TASK – 2

CONVERSATIONAL KNOWLEDGE BOT

(LANGCHAIN + TOOLS + MEMORY)

Submitted By/-
Gollapally Sai Archan
22d41a6637@gmail.com
6303043817

1. ABSTRACT

The rapid advancement of large language models (LLMs) has enabled the development of intelligent conversational systems capable of understanding context, retrieving external information, and generating human-like responses. This project presents a **Conversational Knowledge Bot** built using **Google Gemini API** and **Streamlit**, designed to provide accurate, contextual, and interactive responses to user queries. The system maintains conversational continuity by leveraging session-based memory and enriches responses using real-time web search, enabling it to answer both factual and follow-up questions effectively. The application demonstrates a practical implementation of LLM-powered conversational AI with an intuitive and visually enhanced user interface.

The proposed solution focuses on stability, usability, and modular design while addressing challenges such as API limitations and evolving library dependencies. By combining prompt-based context handling, external knowledge retrieval, and a responsive chat interface, the bot ensures reliable performance even under constrained free-tier resources. This project highlights the applicability of generative AI in building scalable knowledge-driven assistants and serves as a foundation for further enhancements such as vector-based memory, multi-agent collaboration, and deployment in real-world enterprise and educational scenarios.

2. INTRODUCTION

Conversational Artificial Intelligence has emerged as a key application of generative models, enabling systems to interact with users in natural language while providing meaningful and context-aware responses. With the evolution of large language models (LLMs), conversational agents are no longer limited to predefined scripts but can dynamically reason, retrieve information, and adapt to user intent. These capabilities have significantly expanded the use of AI-powered chatbots across domains such as education, customer support, research assistance, and decision support systems.

The integration of external knowledge sources and conversational memory plays a crucial role in enhancing the effectiveness of such systems. By maintaining context across multiple interactions and incorporating real-time information retrieval, conversational agents can provide more accurate and relevant responses. This project leverages the **Google Gemini API** for natural language understanding and generation, combined with a session-based memory mechanism and web search functionality to ensure continuity and factual correctness throughout the conversation.

This project focuses on the design and implementation of a **Conversational Knowledge Bot** with an intuitive user interface built using **Streamlit**. The system emphasizes modularity, scalability, and robustness while addressing practical challenges such as API rate limits and evolving library dependencies. By demonstrating a complete end-to-end conversational AI application, the project showcases how modern generative AI technologies can be effectively applied to build reliable, user-friendly knowledge assistants.

3. EXISTING SYSTEM

In the existing system, conversational applications are typically built using **rule-based or static response mechanisms** without the integration of advanced language models or external APIs. These systems rely on predefined rules, keyword matching, or fixed decision trees to generate responses. As a result, they are limited in their ability to understand natural language variations, handle complex queries, or maintain meaningful conversational context across multiple interactions.

The base version of such systems does not support real-time information retrieval or dynamic knowledge updates. Responses are generated from a **static, locally stored knowledge base**, which must be manually updated to reflect new information. This approach makes the system inefficient for answering current or factual queries and restricts its usefulness to a narrow set of predefined scenarios.

Furthermore, the absence of conversational memory and intelligent reasoning capabilities limits user experience and scalability. The system cannot adapt to follow-up questions or personalize responses based on previous interactions. Due to these constraints, existing non-API-based conversational systems are unsuitable for modern applications that require flexibility, contextual understanding, and access to up-to-date information.

4. PROPOSED SYSTEM

The proposed system introduces an advanced Conversational Knowledge Bot that leverages modern Large Language Models (LLMs) through secure API-based integration to deliver intelligent, context-aware, and dynamic conversational experiences. Unlike traditional rule-based systems, this solution utilizes the Google Gemini API to understand natural language inputs, generate human-like responses, and adapt to varying user intents. API keys are used to securely authenticate requests, ensuring controlled access, usage monitoring, and scalability of the system in real-world environments.

The system architecture is designed to maintain conversational continuity by incorporating session-based memory, which preserves user interactions during a conversation. This enables the bot to handle follow-up questions, reference previous responses, and provide coherent multi-turn dialogue. Additionally, the proposed system integrates external knowledge retrieval mechanisms, such as web search tools, allowing the bot to fetch up-to-date information dynamically. This significantly enhances response accuracy and relevance, especially for factual and time-sensitive queries.

