INFO - 698
Capstone Project

Dr Greg Chism, Dr Enrique Noriega

PubMed Agentic Retrieval-Augmented Generation

Project Final Report

By

Syed Junaid Hussain, Dhawal Gajwe, Abhishek Kumar, Abhay Kumara Sri Krishna Nandiraju

THE UNIVERSITY OF ARIZONA

PubMed Agentic Retrieval-Augmented Generation

Abstract. The rapid growth of medical research publications necessitates efficient systems for retrieving and synthesizing information. This project addresses this need by developing a Retrieval-Augmented Generation (RAG) agent for PubMed research papers. The system processes 10,000 NXML files, extracting key metadata such as PMID, authors, journal, title, and text.The extracted data is chunked and transformed into embeddings using the S-PubMed BERT-MS-MARCO embedding generator, which are stored in a Weaviate vector database for fast retrieval. A JS2 instance optimizes the processes of chunking, embedding generation, and database uploads. Using LangChain, the RAG agent integrates the vector database with a large language model (LLM) to generate contextually relevant responses. Deployed as a chatbot via Streamlit, it provides an intuitive interface for querying medical research. This report details the system architecture, design choices, challenges, and performance comparisons, showcasing the potential of RAG agents in advancing medical information retrieval.

Introduction

The vast and ever-growing volume of medical research publications presents a significant challenge for researchers, clinicians, and other stakeholders in efficiently accessing and synthesizing relevant information. PubMed alone hosts millions of research papers, with thousands added daily, covering a wide range of medical topics. While this wealth of information is invaluable, the lack of efficient tools to retrieve and synthesize relevant knowledge poses a significant barrier to its practical application. Traditional methods of information retrieval are no longer sufficient to meet the demands of modern research workflows, especially in critical fields like medicine where timely and accurate information is essential.

This project aims to address these challenges by developing a Retrieval-Augmented Generation (RAG) agent specifically designed for research papers from PubMed. The primary objective is to create a system that can process, store, and retrieve information from a large corpus of medical research papers, enabling users to query the system and receive accurate, contextually relevant responses. The system processes approximately 10,000 NXML files, extracting key metadata such as PMID, authors, journal, title, and text. These extracted chunks are then transformed into embeddings using the S-PubMedBERT-MS-MARCO embedding generator, which are stored in a Weaviate vector database for efficient retrieval.

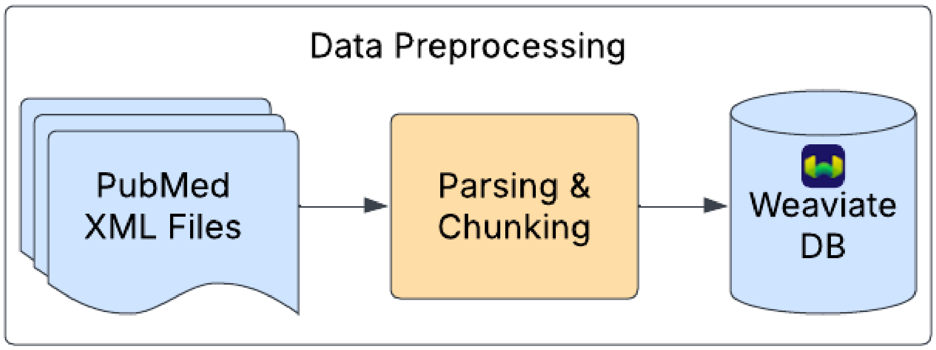
In the context of modern AI research, this project aligns with the industry's focus on Retrieval-Augmented Generation systems, which combine the power of large language models (LLMs) with efficient retrieval mechanisms. By leveraging state-of-the-art technologies such as S-PubMedBERT-MS-MARCO for embedding generation, Weaviate for vector storage, and LangChain for LLM integration, this project demonstrates the practical application of cutting-edge AI techniques in a domain as critical as medicine. The use of RAG agents is particularly relevant in AI research today, as they address the limitations of standalone LLMs by grounding their responses in factual, retrievable data, ensuring accuracy and reliability.

To enable intelligent responses, the RAG agent integrates the vector database with a large language model (LLM) using LangChain. The system is deployed as an interactive chatbot via Streamlit, providing an intuitive interface for users to explore the repository of medical research. This approach not only enhances the accessibility of medical knowledge but also demonstrates the potential of RAG agents in revolutionizing information retrieval in specialized domains.

This project has the potential to significantly impact the medical field by empowering researchers and clinicians with tools to access and apply knowledge more effectively. For researchers, it provides a streamlined way to explore vast repositories of medical literature, accelerating the pace of discovery and innovation. For clinicians, the system offers a means to retrieve accurate, evidence-based information in real time, enhancing their reasoning capacity and decision-making when dealing with live patients. By integrating advanced AI technologies with medical research, this project not only advances the state of information retrieval but also contributes to improving patient care and outcomes in real-world medical settings.

