|  |
| --- |
| Title: Nexa AI Assistant Chatbot |
| Done by : Pritesh Ram Keshri Project Number : 01, Submission Date: 29th May 2025 |
| **Overview:**  This project creates a user-friendly web interface for interacting with Ollama language models using Streamlit. The application works as an AI Assistant Chatbot which allows users to configure model parameters, customize the AI personality, and engage in conversational interactions through an intuitive chat interface. The system maintains conversation history and provides real-time streaming responses.  Moreover, it would work in offline mode also.    Key features include:  - Model selection and parameter configuration  - Customizable AI personality through system prompts  - Real-time streaming responses with typing indicators  - Session-based chat history management  - Responsive UI with custom styling |
| **Libraries used:**  **streamlit** - For building the web application interface and UI.    **ollama** - Python client for interacting with Ollama models.    **time** - For adding response streaming delays. |
| **Dataset Details:**  No Dataset is used in this project. Only we have used pretrained model llama3.2.  Llama 3.2 utilizes 3billion parameters. These models are designed for various use cases, from lightweight text-only applications on mobile devices to more sophisticated vision and reasoning tasks. |
| **APIs Integrated:**  Ollama Local API (http://localhost:11434) |
| **Source code 1: App Configuration and Styling**  **File Name : Importing libraries, app.py**  import streamlit as st  from ollama import Client  import time    # Initialize Ollama client  client = Client(host='http://localhost:11434')    # App configuration with custom theme  st.set\_page\_config(  page\_title="Nexa AI Assistant",  page\_icon="🤖",  layout="wide",  initial\_sidebar\_state="expanded"  )  # Custom CSS styling  st.markdown("""<style>...custom CSS rules...</style>""",  unsafe\_allow\_html=True) |
| **Source code 2: Sidebar Configuration**  # Sidebar for settings  with st.sidebar:  st.title("⚙️ Nexa Settings")  st.markdown("---")    # Model selection  st.subheader("Model Configuration")  available\_models = ["llama3.2"]  selected\_model = st.selectbox("Select Model", available\_models)    # Configuration sliders  temperature = st.slider("Creativity", 0.0, 1.0, 0.7)  max\_length = st.slider("Response Length", 100, 2000, 500)    # Personality settings  st.subheader("Personality Settings")  system\_prompt = st.text\_area("Customize Nexa's personality",  "You are Nexa, a helpful AI assistant...", height=150)  # Connection status  st.subheader("System Status")  try:  client.generate(model=selected\_model, prompt="test")  st.success("✅ Ollama is connected and ready")  except:  st.error("⚠️ Ollama connection failed") |
| **Source code 3: Main Chat Interface, app.py**  # Main chat area  st.title("💬 Nexa AI Assistant")  st.caption("Your intelligent assistant for all queries")    # Initialize chat history  if "messages" not in st.session\_state:  st.session\_state.messages = []  st.session\_state.messages.append({  "role": "assistant",  "content": "Hello! I'm Nexa..."})  # Display chat history  for message in st.session\_state.messages:  with st.chat\_message(message["role"]):  st.markdown(message["content"]) |
| **Source Code 4: Response Generation**    # Generate response function  |def generate\_response(prompt):  try:  response = client.generate(  model=selected\_model,  prompt=prompt,  system system\_prompt,  options={'temperature': temperature,  'num\_predict': max\_length})  return response['response']  except Exception as e:  return f"⚠️ Error: {str(e)}"    # Chat input processing  if prompt := st.chat\_input("Type your message..."):  # Add user message to chat history  st.session\_state.messages.append({"role": "user", "content": prompt})    # Display user message  with st.chat\_message("user"):  st.markdown(prompt)  # Generate and display assistant response  with st.chat\_message("assistant"):  message\_placeholder = st.empty()  full\_response = ""    # Get response  assistant\_response = generate\_response(prompt)    # Stream response with typing effect  for chunk in assistant\_response.split():  full\_response += chunk + " "  time.sleep(0.05)  message\_placeholder.markdown(full\_response + "▌")  message\_placeholder.markdown(full\_response)    # Add assistant response to history  st.session\_state.messages.append({  "role": "assistant",  "content": full\_response  }) |
| **Source Code 4: Initialization Commands**  **Step-by-step Python Code to Run Ollama 3.2 from Terminal:**  import subprocess  # Function to run a terminal command and print the output  def run\_command(command):  print(f"Running: {command}")  result = subprocess.run(command, shell=True, text=True, capture\_output=True)  print(result.stdout)  if result.stderr:  print("Error:", result.stderr)  # 1. Install Ollama (Linux/macOS) — skip if already installed  # You can comment this out if Ollama is already installed  run\_command("curl -fsSL https://ollama.com/install.sh | sh")  # 2. Start the Ollama service (only necessary if it isn't running)  run\_command("ollama serve &")  # 3. Pull the llama3 model (or any other model you want to use)  run\_command("ollama pull llama3")  # 4. Run a prompt with the model  response = subprocess.run(  ['ollama', 'run', 'llama3', '-p', 'What is the capital of France?'],  text=True,  capture\_output=True  )  print("Model response:", response.stdout) |
| **Output screenshots:**  Here , are the snapshots of my working project.  **1.This is the starting interface of my project.**    **2.Sidebar Configuration : This would be a combination of the model configuration and personality settings.**      **3.Main Chat Interface : The primary chat interface shows a natural conversation flow and the input area.** |
| **What you learned:**  Through this project, I gained hands-on experience with:   * **Streamlit Framework:** Learned to build interactive web apps using Streamlit components. * **Ollama Integration:** Connected to local LLMs through Ollama API. * **Session Management:** Implemented chat history persistence using Streamlit's session state. * **UI Customization:** Created responsive layouts with custom CSS styling. * **Real-time Streaming:** Implemented typing indicators and progressive response display. |
| **What the Skills you gained:**   * Building conversational AI interfaces with Streamlit. * Integrating local language models into web applications. * Creating dynamic UIs with interactive widgets. * Managing application state and session persistence. * Implementing responsive designs with custom CSS. * Handling API communication and error management. |
| **Real Time Applications:**   * **Personal AI Assistant:** Local, private alternative to cloud-based chatbots. * **Education Tool:** Interactive learning companion for students. * **Development Prototyping:** Rapid interface creation for testing. * **Attendance Systems:** Authenticate signatures in paper-based attendance or approval sheets. * **Research Interface**: Configurable platform for experimenting with LLM parameters. |
| **Further Enhancement Suggestions:**   * **Multi-model Support :** Add support for additional Ollama models (Llama 3, Mistral, etc.) * **Conversation Memory :** Implement long-term conversation memory using vector databases. * **File Uploads :** Add document analysis capabilities through file uploads. * **Deployment Options :** Containerize app for easy deployment (Docker). * **Advanced Features :** Add image generation, code execution, or web search capabilities. |