From an implementation perspective, the proposed system emphasizes modularity, robustness, and user experience. The conversational logic is separated from the user interface, enabling easy maintenance and future upgrades. A Streamlit-based web interface provides an intuitive and visually appealing chat environment, while backend components handle prompt construction, context management, and API communication. Error handling mechanisms are incorporated to manage API rate limits and ensure graceful system behavior under constrained conditions. Overall, the proposed system demonstrates a scalable, intelligent, and future-ready conversational AI solution that overcomes the limitations of non-API-based systems and serves as a strong foundation for further enhancements such as vector-based memory, multi-agent architectures, and enterprise deployment.

5. SYSTEM REQUIREMENTS

This section describes the hardware and software requirements necessary to develop, deploy, and run the Conversational Knowledge Bot using Gemini API. The system is designed to be lightweight, modular, and platform-independent, ensuring compatibility with commonly available computing resources while supporting API-based intelligent conversational functionality.

5.1 Hardware Requirements

The proposed system does not require high-end hardware, as the core computational tasks are handled by cloud-based generative AI models accessed through APIs. The local system primarily manages user interaction, session memory, and request handling. The following hardware specifications are sufficient for smooth execution and testing.

Minimum Hardware Requirements:

- Processor: Intel Core i3 or equivalent
- RAM: 4 GB
- Storage: 5 GB free disk space
- System Type: 64-bit architecture
- Internet Connection: Required for Gemini API access and web-based information retrieval

Recommended Hardware Requirements:

- Processor: Intel Core i5 / AMD Ryzen 5 or higher
- RAM: 8 GB or more
- Storage: SSD with at least 10 GB free space
- Display: Standard monitor with a minimum resolution of 1366 × 768

5.2 Software Requirements: The system is implemented using Python and modern AI development frameworks that support conversational workflows and API-based integrations. All required software components are open-source, well-documented, and widely supported across platforms.

Operating System:

- Windows 10 / 11
- Linux (Ubuntu 20.04 or later)
- macOS (latest stable version)

Programming Language:

- Python 3.9 or higher

Frameworks and Libraries:

- Streamlit – for building the interactive web-based user interface
- LangChain (Google GenAI integration) – for managing prompts and LLM interactions
- DuckDuckGo Search API – for external knowledge retrieval
- Python-dotenv – for managing environment variables and API keys

Development Tools:

- Code Editor / IDE (Visual Studio Code, PyCharm, etc.)
- Web Browser (Google Chrome, Microsoft Edge, or Mozilla Firefox)
- Python Virtual Environment (venv) for dependency management

5.3 Additional Requirements

- Valid API key for Google Gemini or other supported generative AI services
- Stable internet connection for real-time API communication
- Basic knowledge of Python programming, conversational AI concepts, and prompt design
- Familiarity with virtual environments and dependency management.