This report provides a comprehensive overview of the system architecture, design choices, challenges encountered, and mitigation strategies. It also includes a detailed analysis of the data and performance comparisons between Llama 3.2 and GPT-4o, highlighting the strengths and limitations of the system.

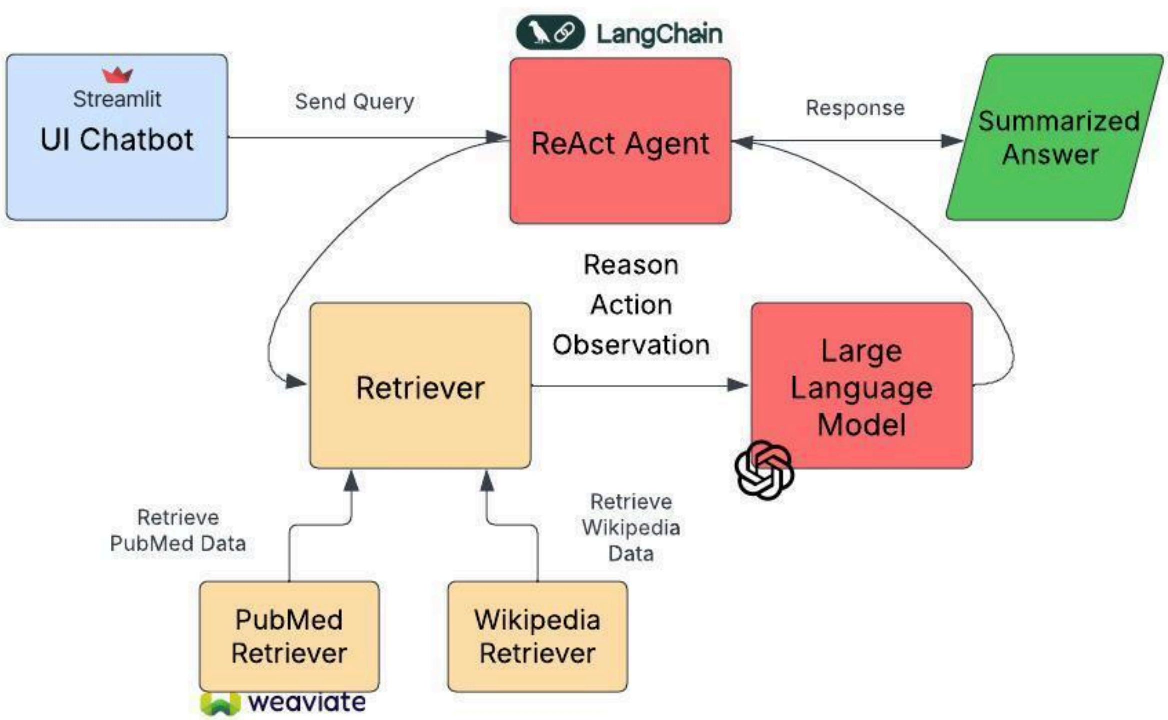
System Architecture



Data Preprocessing

The data preprocessing pipeline illustrated in the above diagram consists of three sequential stages. Initially, PubMed XML Files serve as the primary data source, containing structured biomedical literature and research information. These XML files are then parsed and chunked to systematically extract, transform and segment into appropriate units for efficient processing. Finally, the processed data is stored in a Weaviate database, a vector database optimized for semantic search and AI applications, enabling effective retrieval and utilization of the biomedical information. This streamlined workflow ensures that raw scientific literature is converted into a format suitable for advanced analysis and knowledge extraction.

Figure 1: RAG workflow integrating a ReAct Agent for dynamic reasoning, retrieval, and generation of context-rich answers.



RAG Workflow

The RAG (Retrieval-Augmented Generation) workflow diagram introduces an enhanced system that enables more dynamic information processing through reasoning and decision-making steps. The process is initiated when users send queries via the Streamlit UI Chatbot (blue box). These queries are received by the ReAct Agent (red box), which orchestrates the flow by performing Reasoning, selecting appropriate Actions, and observing Outcomes throughout the retrieval and generation phases.

The ReAct Agent determines whether information needs to be retrieved and interacts with the central Retriever (orange box). The Retriever aggregates information from two specialized retrieval sources — the PubMed Retriever and the Wikipedia Retriever (both shown as orange boxes at the bottom) — each accessing domain-specific datasets. Retrieved content is then either reasoned upon further or passed to the Large Language Model (red box), which processes and synthesizes the information into a coherent output.

Finally, the ReAct Agent delivers a Summarized Answer (green box) to the user. This updated architecture empowers the system to generate more context-aware, thorough, and dynamically reasoned responses by integrating information from both medical literature and general knowledge bases.

