Subject: Large Language Models(UE21CS421AC1)

Hackathon: Pharma Knowledge Assistant — ESA

TEAM - 5

1. Anshul Ranjan (PES1UG21CS096)
2. Ashwin Ragupathy (PES1UG21CS123)
3. Chandana S M (PES1UG21CS149)
4. Gariman Gangwani (PES1UG21CS202)
5. Vandana S (PES1UG21CS697)

Table of Contents

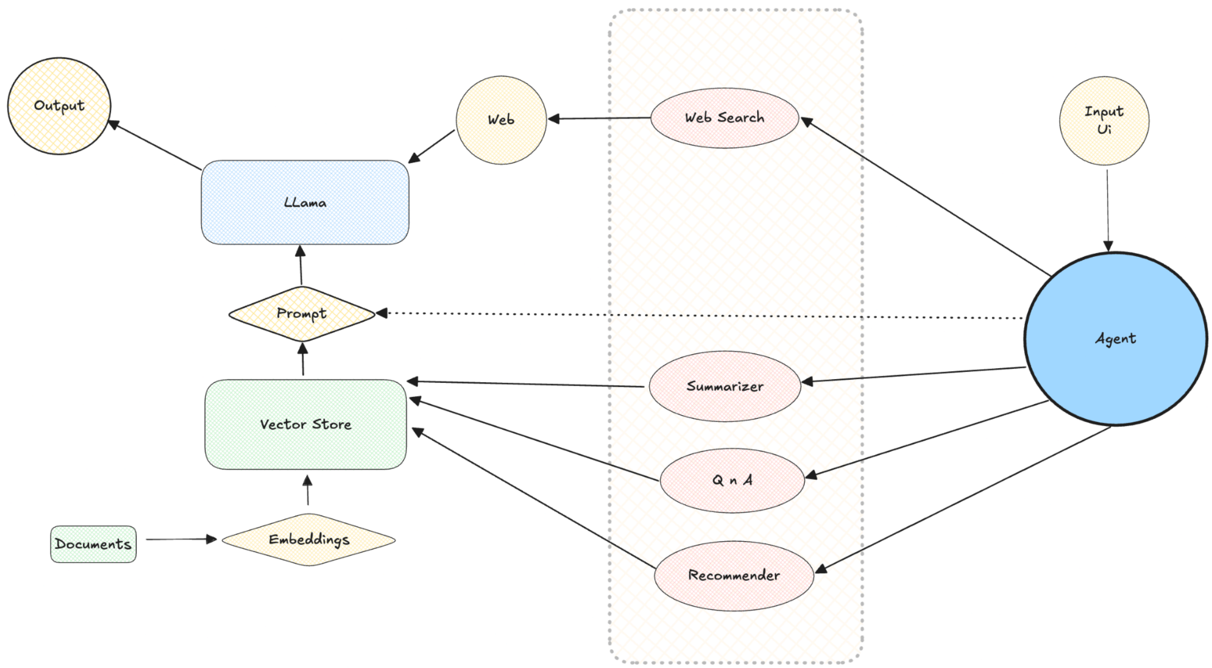
1. Problem Statement

In the pharmaceutical industry, professionals such as pharmacists, doctors, and researchers often face challenges when retrieving relevant information about drug products, alternative medicines, or treatment options. The vast amount of medical literature, drug databases, and research papers make it time-consuming and difficult to extract actionable insights. Moreover, users frequently struggle with accessing real-time information about drug interactions, side effects, and alternative therapies.

Currently available systems either focus on limited datasets, provide static information, or lack real-time interactivity, making them inefficient for quick decision-making in fast-paced clinical environments. Additionally, professionals often require personalized recommendations or specific answers to niche medical queries that cannot be easily found using conventional search engines.

This project aims to address these issues by developing an intelligent Pharma Knowledge Assistant that utilizes modern Natural Language Processing (NLP) techniques and large-scale drug datasets. The system will leverage Retrieval-Augmented Generation (RAG) for real-time question answering, summarize product information, recommend alternative drugs, and assist users in navigating complex medical data. By offering a user-friendly interface and powerful querying capabilities, the Pharma Knowledge Assistant seeks to improve the accessibility and accuracy of pharmaceutical information, ultimately enhancing decision-making processes in the healthcare domain.

