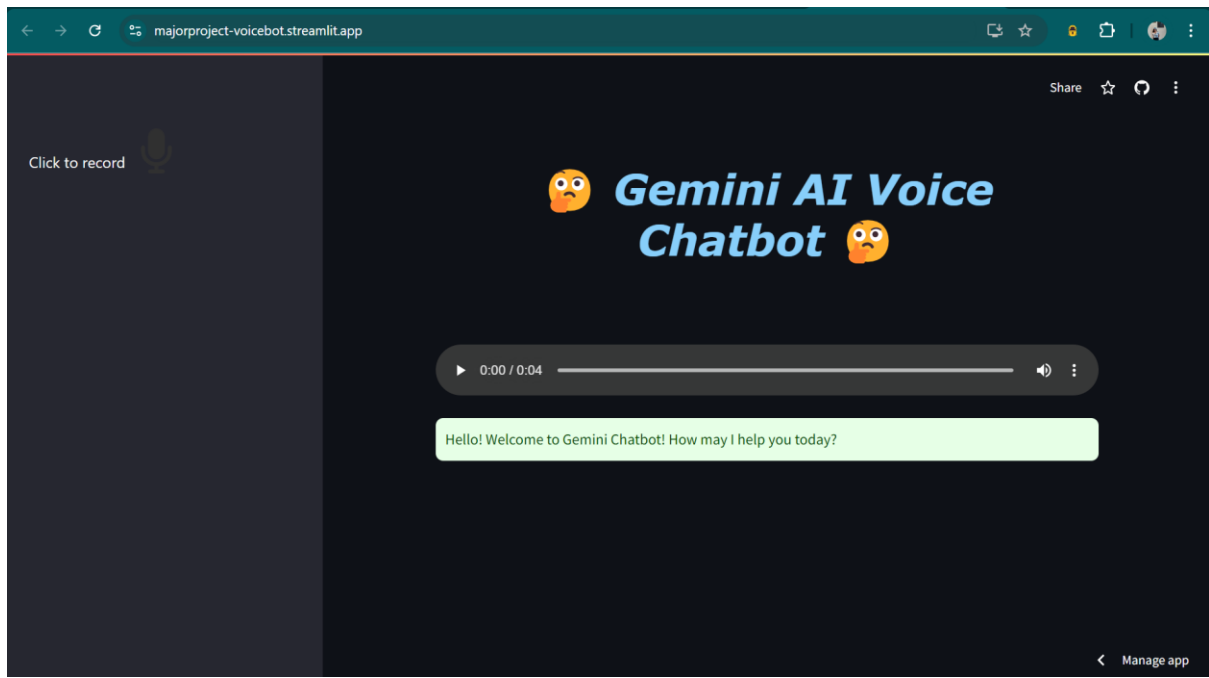


Figure 3. Page setup of chatbot

Figure 3 represents the page setup for the chatbot. The microphone button is present in the sidebar so that its position is fixed and users are easily able to locate and access it.

The first recording is a welcome message from the chatbot. The reason why recorded messages are used in this project is due to the accessibility of users. If suppose a user wants to listen to the chatbot's response, he can simply click on the play button and listen to it and stop whenever he wants. He does not have to listen to the entire response if he is not willing to.

Hence, considering the above reasons, the recorded responses are chosen for this particular project.