Design Choices/Tech Stack

This RAG application was developed using the ReAct framework, Langchain, Streamlit, Weaviate, GPT-4o, Llama-3.2 and deployed on Jetstream 2. The detailed reasons for picking the following tools and frameworks is discussed below:

ReAct Framework

The ReAct (Reasoning + Action) framework[1] represents a significant advancement in the development of interactive AI systems. This architecture integrates reasoning and decision-making capabilities with action execution and observation processing to create more effective agent-based solutions. The ReAct framework functions through a recursive cycle of three core components:

Reasoning: The agent analyzes the current situation, including the user's query and available context. It formulates a plan of action based on its understanding of the task and available tools. This step involves critical thinking, decomposition of complex queries, and prioritization of information needs.
Action: Based on its reasoning, the agent executes specific actions using available tools. These actions might include retrieving information from databases, querying external APIs, or processing data. In our implementation, these actions primarily involve querying specialized retrievers for relevant medical information.
Observation: After performing an action, the agent observes and processes the results. It evaluates the information obtained, determines its relevance and quality, and decides whether additional actions are needed or if sufficient information has been gathered to formulate a response.

This recursive cycle continues until the agent determines it has sufficient information to provide a comprehensive answer to the user's query. The strength of the ReAct framework lies in its ability to decompose complex queries into manageable sub-tasks, maintaining a chain of thought that leads to more accurate and contextually relevant responses.

LangChain ReAct Framework Implementation

In our PubMed RAG system, we implemented the ReAct framework using LangChain[2], a comprehensive library designed to develop applications powered by language models. LangChain's implementation of ReAct provides a structured approach to creating agents that can reason about user queries and leverage external tools to retrieve and process information.
The LangChain ReAct framework allows our system to:

Process natural language queries: Convert user input into a structured format that can guide information retrieval.
Plan a sequence of actions: Determine which tools to use and in what order based on the query's requirements.

Execute actions through tool integration: Access specialized retrievers and process their results.
Maintain contextual awareness: Track the state of the conversation and previously retrieved information.
Generate coherent responses: Synthesize information from multiple sources into a comprehensive answer.

Our implementation integrates two primary retrieval tools:

PubMed Retriever: This specialized tool conducts semantic searches within our Weaviate vector database containing embeddings of medical research papers. It retrieves the most relevant chunks of information based on semantic similarity to the query, ensuring that responses are grounded in peer-reviewed medical literature.
Wikipedia Retriever: This tool supplements the medical literature with general knowledge from Wikipedia, providing broader context when necessary. This is particularly useful for queries that benefit from background information not typically found in specialized research papers.
The LangChain ReAct agent coordinates between these retrievers, deciding which to query based on the nature of the user's question. It can also choose to use both retrievers in sequence, combining specialized medical knowledge with general contextual information to provide more comprehensive responses.

Weaviate

Weaviate is an open-source vector database designed to store, manage, and retrieve high-dimensional vector embeddings efficiently[3]. It is purpose-built for applications involving machine learning and artificial intelligence, where embeddings generated from text, images, or other data types need to be stored and queried for similarity-based retrieval. Weaviate supports advanced features such as hybrid search (combining vector and keyword search), real-time updates, and scalability, making it an ideal choice for projects requiring fast and accurate information retrieval.

In this project, Weaviate was chosen as the vector database to store the embeddings generated from the research paper chunks using the S-PubMedBERT-MS-MARCO embedding generator. Its ability to handle large-scale datasets, combined with its optimized vector search capabilities, ensures that the system can retrieve relevant embeddings quickly and accurately. Weaviate's support for approximate nearest neighbor (ANN) search algorithms, such as HNSW (Hierarchical Navigable Small World), significantly enhances the speed and efficiency of similarity-based queries, which is critical for real-time applications like the RAG agent.

Additionally, Weaviate's schema flexibility and integration capabilities with modern AI frameworks, such as LangChain, make it a seamless fit for this project. By storing embeddings in Weaviate, the system can efficiently retrieve the most relevant chunks of information based on user queries, enabling the large language model (LLM) to generate accurate and contextually relevant responses. This streamlined retrieval process not only improves the performance of the RAG agent but also ensures scalability and reliability, making Weaviate a cornerstone of the system's architecture.

