**1)Abstract**

**Overview:**

In the rapidly evolving digital landscape, chatbots have become indispensable tools for providing instant responses to user queries. This project focuses on the development of a simple yet effective rule-based chatbot designed to answer predefined questions using Streamlit and the Google Gemini-Pro AI model. By combining the intuitive interface of Streamlit with the advanced natural language processing capabilities of Google Gemini-Pro, the chatbot aims to deliver a seamless and efficient user experience.

**Objective:**

The primary objective of this project is to create a chatbot that can quickly and accurately respond to a set of predefined questions. By using rule-based logic, the chatbot is designed to handle specific queries with precision, making it an ideal solution for scenarios where the questions and answers are known and repetitive. This approach ensures that users receive relevant information without the need for complex machine learning models.

**Scope:**

The scope of this project includes the design, implementation, and deployment of a rule-based chatbot using Streamlit for the user interface and Google Gemini-Pro for the backend processing. The project covers the following aspects:

1. **Loading and managing environment variables securely:** The environment variables, which may include sensitive information such as API keys, are loaded securely using the dotenv library. This ensures that sensitive data is not exposed in the codebase.
2. **Configuring the Streamlit page with user-friendly settings:** The Streamlit page is set up with specific configurations to provide a user-friendly interface. This includes setting the page title, favicon, and layout to enhance the user experience.
3. **Initializing and utilizing the Google Gemini-Pro AI model:** The Google Gemini-Pro AI model is initialized and configured using the API key. This model is then used to process user queries and generate appropriate responses.
4. **Managing chat sessions and displaying chat history:** Chat sessions are managed to ensure continuity of conversation. The chat history is displayed on the Streamlit page to provide context to the ongoing conversation.
5. **Handling user inputs and generating appropriate responses:** User inputs are captured through a chat input field. These inputs are then processed, sent to the AI model, and the generated responses are displayed in the chat interface.

**Importance:**

The importance of this project lies in its ability to provide a practical and efficient solution for handling predefined questions. By leveraging rule-based logic, the chatbot can deliver quick and accurate responses, improving user satisfaction and reducing the need for human intervention. The integration of Streamlit and Google Gemini-Pro also demonstrates the potential of combining user-friendly interfaces with powerful backend processing, making this project a valuable reference for future chatbot developments.

**2)Algorithm**

**Description:**

The algorithm for the rule-based chatbot involves a series of defined steps that facilitate the processing of user inputs and the generation of appropriate responses. The primary components of the algorithm include environment setup, page configuration, AI model initialization, chat session management, role translation, user input handling, and response generation.

**Steps:**

1. **Load Environment Variables:** Use the dotenv library to load environment variables from a .env file, ensuring secure storage and access to sensitive information such as API keys. This step is crucial for keeping sensitive information like API keys and other credentials out of the codebase and thus secure.
2. **Configure Streamlit Page Settings:** Set up the Streamlit page with specific configurations, including the page title, favicon, and layout. These settings enhance the user experience by making the interface more intuitive and visually appealing. For example, setting the page title to "Chat with Gemini-Pro!" and using a cat emoji 😼 as the favicon adds a friendly touch to the interface.
3. **Configure Google Gemini-Pro AI Model:** Initialize and configure the Google Gemini-Pro AI model using the API key from the environment variables. This step involves setting up the AI model to be ready to process user queries and generate responses. The model’s configuration is a one-time setup that prepares it to handle multiple user inputs effectively.
4. **Initialize Chat Session:** Start a new chat session if one does not already exist in the session state, ensuring the continuity of conversations. This involves creating a new session or restoring an existing one to maintain the chat history and context, which is essential for providing coherent responses.
5. **Role Translation Function:** Implement a function to translate roles between Gemini-Pro and Streamlit terminology, ensuring correct identification and display of user and assistant messages. This function helps in mapping the roles used by the AI model to the roles displayed in the chat interface, making it easier to identify who is speaking.
6. **Display Chat History:** Iterate through the chat history and display each message on the Streamlit page, providing context to the ongoing conversation. This helps users to see the previous interactions, which can be helpful in understanding the context of the current conversation.
7. **User Input Handling:** Capture user input through a chat input field, process it, and send it to the AI model for response generation. This step involves taking the user’s query, formatting it as needed, and sending it to the AI model for processing.
8. **Generate Response:** Send the user's message to the Gemini-Pro AI model and receive a response. The AI model processes the user’s input and generates a response based on the predefined rules and logic.
9. **Display Response:** Display the AI model's response in the chat interface, continuing the conversation. This step involves taking the AI model’s response and displaying it in the chat interface so the user can see and interact with it.