6. IMPLEMENTATION

📁 Project Structure

Soulpage-genai-assignment-saiarchan 2 /

│

├── app.py # Streamlit chat UI

├── bot.py # LangChain agent + memory

├── tools.py # Search tools

├── requirements.txt

├── README.md

└── .env

app.py

```
import streamlit as st
from bot import chat_with_bot
st.set_page_config(
    page_title="Knowledge Bot",
    page_icon="🤖",
    layout="centered"
)
st.markdown("""
<style>
@import
url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;600&display=swap');
```

```
html, body, [class*="css"] {
    font-family: 'Inter', sans-serif;
}
.stApp {
    background: linear-gradient(135deg, #000000, #1c1c1c, #2b2b2b);
    color: #ffffff;
}
/* Title */
h1 {
    text-align: center;
    font-weight: 600;
    margin-bottom: 5px;
}
.subtitle {
    text-align: center;
    opacity: 0.75;
    margin-bottom: 20px;
}
.chat-container {
    background: rgba(255, 255, 255, 0.08);
    backdrop-filter: blur(12px);
    border-radius: 20px;
    padding: 20px;
    box-shadow: 0 8px 32px rgba(0, 0, 0, 0.6);
```

```
    margin-bottom: 10px;
}
.user-msg {
    background: linear-gradient(135deg, #667eea, #764ba2);
    padding: 12px 16px;
    border-radius: 18px;
    margin: 8px 0;
    text-align: right;
}
.bot-msg {
    background: rgba(255, 255, 255, 0.15);
    padding: 12px 16px;
    border-radius: 18px;
    margin: 8px 0;
    text-align: left;
}
.stChatInput textarea {
    background: #000000 !important;
    color: #ffffff !important;
    border-radius: 14px;
    border: 1px solid #333333;
}
.stChatInput textarea::placeholder {
    color: #aaaaaa;
```

```
}
.stChatInput textarea:focus {
    border-color: #667eea;
    box-shadow: 0 0 0 1px #667eea;
}
</style>
""", unsafe_allow_html=True)
st.markdown("""
<h1>🤖 Conversational Knowledge Bot</h1>
<p class="subtitle">
Powered by Gemini API • Context Memory • Web Search
</p>
""", unsafe_allow_html=True)
if "messages" not in st.session_state:
    st.session_state.messages = []
st.markdown("<div class='chat-container'>", unsafe_allow_html=True)
for msg in st.session_state.messages:
    if msg["role"] == "user":
        st.markdown(
            f"<div class='user-msg'>👨‍💻 {msg['content']}</div>",
            unsafe_allow_html=True
        )
    else:
        st.markdown(
```

```
            f"<div class='bot-msg'>🤖 {msg['content']}</div>",
            unsafe_allow_html=True
        )
st.markdown("</div>", unsafe_allow_html=True)
user_input = st.chat_input("Ask me anything...")
if user_input:
    st.session_state.messages.append(
        {"role": "user", "content": user_input}
    )
    history_text = ""
    for m in st.session_state.messages:
        role = "User" if m["role"] == "user" else "Assistant"
        history_text += f"{role}: {m['content']}\n"
    with st.spinner("Thinking..."):
        response = chat_with_bot(user_input, history_text)
    st.session_state.messages.append(
        {"role": "assistant", "content": response}
    )
    st.rerun()
```

bot.py

```
from langchain_google_genai import ChatGoogleGenerativeAI
from tools import web_search
import os
from dotenv import load_dotenv
```

```
load_dotenv()
llm = ChatGoogleGenerativeAI(
    model="models/gemini-2.5-flash",
    temperature=0.3,
    google_api_key=os.getenv("GOOGLE_API_KEY")
)
def chat_with_bot(user_input: str, chat_history: str) -> str:
    search_result = web_search(user_input)
    prompt = f"""
You are a conversational knowledge bot.
You remember previous conversation context.
Conversation so far:
{chat_history}
Relevant web information:
{search_result}
User question:
{user_input}
Give a clear, factual, and contextual answer.
"""
    response = llm.invoke(prompt)
    return response.content
```