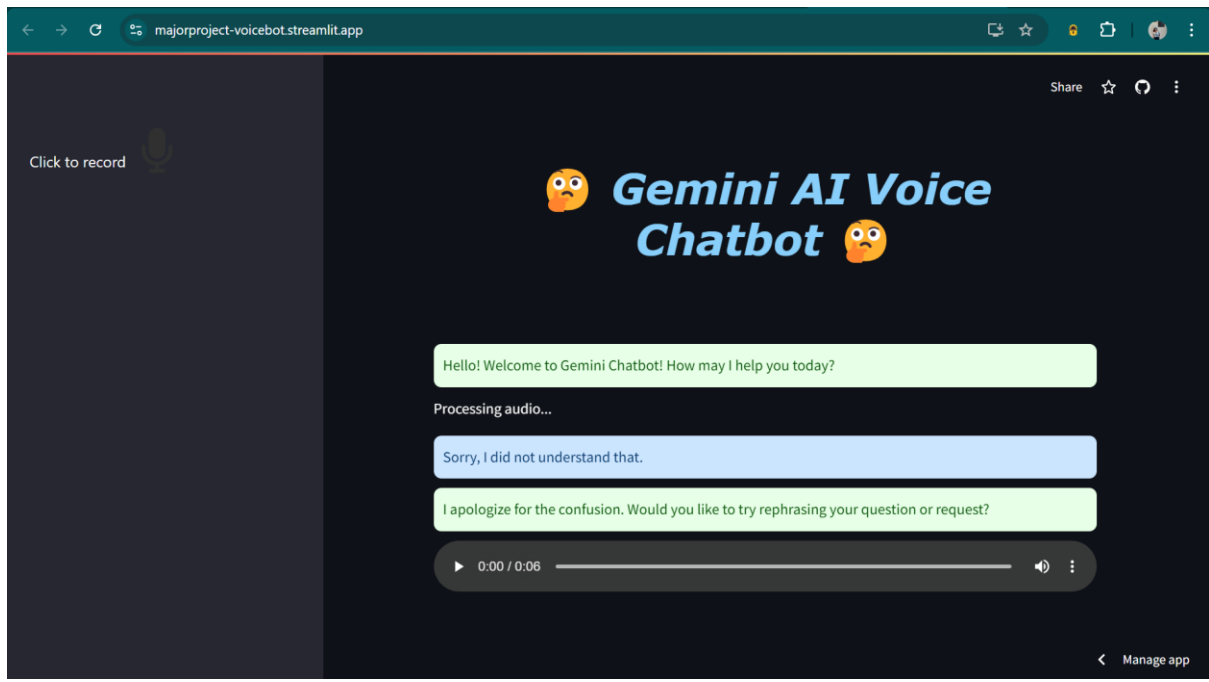


Figure 4. Unknown message error

Figure 4 represents the case of Unknown message error. In this, the bot is unable to identify or analyze what the user is trying to ask due to which it responds with the message “Sorry, I did not understand”.

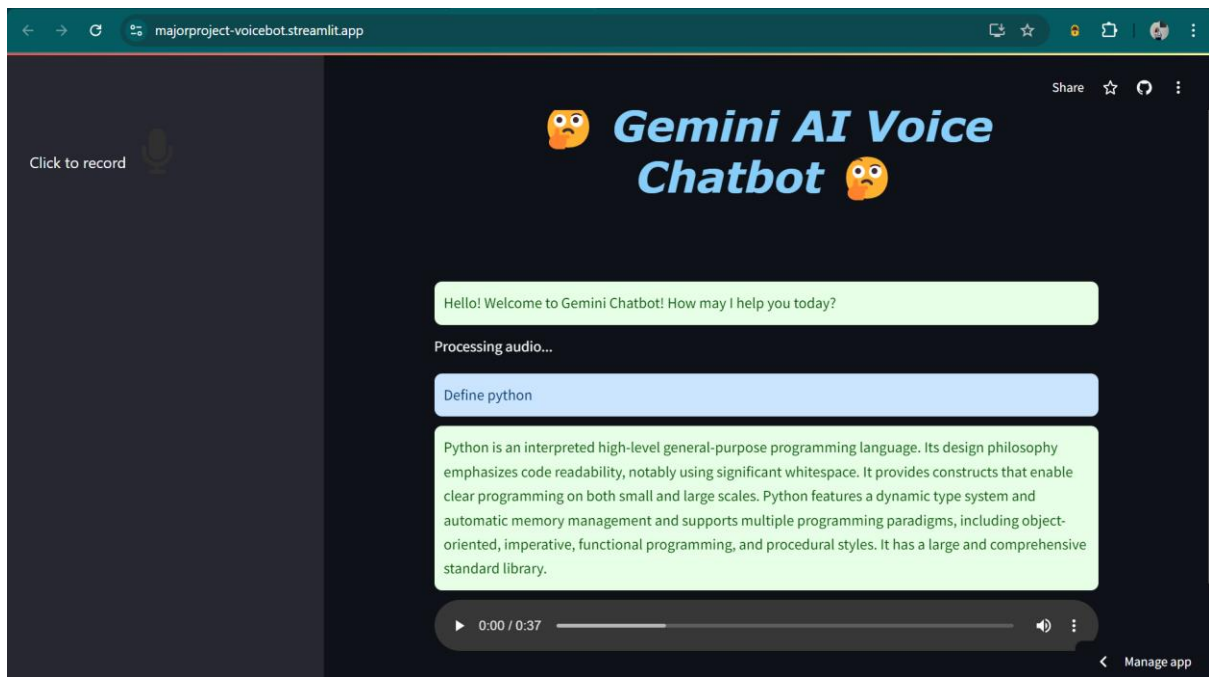


Figure 5. Example of user-bot conversation

Figure 5 depicts an interaction between the user and the chatbot in which the user asks the bot a question and the bot responds with the answer and the recorded message.

