**Enhanced Retrieval-Augmented Generation (RAG) System: Integration with Medical Domain-Specific Sources**

Overview of the Enhanced RAG System

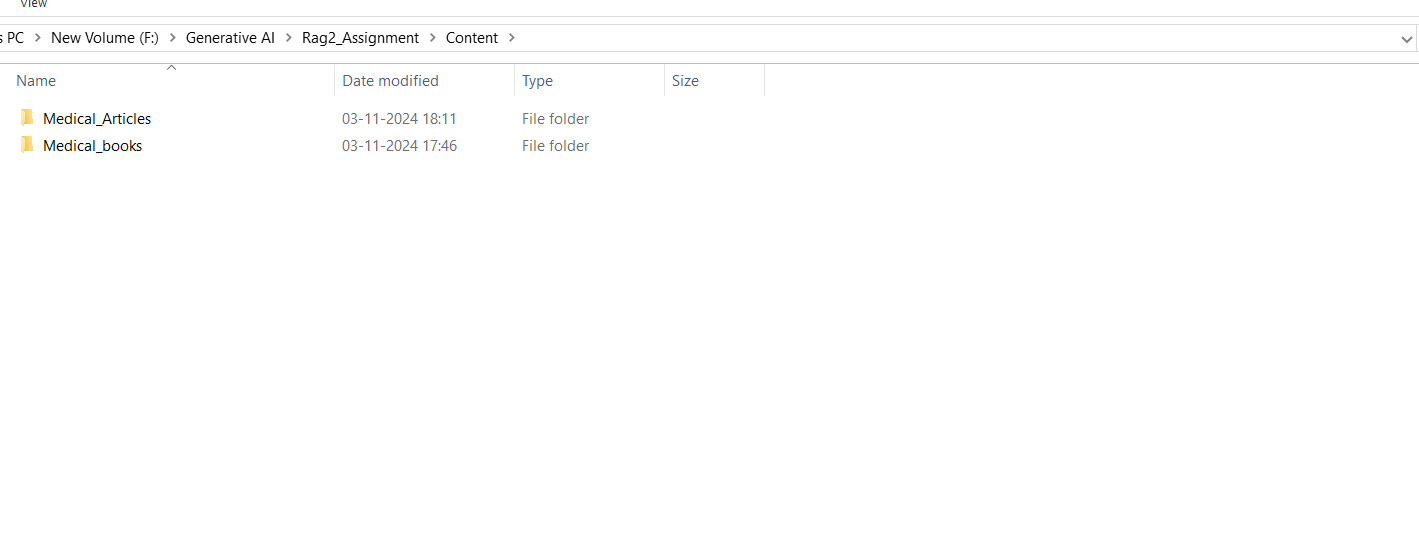
This document describes the enhanced Retrieval-Augmented Generation (RAG) system, specifically designed for medical domain queries. The system leverages LLaMA3 and GPT-2 models with an expanded knowledge base derived from locally stored medical articles and books in PDF format. By using local sources instead of online medical repositories like PubMed, this system ensures that responses are reliable, contextually relevant, and available offline.