LLMs

For our PubMed RAG system, we experimented with two leading large language models to determine which would provide the best performance for medical information retrieval and synthesis:

Llama 3.2 11B Vision Instruct

Llama 3.2 11B Vision Instruct[4] was initially considered as an open-source alternative for our system. Its advantages included:

- **Open-source nature:** Providing greater flexibility for customization and deployment
- **Local deployment capabilities:** Reducing dependency on external API services
- **Multimodal capabilities:** Offering potential for future expansion to incorporate medical imaging

However, our testing revealed several limitations:

- **Inconsistent response quality:** The model often struggled to provide concise, focused answers to medical queries
- **Inefficient reasoning process:** Even when maxing out the iteration limit (25 iterations in LangChain's ReAct implementation), the model frequently failed to converge on accurate answers

- **Source grounding issues:** We observed that responses tended to be disproportionately grounded in Wikipedia information rather than the more authoritative PubMed sources
- **Citation problems:** The model rarely provided proper citations to the source material, reducing the traceability and credibility of the information
- **Resource intensity:** The model's inefficient reasoning process led to higher computational demands and longer response times

GPT-4o

After identifying the limitations of Llama 3.2, we transitioned to testing GPT-4o[5], which demonstrated significant improvements:

- **Superior response quality:** Generated more accurate, concise, and contextually relevant answers to medical queries
- **Efficient reasoning process:** Typically resolved queries in 1-3 iterations, dramatically reducing computational overhead
- **Balanced source utilization:** Appropriately prioritize information from PubMed when answering specialized medical questions
- **Consistent citation practices:** Regularly included citations to source materials, enhancing the credibility and traceability of information
- **Optimal token utilization:** More efficiently managed the available context window, allowing for more complex queries

While GPT-4o requires API access and has associated costs, the substantial improvements in response quality, reasoning efficiency, and appropriate source utilization justified its selection as the primary LLM for our production system.

Jetstream 2(JS2)

Jetstream 2 is an accessible cloud computing infrastructure for research and education communities. Jetstream 2 was used in this project since it provides on-demand, user-friendly computing resources specifically designed to support a wide range of compute intensive tasks. In our implementation, Jetstream 2[6] instances serve as the robust hosting environment for our Streamlit-based RAG application, offering the computational power and scalability necessary for processing biomedical literature queries efficiently. In addition to this, the GPUs provided helped us in speeding up the process of vector embedding creation of the XML files that were later stored in the Weaviate vector database. The platform's virtual machines provided us with flexible and configurable resources that can be tweaked to specific workload requirements, eliminating the need for specialized hardware investments. Jetstream 2's integration with the ACCESS program (Advanced Computing Coordination Ecosystem: Services & Support) helped us leverage high-performance computing capabilities such as CPU, GPU and data volumes using a simple interface. Additionally, Jetstream 2's focus on reproducibility and collaboration aligns perfectly with our software needs, enabling us to share environments, methodologies, and results seamlessly while maintaining the performance and rapid application deployment.

Streamlit

Streamlit was used in this project since it is a powerful open-source Python framework designed specifically for creating and deploying data applications with minimal effort[7]. Its intuitive API allows developers to transform data scripts into shareable web applications using pure Python, eliminating the need for front-end development expertise. In our biomedical RAG implementation, Streamlit provides an ideal interface solution by enabling rapid development of an interactive chatbot that connects users directly with complex retrieval-augmented generation capabilities. The framework's widget system seamlessly handles user queries through text inputs, while its flexible display options effectively present the retrieved biomedical information and generated responses in a clean, organized format. Streamlit's stateful nature maintains conversation context throughout user sessions, essential for meaningful dialogue with our LLM-powered system. Additionally, its built-in caching mechanisms optimize performance when processing repeated biomedical queries, reducing computational overhead. In addition to this, Streamlit's lightweight deployment requirements align perfectly with Jetstream 2's infrastructure, creating an efficient, accessible platform for researchers to interact with complex biomedical literature through natural language queries without navigating complicated interfaces. In this project, the Streamlit application has a dropdown to select between gpt-4o and Llama-3.2 model, a user input field to enter questions and an indicator showing the status of the RAG application.

Challenges and Mitigation Strategies

Throughout the development of our PubMed RAG system, we encountered several significant challenges that required innovative solutions:

Challenge 1: Recursion Limits and Token Constraints

Problem: When using Llama 3.2 Vision Instruct, we frequently encountered issues with the model reaching the maximum recursion limit (25 iterations) set by LangChain's ReAct implementation. Even after exhausting all available iterations, the model often failed to produce satisfactory answers. This not only led to poor response quality but also increased computational costs and response times.

Mitigation: We discovered that modifying the system prompt with specific instructions about iteration usage yielded surprising improvements. By explicitly instructing the model to limit itself to 5-10 iterations and focus on generating concise answers, we observed more efficient reasoning patterns. This approach reduced unnecessary thinking loops and encouraged the model to converge on answers more quickly. However, this optimization was ultimately insufficient to overcome the fundamental limitations of Llama 3.2 for our specific use case.