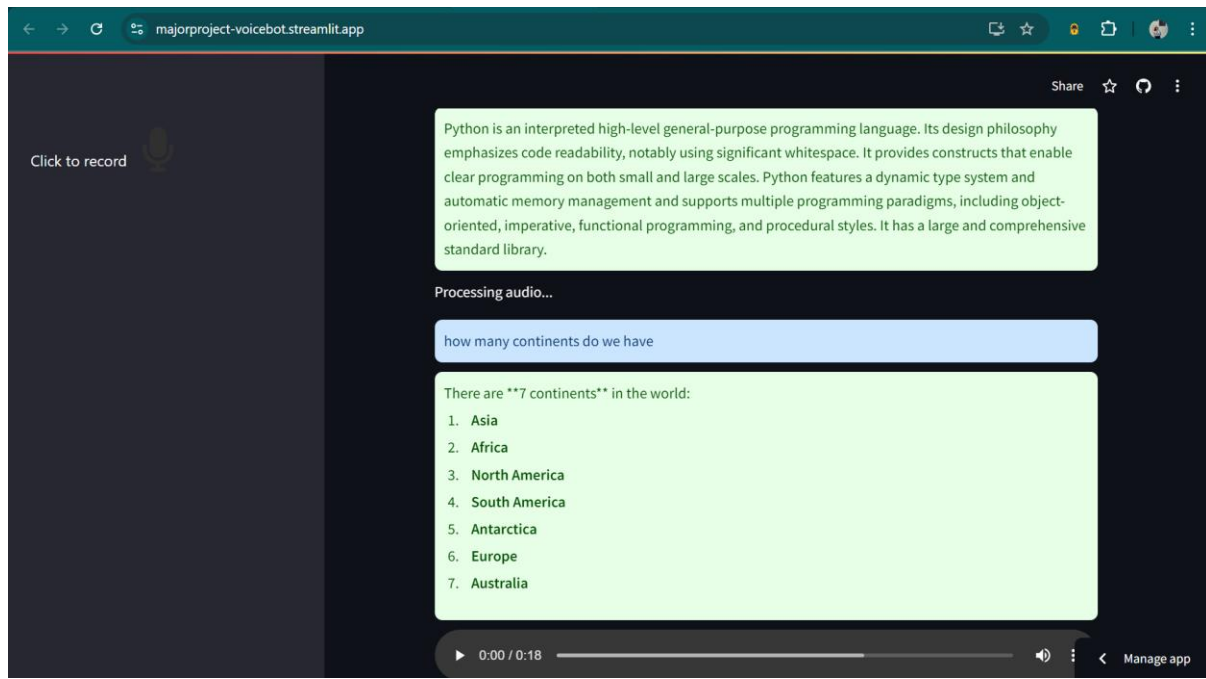


Figure 6. Follow up question by user

Figure 6 is a representation of a scenario in which the user asks a second question to the chatbot. It showcases the chat history and the feature of the bot to save the recorded answers temporarily for each answer.

Overall, the conversational chatbot developed in this project is a great example of an interactive, user-friendly system that answers user queries by providing both an audio response and a text-based response. Additionally, it retains and displays the chat history for the user to review past interactions, making it convenient for the user to track previous questions and responses.

Conclusion

The development of the Gemini Voice Chatbot represents a significant advancement in creating intuitive and interactive conversational agents. By leveraging advanced technologies such as Google's Gemini AI, 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.

The inclusion of custom CSS for styling ensures a visually appealing interface, while the use of gTTS and BytesIO for speech synthesis adds a layer of accessibility and immersion. By maintaining conversation history and integrating voice functionality, the chatbot offers a coherent and engaging user experience that feels both natural and responsive.

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 Gemini 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.

References

- [1] <https://www.analyticsvidhya.com/blog/2023/12/building-a-conversational-qa-chatbot-with-a-gemini-pro-free-api/>
- [2] <https://github.com/chrisMartindiaz/voice-chat-gemini>
- [3] <https://github.com/ilhansevval/Gemini-Chatbot-Interface-with-Streamlit>
- [4] <https://www.linkedin.com/pulse/how-create-gemini-pro-chatbot-using-python-streamlit-hafiz-m-ahmed-pxscf/>
- [5] <https://skolo.online/course-detail/step-by-step-guide-to-building-an-ai-voice-assistant-with-streamlit-openai-111e921a222a>