. env:

```
GOOGLE_API_KEY=AIzaSyD9kllU00y_mCAouSJbxpfLL-nikkIDrqc
```

tools.py

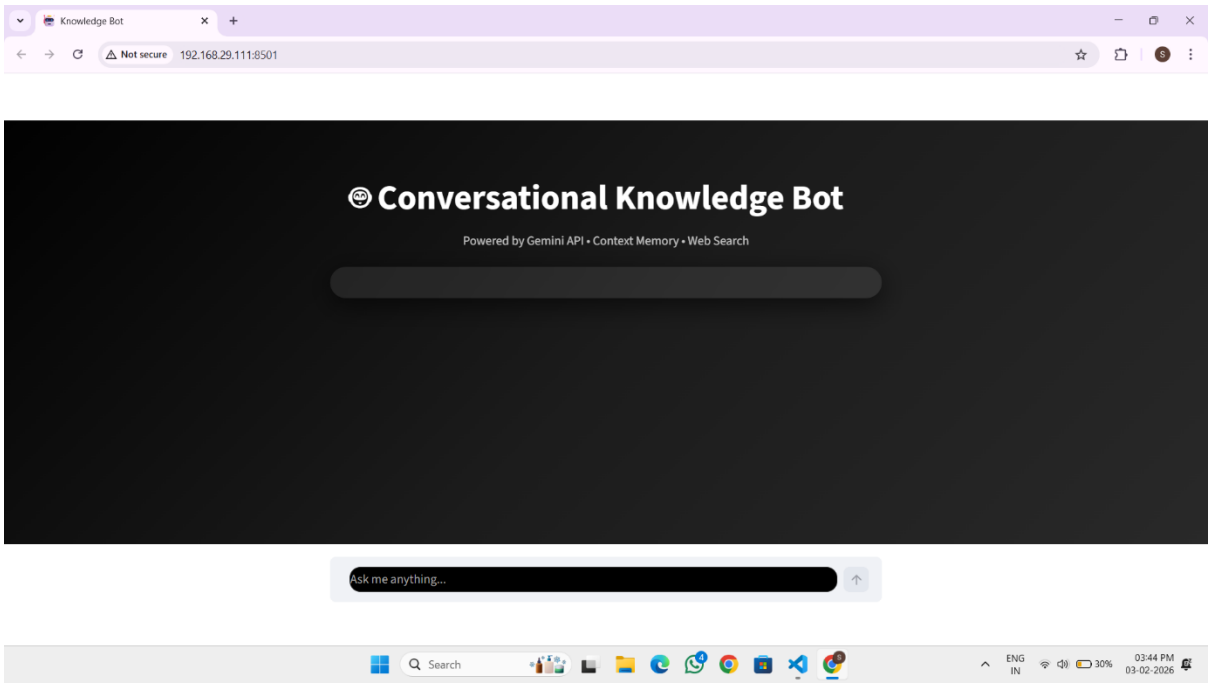
```
from duckduckgo_search import DDGS

def web_search(query: str) -> str:
    with DDGS() as ddgs:
        results = ddgs.text(query, max_results=3)
        return "\n".join([r["body"] for r in results])
```

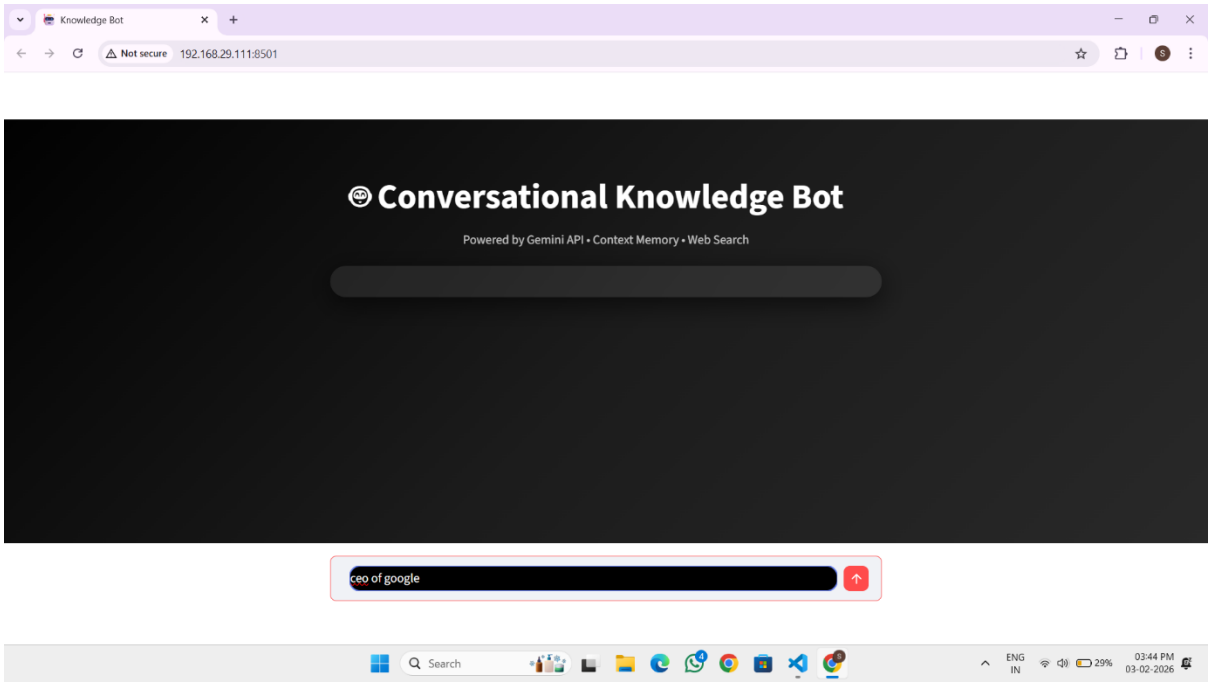
requirements.txt

```
langchain-google-genai
streamlit
duckduckgo-search
python-dotenv
```

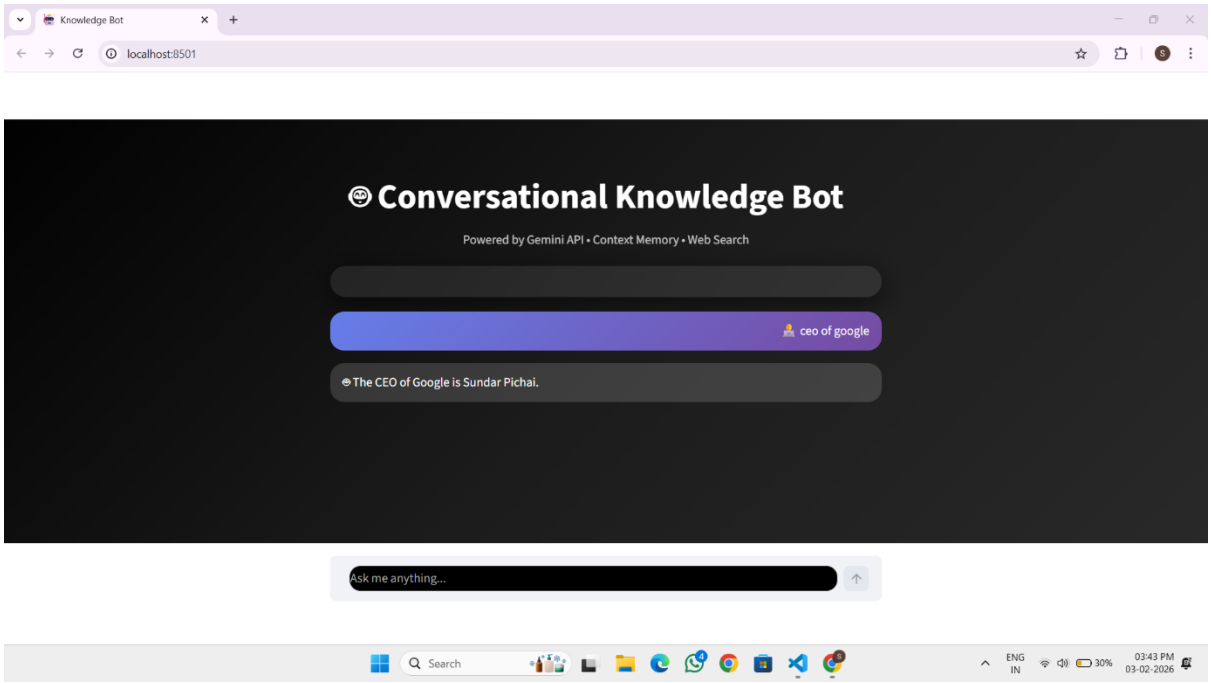
7. RESULT



Home Page



Question Page


Knowledge Bot
localhost:8501
Conversational Knowledge Bot
Powered by Gemini API • Context Memory • Web Search
ceo of google
The CEO of Google is Sundar Pichai.
Ask me anything...
Search
ENG
IN
30%
03:43 PM
03-02-2026

Answer page

8. CONCLUSION

The development of the Conversational Knowledge Bot demonstrates the effective application of generative artificial intelligence and API-driven architectures in building intelligent, context-aware conversational systems. By integrating the Google Gemini API with a modular backend and an interactive Streamlit-based user interface, the system successfully delivers accurate, coherent, and real-time responses to user queries. The incorporation of session-based conversational memory and external knowledge retrieval enhances the system's ability to handle multi-turn conversations and factual queries, addressing the limitations of traditional rule-based chat systems.

Overall, the proposed system provides a scalable and flexible foundation for future conversational AI applications. Its lightweight design, minimal hardware requirements, and robust error-handling mechanisms make it suitable for academic, research, and practical use cases. The project highlights how modern AI tools can be leveraged to create reliable and user-friendly knowledge assistants, with potential for further enhancements such as vector-based long-term memory, multi-agent collaboration, and deployment in enterprise or cloud-based environments.