Challenge 2: Model Performance and Source Fidelity

Problem: Our evaluation of Llama 3.2 Vision Instruct revealed several critical limitations for medical information retrieval. The model consistently prioritized general knowledge from Wikipedia over the specialized medical information from PubMed, significantly reducing the value of our curated research database. Furthermore, responses lacked the precision and specificity required for medical queries, often providing overly generalized information when detailed clinical insights were needed. The absence of proper citations to source materials undermined the credibility of the information provided, a critical issue in medical contexts where verifiability is essential. From a performance perspective, response generation required excessive iterations, sometimes using the entire allowed recursion limit without producing satisfactory answers, which substantially increased latency and computational costs. Perhaps most concerning for a medical information system, we observed occasional hallucinations when addressing topics where knowledge gaps existed, creating potential risks for users relying on this information.

Mitigation: After comprehensive testing across multiple medical query types, we transitioned to GPT-4o, which demonstrated transformative improvements in system performance. The model appropriately prioritized information from PubMed research papers when answering specialized medical queries, effectively leveraging our vector database of peer-reviewed literature. Its responses demonstrated consistently higher accuracy and relevance, addressing the specific nuances of complex medical questions rather than providing generic information. GPT-4o regularly included citations to specific papers, enhancing the traceability and credibility of the information provided. From an efficiency standpoint, the model typically resolved queries in a single iteration, dramatically reducing response times and computational overhead compared to Llama 3.2. Perhaps most importantly for a medical information system, GPT-4o demonstrated better awareness of knowledge boundaries, reducing the risk of hallucination and clearly indicating uncertainty when appropriate. While this solution increased our API costs, the substantial improvements in response quality and system efficiency fully justified the investment for a system handling potentially critical medical information.

Challenge 3: Streaming and Response Formatting

Problem: The raw output from LangChain's ReAct agent presents significant usability challenges for end users. The default format includes the entire reasoning trace (Agent > Tool > Observation cycles), creating a cluttered and confusing user experience. This technical format exposes the internal dialogue of the agent as it reasons through problems, which, while valuable for debugging and development, overwhelms users with implementation details rather than focusing on the requested information. Medical professionals and researchers expect concise, well-structured responses that clearly communicate findings and their sources, not a verbose transcript of an AI system's thought process.

Mitigation: We addressed this challenge by developing a custom response parser that transforms the raw ReAct output into a user-friendly format optimized for medical information retrieval. Our solution extracts the final synthesized answer from the agent's reasoning trace while also identifying and highlighting the specific tools used and queries executed. This maintains transparency about the information sources without overwhelming the user with technical details. The parser formats research findings in a clean, readable structure and preserves citations while integrating them naturally into the response, enhancing the scholarly value of the system. Additionally, we implemented a progressive, streaming format that provides immediate feedback as information is processed, creating a more responsive user experience that acknowledges the time-sensitive nature of many medical queries.

This custom formatting layer significantly improved user satisfaction by hiding the technical complexity of the ReAct framework while preserving the value of its multi-step reasoning process. Users now receive clearly structured responses that highlight key findings from medical literature while maintaining full transparency about information sources. By delivering a streamlined, intuitive interface that meets the expectations of medical professionals and researchers, our system bridges the gap between sophisticated AI reasoning techniques and practical clinical information needs.

Dataset

Our PubMed Agentic Retrieval-Augmented Generation system processed a substantial corpus of biomedical literature. The key dataset metrics include:

Metric	Value
Number of papers	18,015 NXML files from PubMed Central
Average Length in Tokens	Approximately 4,571 tokens per paper
Number of Chunks	235,161 text segments across all documents
Average Chunks per Document	13.1 chunks per paper

The preprocessing pipeline consisted of multiple steps including XML document intake using lxml, metadata extraction through XPath queries, and semantic text chunking with a target size of 300-400 tokens and 10-20% overlap. This chunking strategy was specifically designed to balance context preservation with retrieval precision.

Each chunk maintains comprehensive metadata extracted from the original articles, including:

- PubMed ID (PMID)
- Article title
- Journal name
- Publication date
- Author information
- Section information (e.g., Abstract, Results, Discussion)
- MeSH terms (Medical Subject Headings)

For vector representations, we utilized PubMedBERT (pritamdeka/S-PubMedBERT-MS-MARCO), a model specifically pre-trained on biomedical literature, which captures domain-specific semantic relationships and medical terminology with high fidelity. This specialized model significantly outperforms general-purpose models on biomedical retrieval tasks.

All chunks and their associated metadata were indexed in Weaviate, enabling semantic search across the entire corpus of biomedical literature.

Examples

Screenshots

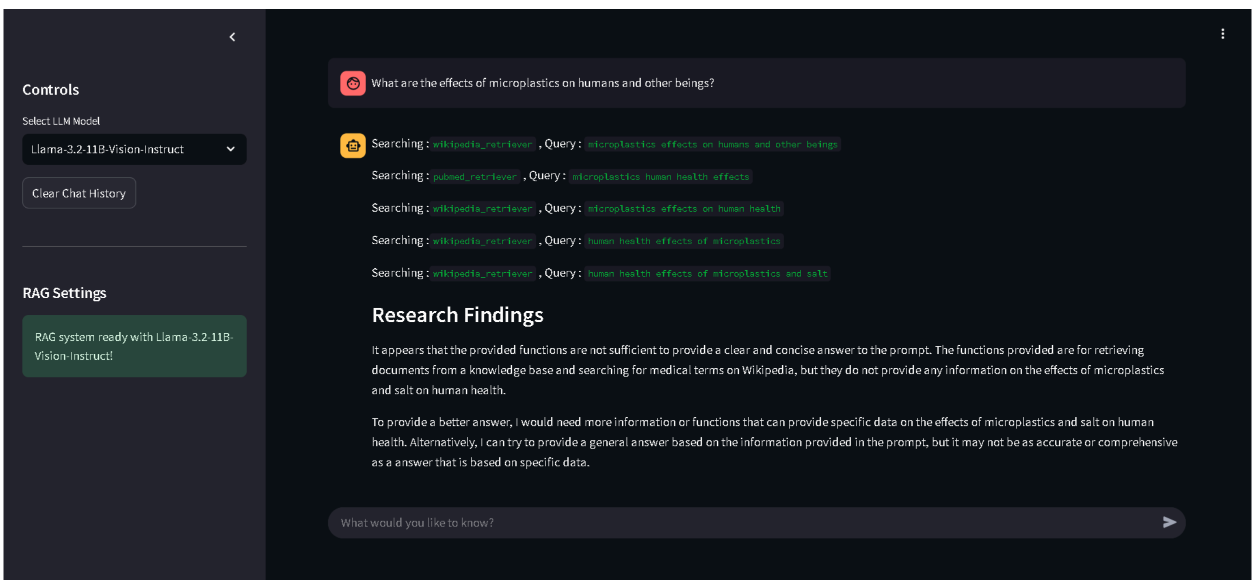


Figure 2: PubMed RAG Chatbot using Llama 3.2 Vision Instruct as the underlying LLM. Despite utilizing 5 iterations of the ReAct framework, the system struggles to formulate a coherent response. The output lacks clarity and proper grounding in PubMed research, illustrating the limitations of Llama 3.2 for specialized medical information retrieval tasks.