**Flowchart:**

1. Start
2. Load environment variables
3. Configure Streamlit page settings
4. Configure Google Gemini-Pro AI model
5. Initialize chat session
6. Implement role translation function
7. Display chat history
8. Capture user input
9. Send user input to AI model
10. Generate response
11. Display response
12. End

**Examples:**

* **Example 1:**
  + User asks: "What is the weather today?"

Chatbot responds: "The weather today is sunny with a high of 25°C."

* **Example 2:**
  + User asks: "How do I reset my password?"
  + Chatbot responds: "To reset your password, go to the settings page and click on 'Forgot Password'."

**3)Source Code**

**Python Code:** The source code for the project includes the implementation of the algorithm steps using Python, Streamlit, and the Google Gemini-Pro AI model. The code is structured to ensure clarity and modularity, making it easy to understand and extend.

import os

import streamlit as st

from dotenv import load\_dotenv

import google.generativeai as gen\_ai

# Load environment variables

load\_dotenv()

# Configure Streamlit page settings

st.set\_page\_config(

    page\_title="Chat with Gemini-Pro!",

    page\_icon="😼",  # Favicon emoji

    layout="centered",  # Page layout option

)

GOOGLE\_API\_KEY = os.getenv("GOOGLE\_API\_KEY")

# Set up Google Gemini-Pro AI model

gen\_ai.configure(api\_key=GOOGLE\_API\_KEY)

model = gen\_ai.GenerativeModel('gemini-pro')

# Function to translate roles between Gemini-Pro and Streamlit terminology

def translate\_role\_for\_streamlit(user\_role):

    if user\_role == "model":

        return "assistant"

    else:

        return user\_role

# Initialize chat session in Streamlit if not already present

if "chat\_session" not in st.session\_state:

    st.session\_state.chat\_session = model.start\_chat(history=[])

# Display the chatbot's title on the page

st.title("🤖 Gemini Pro - ChatBot")

# Display the chat history

for message in st.session\_state.chat\_session.history:

    with st.chat\_message(translate\_role\_for\_streamlit(message.role)):

        st.markdown(message.parts[0].text)

# Input field for user's message

user\_prompt = st.chat\_input("Ask Gemini-Pro...")

if user\_prompt:

    # Add user's message to chat and display it

    st.chat\_message("user").markdown(user\_prompt)

    # Send user's message to Gemini-Pro and get the response

    gemini\_response = st.session\_state.chat\_session.send\_message(user\_prompt)

    # Display Gemini-Pro's response

    with st.chat\_message("assistant"):

        st.markdown(gemini\_response.text)

**Libraries Used:**

1. **Streamlit:** For creating the web interface.
2. **Google Generative AI:** For configuring and using the Google Gemini-Pro AI model.
3. **dotenv:** For loading environment variables.
4. **os:** For accessing environment variables.

**Code Explanation:**

1. **Loading Environment Variables:** The dotenv library is used to load environment variables from a .env file, ensuring that sensitive information such as API keys is securely stored and accessed.
2. **Configuring Streamlit Page Settings:** Streamlit page settings are configured to set the page title, favicon, and layout, enhancing the user interface and providing a consistent look and feel.
3. **Initializing Google Gemini-Pro AI Model:** The Google Gemini-Pro AI model is configured using the API key loaded from the environment variables, making it ready to process user queries and generate responses.
4. **Initializing Chat Session:** A chat session is started if one does not already exist in the session state, maintaining chat history and ensuring a seamless user experience.
5. **Implementing Role Translation Function:** A function is implemented to translate roles between Gemini-Pro and Streamlit terminology, ensuring the correct identification and display of user and assistant messages.
6. **Displaying Chat History:** The chat history is iterated through, and each message is displayed with the appropriate roles, providing context to the ongoing conversation.
7. **Handling User Input:** User input is captured through a chat input field, processed, and sent to the AI model for response generation.
8. **Generating and Displaying Response:** The AI model's response is displayed in the chat interface, continuing the conversation and providing relevant information to the user.

**Examples:**

* **Example 1:**
  + User inputs: "Tell me a joke."
  + Chatbot responds: "Why don't scientists trust atoms? Because they make up everything!"
* **Example 2:**
  + User inputs: "What's the capital of France?"
  + Chatbot responds: "The capital of France is Paris."

**Output:**

A screenshot of a chatbot

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated