AI-Powered Mental Health Chatbot

Internship Project Report

By - Manini

Abstract

This project presents the **design and development** of an AI-powered Mental Health Chatbot using **Retrieval-Augmented Generation (RAG)** to deliver *empathetic and context-aware* responses. It integrates **LangChain**, **Groq-hosted LLaMA 3.3**, and **ChromaDB** for retrieving relevant information and generating responses. A **Gradio interface** enables smooth *real-time conversations*, demonstrating the efficient use of *LLMs, semantic search*, and *vector embeddings*.

1 Introduction

Mental health support often lacks personalization and contextual understanding. This project develops an **AI-driven chatbot** that retrieves mental health information from PDFs and generates **empathetic, accurate responses** using **LLMs and RAG**. It combines document-based retrieval with generative AI to provide informative, context-aware assistance.

2 Tools & Technologies Used

- **LangChain** – Manages LLM pipelines, prompt templates, and retrieval systems.
- **Groq-hosted LLaMA 3.3 Model** – Generates conversational responses.
- **Chroma Vector Database** – Stores document embeddings for semantic search.
- **HuggingFace** – Converts text to vector representations using `all-MiniLM-L6-v2`.
- **PyPDFLoader** – Extracts structured text from PDFs.
- **Gradio** – Creates a web-based chatbot interface.
- **Python Libraries** – `os`, `langchain`, `chromadb`, `sentence-transformers`.

3 Steps Involved in Building the Project

3.1 1.Installing Required Libraries

```
!pip install langchain_groq langchain_core langchain-community
!pip install pypdf chromadb sentence_transformers --upgrade gradio
```

3.2 2. Initializing the LLaMA 3.3 Model

```
from langchain_groq import ChatGroq
llm = ChatGroq(
    temperature=0,
    groq_api_key="YOUR_API_KEY",
    model_name="llama-3.3-70b-versatile")
result = llm.invoke("Who is lord Ram?")
print(result.content)
```

3.3 3.Defining the LLM Initialization Function

```
def initialize_llm():
    llm = ChatGroq(
        temperature=0,
        groq_api_key="YOUR_API_KEY",
        model_name="llama-3.3-70b-versatile")
    return llm
```

3.4 4.Creating the Vector Database

```
def create_vector_db():
    loader = DirectoryLoader("/content/data", glob="*.pdf", loader_cls=PyPDFLoader)
    documents = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
    texts = text_splitter.split_documents(documents)
    embeddings = HuggingFaceBgeEmbeddings(model_name="sentence-transformers/all-MiniLM-
        L6-v2")
    vector_db = Chroma.from_documents(texts, embeddings, persist_directory="./chroma_db"
        )
    vector_db.persist()
    return vector_db
```

3.5 5. Setting Up the RetrievalQA Chain

```
def setup_qa_chain(vector_db, llm):
    retriever = vector_db.as_retriever()
    prompt_template = """ You are a compassionate mental health chatbot.Respond
        thoughtfully: {context}
    User: {question}
    Chatbot:"""
    PROMPT = PromptTemplate(template=prompt_template, input_variables=["context", "
        question"])
    qa_chain = RetrievalQA.from_chain_type(
    llm=llm,chain_type="stuff",retriever=retriever,chain_type_kwargs={"prompt": PROMPT}
        )
    return qa_chain
```

3.6 6. Initializing Chatbot and Loading Data

```
print("Initialising Chatbot....")
llm = initialize_llm()
db_path = "/content/chroma_db"
if not os.path.exists(db_path):
    vector_db = create_vector_db()
else:
    embeddings = HuggingFaceBgeEmbeddings(model_name="sentence-transformers/all-MiniLM-
        L6-v2")
    vector_db = Chroma(persist_directory=db_path, embedding_function=embeddings)
qa_chain = setup_qa_chain(vector_db, llm)
```

3.7 7. Generating Chatbot Responses

```
def chatbot_response(user_input, history):
    if not user_input.strip():
        return history
    try:
        response = qa_chain.run(user_input)
    except Exception as e:
        response = f"Error: {e}"
    history.append((user_input, response))
    return history
```

3.8 8. Building the Gradio Chat Interface

```
with gr.Blocks(theme='earneleh/paris') as app:
    chatbot = gr.Chatbot()
    user_input = gr.Textbox(placeholder="Type your message here...")
    send_btn = gr.Button("Send")

    def respond(user_message, chat_history):
        chat_history = chatbot_response(user_message, chat_history)
        return chatbot, chat_history
    send_btn.click(respond, inputs=[user_input, chatbot], outputs=[chatbot, chatbot])
app.launch()
```

4 Conclusion

The Mental Health Chatbot integrates **LLMs, RAG, ChromaDB, and Gradio** into a scalable conversational agent for **empathetic, context-aware responses**. Future work includes *multilingual support*, *better response filtering*, and enhanced *safety mechanisms*.