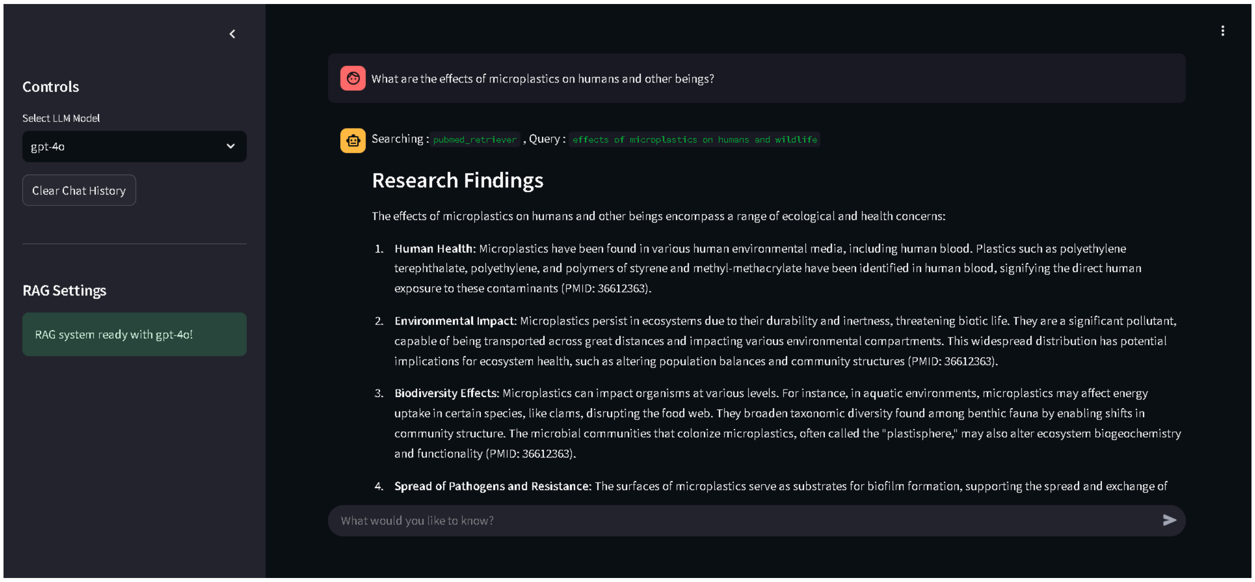
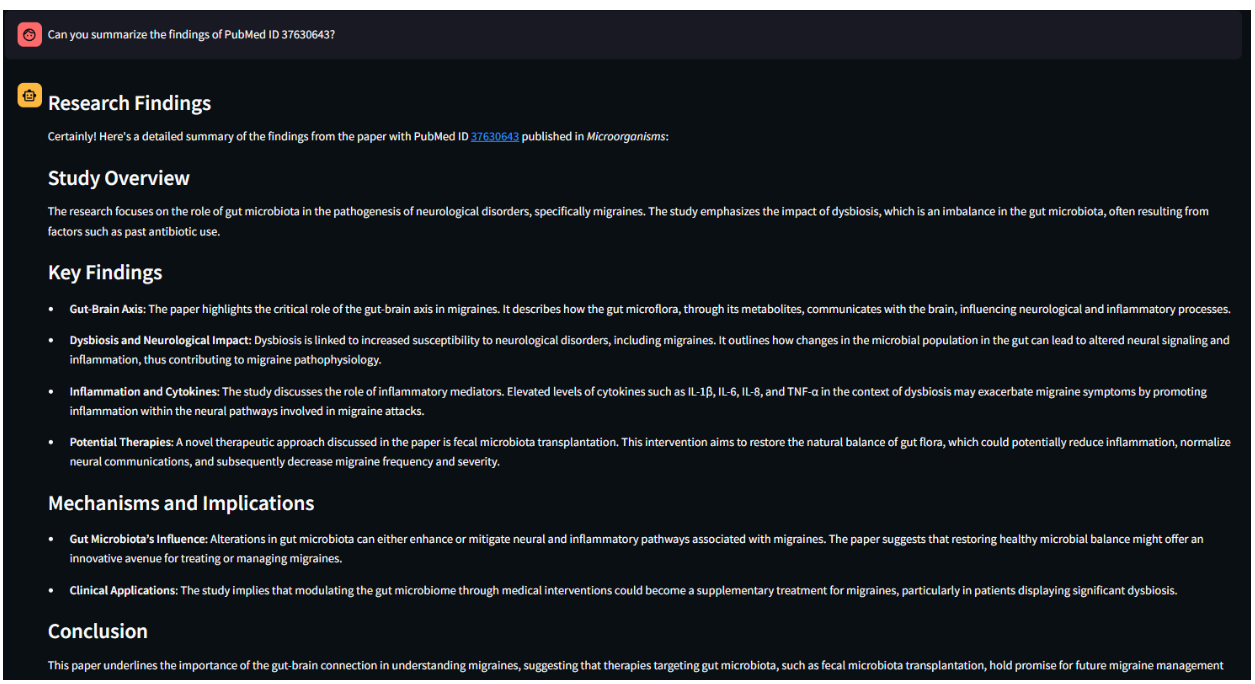


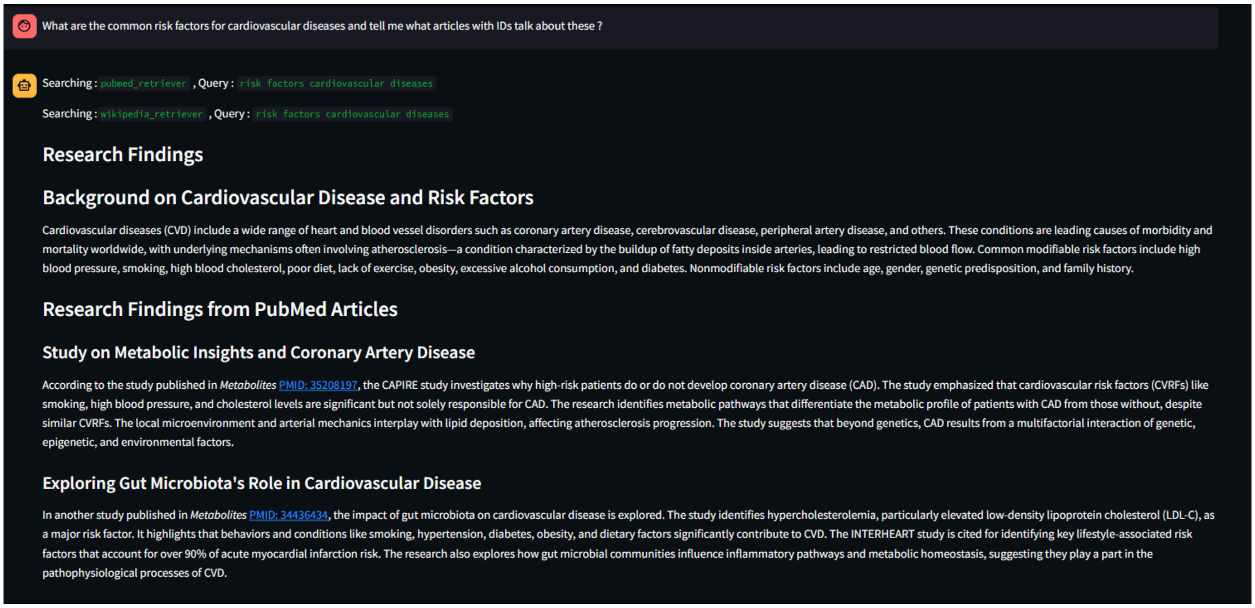
Figure 3: PubMed RAG Chatbot using GPT-4o as the underlying LLM. The system efficiently resolves the medical query in a single iteration, providing a concise response grounded in PubMed literature with proper PMID citations. This demonstrates GPT-4o's superior capability in medical information retrieval and synthesis.