2. Architecture



1. **Input UI:** Accepts user input, which is processed by the agent.
2. **Agent:** Orchestrates interactions among:
   - **Web Search** for fetching online data.
   - **Summarizer, QnA, Recommender** modules, which use LLMs for summarizing, answering questions, and providing recommendations. These use the RAG techniques for providing the answers to queries.
   - **Vector Store**, which stores embeddings derived from documents for querying or contextual prompts.
3. **Vector Store:** Acts as a knowledge base, linking embeddings to generate prompts for the LLM.
4. **Output UI:** Presents results from the LLM or modules.

The system integrates external document embeddings, LLM capabilities, and web search to provide dynamic, context-aware responses.

3. **Implementation Steps**

2.1 Develop a RAG Application for Question Answering

The Retrieval-Augmented Generation (RAG) application was developed using LangChain and ChromaDB to create an intelligent question-answering system for medical information. The implementation involved several key steps:

1. **Data Preprocessing**:
    - JSON files containing medical information were preprocessed using a `RecursiveCharacterTextSplitter`
    - Documents were split into chunks of 1000 characters with a 200-character overlap
    - Each chunk was converted into a LangChain `Document` with metadata including section and source file
2. **Vector Store Creation**:
    - OpenAI embeddings were used to convert text chunks into vector representations
    - ChromaDB was utilized as the vector store for efficient similarity search
    - A custom function `flush_chroma_db()` was implemented to reset the vector store between iterations
3. **Custom LLM Integration**:
    - A custom `LMStudioLLM` class was created to integrate with a local LM Studio endpoint
    - The class handles API calls to the language model, supporting custom prompts and error handling
    - Implemented with timeout and error management for robust interaction
4. **Retrieval and Generation**:
    - Used LangChain's `RetrievalQA` to combine retrieval and generation

- Configured the retriever to return the top 5 most similar documents
- Designed to return both the generated response and source documents

2.2 Implement a Recommender

1. User Query Input and Semantic Search

- The system begins by taking a user-provided medical query, such as alternatives to specific medications or treatment recommendations.
- OpenAI embeddings are used to convert the query into a vector representation.
- The query vector is compared against a ChromaDB vector store of pre-built medical documents using semantic similarity search techniques.
- The top 5 most relevant documents are retrieved for further analysis.

2. Context Compilation

- The retrieved documents are parsed, and their content is aggregated into a single comprehensive context.
- Redundancies and overlaps are handled to ensure the context is cohesive and non-repetitive.
- Metadata, such as the source and section of the document, is retained for traceability.

3. Custom LLM Integration for Analysis

- A custom Language Model (LLM) is used to process the compiled context.
- The model evaluates the context to extract actionable insights, such as alternative treatments, safety considerations, and reasoning.
- It dynamically adjusts its responses based on the nature of the user query, supporting flexibility and precision.

4. Recommendation Generation

- The LLM generates detailed recommendations, highlighting:
  - Potential medication or treatment alternatives.
  - Risks and safety measures associated with the recommendations.
  - Contextual insights relevant to the user's conditions or query.
- Responses are personalized to address user-specific conditions, ensuring relevancy.

5. Complex Query Handling

- The system supports a wide range of complex medical queries, leveraging the flexibility of the LLM and semantic retrieval mechanism.
- It effectively combines the retrieval results with model-generated insights to deliver nuanced and comprehensive recommendations.

6. Personalization and Contextual Insights

- The recommendations emphasize user-centric considerations, such as pre-existing conditions or specific medical histories.
- Contextual nuances are provided to aid in making informed medical decisions.

2.3 Implement an Alternatives Generator

The alternatives generator is implemented as a web-enabled agent framework designed to provide flexible and dynamic information retrieval and suggestion capabilities.

Key Components:

1. Web Search Integration

  - Utilizes DuckDuckGo search as the primary information retrieval tool

- Enables dynamic exploration of alternative information sources
- Supports cross-referencing and contextual recommendations

2.Agent-Based Architecture

- Implemented using LangChain's zero-shot React agent
- Dynamically interprets and responds to diverse query types
- Provides flexible information generation and alternative suggestions

Workflow :

1. Query Processing
   - Accepts input queries across various domains
   - Leverages LLM to interpret and decompose query intent
   - Utilizes DuckDuckGo search to gather alternative information
2. Alternative Generation Strategy
   - Searches web resources for contextually relevant alternatives
   - Generates comprehensive responses exploring multiple perspectives
   - Supports discovery of diverse information and suggestions

2.4 Implement a Summarizer

A specialized summarization function `optimized_summarizer()` was developed to generate concise summaries:

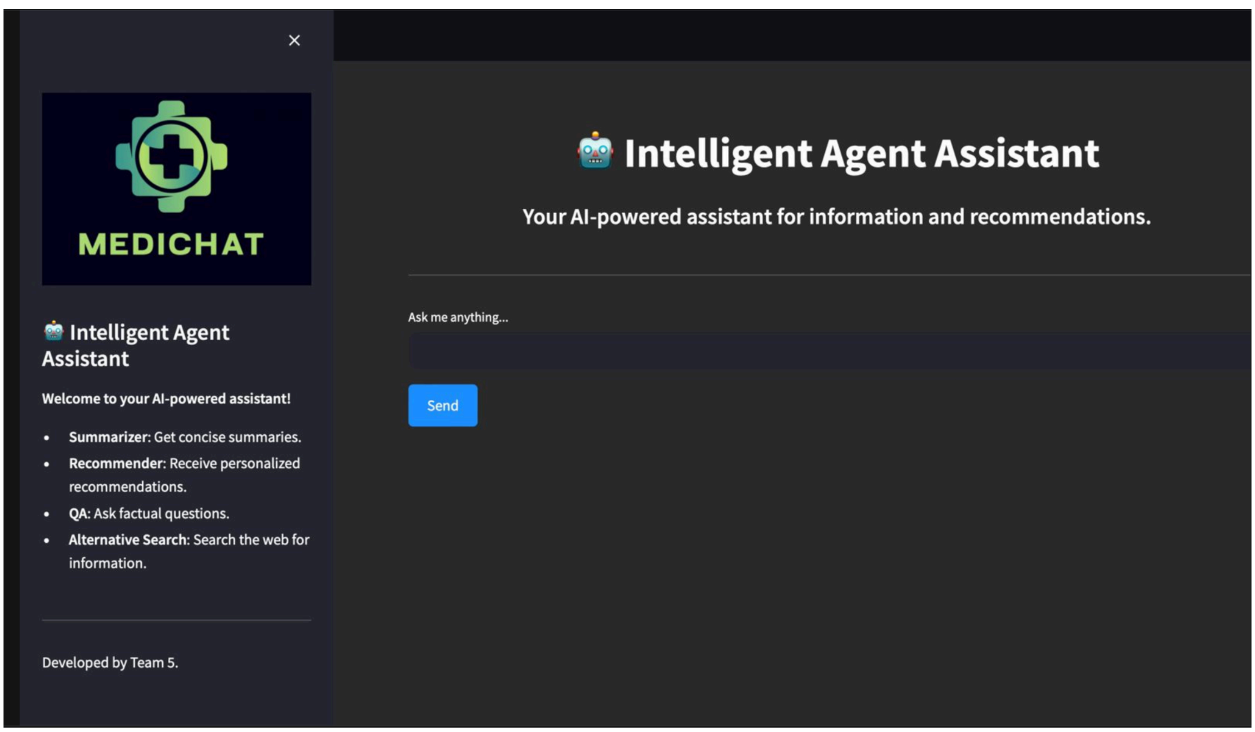
1. **Summarization Process**:
   - Retrieves relevant documents based on the query
   - Combines document content into a single text
   - Uses a custom prompt to guide the LLM in generating a focused summary
   - Supports various medical information summary requests
2. **Key Features**:
   - Flexible query handling
   - Retrieves 3 most relevant documents

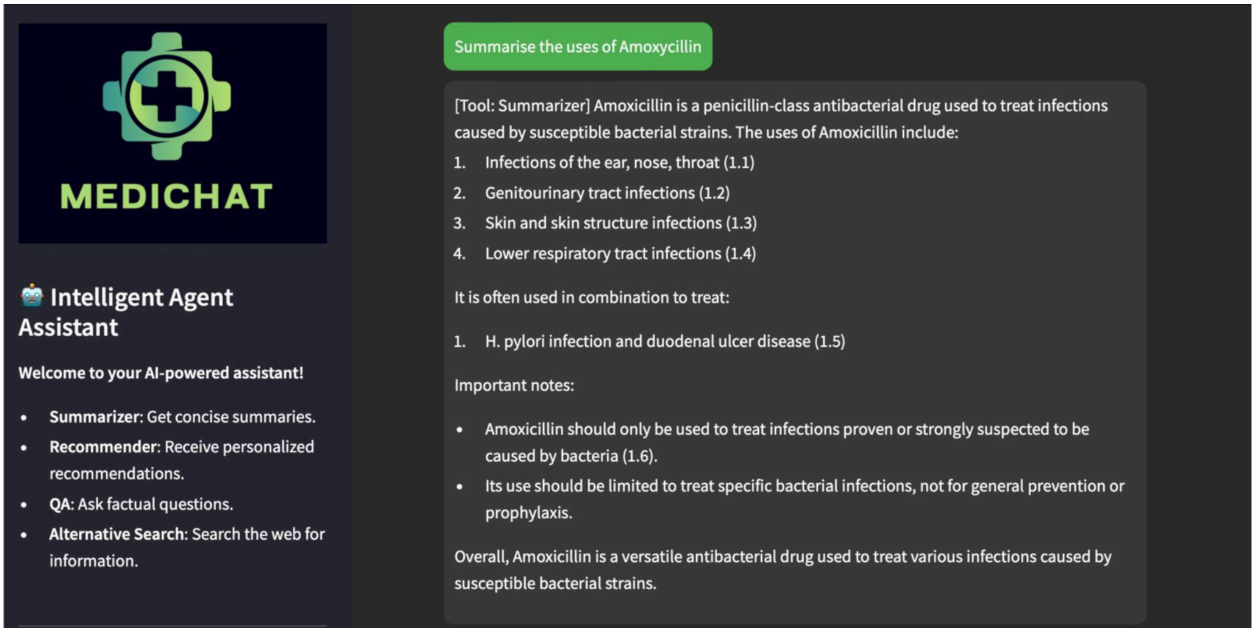
○ Generates summaries across diverse medical topics

2.5 Implementing an Agent-Based Framework

The architecture implements an Agent-Based Framework where the Agent, powered by an LLM, acts as the central decision-maker to handle user inputs and dynamically choose appropriate tools. The Agent interprets queries, determines whether to fetch data from a local Vector Store or perform a Web Search, and routes the information to specialized modules like Summarizer, QnA, or Recommender, all powered by LLMs. The LLM enables the Agent to reason, adapt, and integrate outputs from these tools into a cohesive response. This design ensures flexibility, scalability, and effective task orchestration for diverse user queries

3. Results




MEDICHAT
🤖 Intelligent Agent Assistant
Welcome to your AI-powered assistant!
• Summarizer: Get concise summaries.
• Recommender: Receive personalized recommendations.
• QA: Ask factual questions.
• Alternative Search: Search the web for information.
Summarise the uses of Amoxycillin
[Tool: Summarizer] Amoxicillin is a penicillin-class antibacterial drug used to treat infections caused by susceptible bacterial strains. The uses of Amoxicillin include:
1. Infections of the ear, nose, throat (1.1)
2. Genitourinary tract infections (1.2)
3. Skin and skin structure infections (1.3)
4. Lower respiratory tract infections (1.4)
It is often used in combination to treat:
1. H. pylori infection and duodenal ulcer disease (1.5)
Important notes:
• Amoxicillin should only be used to treat infections proven or strongly suspected to be caused by bacteria (1.6).
• Its use should be limited to treat specific bacterial infections, not for general prevention or prophylaxis.
Overall, Amoxicillin is a versatile antibacterial drug used to treat various infections caused by susceptible bacterial strains.