INTERACTIVE CHATBOT USING GEMINI API AND STREAMLIT

1. Libraries and Modules

```
import streamlit as st
import google.generativeai as genai
```

- streamlit as st: This imports Streamlit, which is used to create interactive web applications directly from Python. It will handle the web interface of the chatbot.
- google.generativeai as genai: This imports the Google Generative AI SDK, which allows interaction with Google's Gemini AI model.

2. API Key Configuration

```
genai.configure(api_key="AIzaSyAL_T-aOIbZXaBtZuheDmPJ2_MKrpeBAV0")
```

- genai.configure(api_key=api_key): This line configures the Google Generative AI SDK by passing the API key, so the application can authenticate requests to the model.

3. Model Initialization

```
model = genai.GenerativeModel("gemini-pro")
```

- genai.GenerativeModel("gemini-pro"): This initializes the **Gemini Pro** model, which is a large language model from Google used to generate responses to user queries.

4. Response Function

```
def get_gemini_response(question):
    response = model.generate_content(question)
    return response.text
```

- This function takes a user's input (question), calls the Gemini Pro model to generate content, and returns the response text. The model's generate_content() method sends the question to the Gemini model and retrieves the generated response.

5. Streamlit Application Setup

```
st.set_page_config(page_title="Gemini Pro Chatbot")
st.header("Gemini Pro Chatbot")
```

- st.set_page_config(page_title="Gemini Pro Chatbot"): This sets the configuration for the Streamlit web page, including the page title.
- st.header("Gemini Pro Chatbot"): This displays a header title for the chatbot interface.

6. Session State for Chat History

```
if 'history' not in st.session_state:
    st.session_state['history'] = []
```

- Streamlit stores information across user interactions through **session state**. Here, st.session_state['history'] is initialized to store the conversation history, ensuring the chat context is maintained even as new questions are asked.

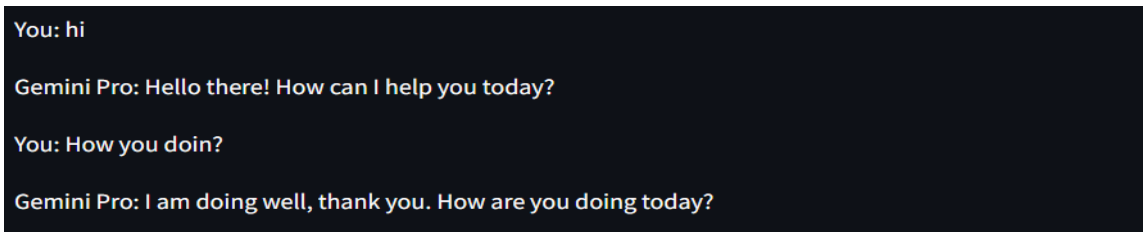
7. Recursive Function for Interaction

```
def rec_fun():
    input = st.text_input("Ask something:", key="user_input")

    if st.button("Generate Response"):
        if input.lower() not in ["quit", "exit", "bye"]:
            response = get_gemini_response(input)

            st.session_state['history'].append({"question": input, "response": response})
```

- st.text_input("Ask something:", key="user_input"): This creates a text input field for the user to type a question. The key "user_input" helps Streamlit manage the state of the input field.
- if st.button("Generate Response"): When the button is clicked, the function gets triggered. If the user hasn't typed "quit", "exit", or "bye", it sends the user's question to the Gemini model and retrieves a response.
- st.session_state['history'].append(): The user's input and the corresponding response are appended to the history in the session state, ensuring that the entire conversation history is stored.

8. Displaying Chat History

```
if st.session_state['history']:
    for chat in st.session_state['history']:
        st.write(f"You: {chat['question']}")
        st.write(f"Gemini Pro: {chat['response']}")
```



- If there is chat history in the session state, the loop iterates through each question-response pair, displaying them on the interface in the format:
    - **You**: [Your Question]
    - **Gemini Pro**: [Model's Response]

9. Recursive Call

```
rec_fun()
```

- Finally, the rec_fun() function is called to initialize the chatbot interface and handle user interactions.

To run the program , we have to type this in the terminal :

```
PS C:\Users\Admin\OneDrive\Documents\ACM projects> streamlit run TASK-4.py
```