Cherry Picked Examples:

1. Can you summarize the findings of PubMed ID 37630643? (Or any other PMID)



2. What are the common risk factors for cardiovascular diseases and tell me what articles with IDs talk about these ?



Conclusion:

The PubMed Agentic Retrieval-Augmented Generation (RAG) project represents a significant advancement in the field of medical information retrieval and synthesis. By leveraging state-of-the-art technologies such as large language models (LLMs), vector databases, and the ReAct framework, the system provides a robust solution to the challenges posed by the ever-growing volume of medical research publications.

By integrating tools such as LangChain, Weaviate, and GPT-4o, the system provides an efficient and accurate platform for querying biomedical literature. The project processed over 18,000 PubMed NXML files, transforming them into a structured and searchable format, enabling researchers and clinicians to access contextually relevant and evidence-based information with ease. The use of specialized embeddings, such as S-PubMed BERT-MS-MARCO, ensured high fidelity in retrieval tasks, while the deployment of the system as a Streamlit-based chatbot offered an intuitive interface for end-users. This project not only addresses the challenges of navigating vast repositories of medical research but also enhances the accessibility and usability of critical medical knowledge. The project has practical implications for a wide range of use cases. For medical researchers, it offers a streamlined way to explore vast repositories of biomedical literature, accelerating the pace of discovery and innovation. For clinicians, the system provides real-time access to accurate, evidence-based information, enhancing their decision-making capabilities and improving patient care outcomes. Additionally, the system's ability to generate contextually relevant and well-cited responses ensures that it can serve as a reliable tool for knowledge dissemination in academic and clinical settings. By integrating advanced AI technologies with medical research, this project not only addresses the limitations of traditional information retrieval methods but also sets a new benchmark for the application of AI in specialized domains like medicine.

Future Scope:

While the PubMed RAG project has demonstrated remarkable capabilities, there are several avenues for future improvement and expansion to create a more efficient and world-class system:

1. **Multimodal Capabilities:** Incorporating the ability to process and analyze images, diagrams, and other non-textual data would significantly enhance the system's utility, particularly for medical imaging and graphical abstracts.
2. **Scaling the Database:** Expanding the database to include a larger corpus of medical literature and integrating additional sources such as clinical trial data, guidelines, and patient records would improve the comprehensiveness of the system.
3. **Enhancing Retrieval Speed:** Optimizing the retrieval algorithms and leveraging more efficient indexing techniques can reduce latency, ensuring real-time responses even for complex queries.
4. **Integration of Advanced AI Models:** Incorporating cutting-edge AI models with improved reasoning and contextual understanding capabilities can further enhance the accuracy and relevance of the system's responses.
5. **User-Centric Features:** Developing a more intuitive and customizable user interface, along with features like personalized recommendations and query history, can improve user experience and engagement.
6. **Cross-Domain Applications:** Adapting the system for use in other specialized domains, such as legal research or engineering, can broaden its applicability and impact.

By addressing these areas, the PubMed RAG system can evolve into a more versatile, efficient, and impactful tool, setting new standards for AI-driven information retrieval and synthesis in the medical field and beyond.

References:

1. S. Yao *et al.*, "REACT: Synergizing reasoning and acting in language models," *arXiv (Cornell University)*, Jan. 2022, doi: 10.48550/arxiv.2210.03629.
2. "Introduction | 🔗 LangChain." https://python.langchain.com/docs/introduction/
3. "Home | Weaviate." https://weaviate.io/developers/weaviate
4. "meta-llama/Llama-3.2-11B-Vision-Instruct · Hugging Face," Dec. 06, 2024. https://huggingface.co/meta-llama/Llama-3.2-11B-Vision-Instruct
5. "Model - OpenAI API." https://platform.openai.com/docs/models/gpt-4o
6. "JetStream2 Documentation." https://docs.jetstream-cloud.org/
7. "Streamlit Docs," Sep. 10, 2024. https://docs.streamlit.io/