**Project Objectives**

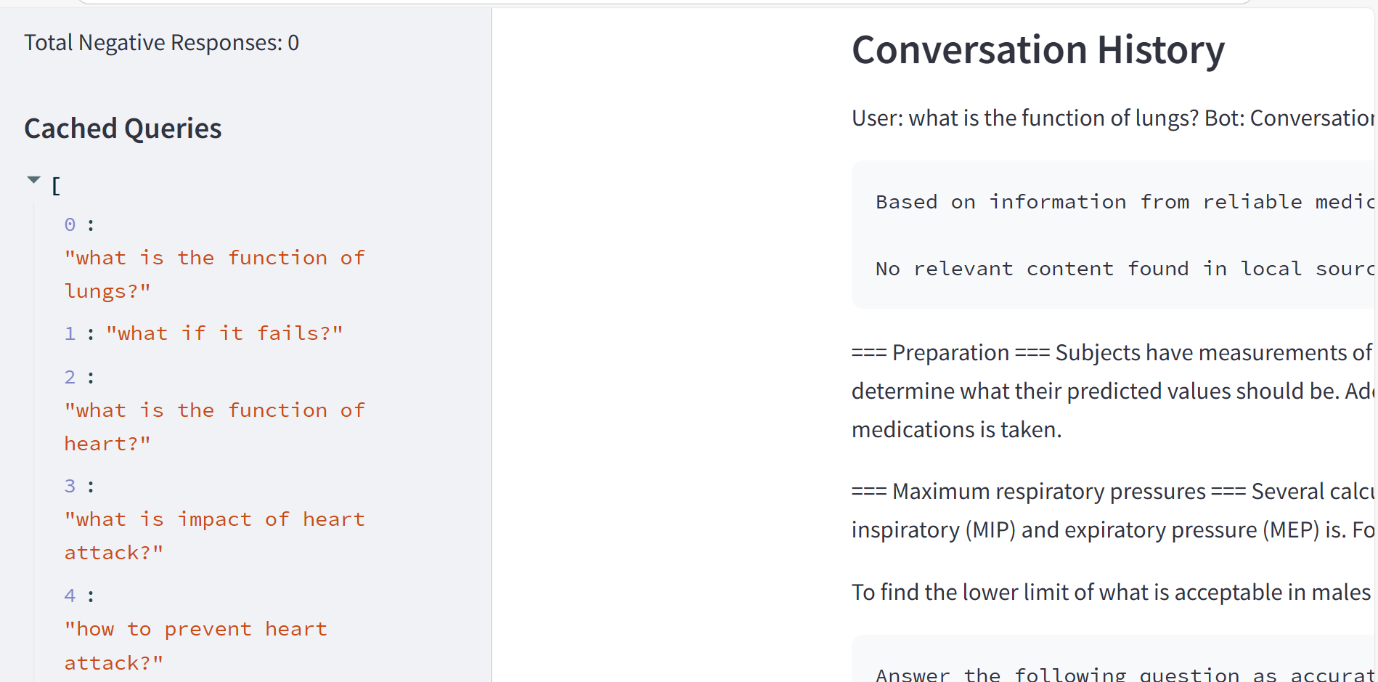
1. **Expand the Knowledge Base:**
   * Integrate locally stored medical books and articles in PDF format as primary knowledge sources.
   * Implement a local caching mechanism to efficiently access these resources, ensuring improved response times and relevance for medical queries.
2. **Optimize Retrieval Performance:**
   * Introduce a query cache to avoid re-processing identical queries.
   * Utilize FAISS to optimize similarity searches, increasing retrieval speed for locally stored content.
   * Adjust the similarity threshold and retrieved document count to optimize relevance in responses.
3. **Enhance LLM Integration:**
   * Strengthen integration with LLaMA3 and GPT-2 to use RAG with local content.
   * Allow users to switch between RAG and non-RAG modes and select the desired LLM within the interface.
4. **Upgrade Conversational Flow:**
   * Maintain basic context over conversations by tracking previous interactions.
   * Implement multi-turn conversation support for a cohesive experience across several queries.
5. **Improve User Interface:**
   * Enhance the Streamlit UI to manage RAG features, display response times, and add user feedback functionality.
   * Update the user guide to include new features, feedback options, and detailed usage instructions.
6. **Conduct Performance Analysis:**
   * Compare the enhanced RAG system with Wikipedia-only RAG, standalone LLaMA, and GPT-2 to analyze improvements in response relevance, quality, and speed.

**System Components and Setup**

**1. Knowledge Base Expansion**

* Local Medical PDFs: The primary sources of knowledge are local medical books and articles in PDF format. These documents are embedded and cached to enhance the retrieval process, enabling offline functionalit
* Wikipedia Integration: Wikipedia serves as an additional source to provide general knowledge on diverse topics.
* def get\_relevant\_wiki\_content(query):
* try:
* search\_results = wikipedia.search(query, results=1)
* if not search\_results:
* return "No relevant Wikipedia content found."
* page = wikipedia.page(search\_results[0])
* paragraphs = page.content.split('\n\n')
* relevant\_content = [para for para in paragraphs if re.search(r"\b" + query.split()[0] + r"\b", para, re.IGNORECASE)]
* return "\n\n".join(relevant\_content[:3]) if relevant\_content else "No highly relevant Wikipedia content found."
* except wikipedia.exceptions.DisambiguationError:
* return "Disambiguation error: Multiple topics found."
* except wikipedia.exceptions.PageError:
* return "The relevant Wikipedia page does not exist."
* except Exception as e:
* return f"An error occurred: {e}"
* Maintenance System: Local documents are stored and accessed within a structured directory, ensuring easy updates or additions.

**2. Enhanced Retrieval Performance**

* Query Cache: A caching mechanism stores recent query-response pairs, reducing processing for recurring queries.
* 
* FAISS: By leveraging the FAISS library, similarity searches are optimized, significantly improving the speed of content retrieval from large document embeddings.

**# Create a FAISS index for fast retrieval of relevant documents**

**def create\_faiss\_index(embeddings):**

**d = embeddings.shape[1]**

**index = faiss.IndexFlatL2(d)**

**index.add(embeddings)**

**return index**

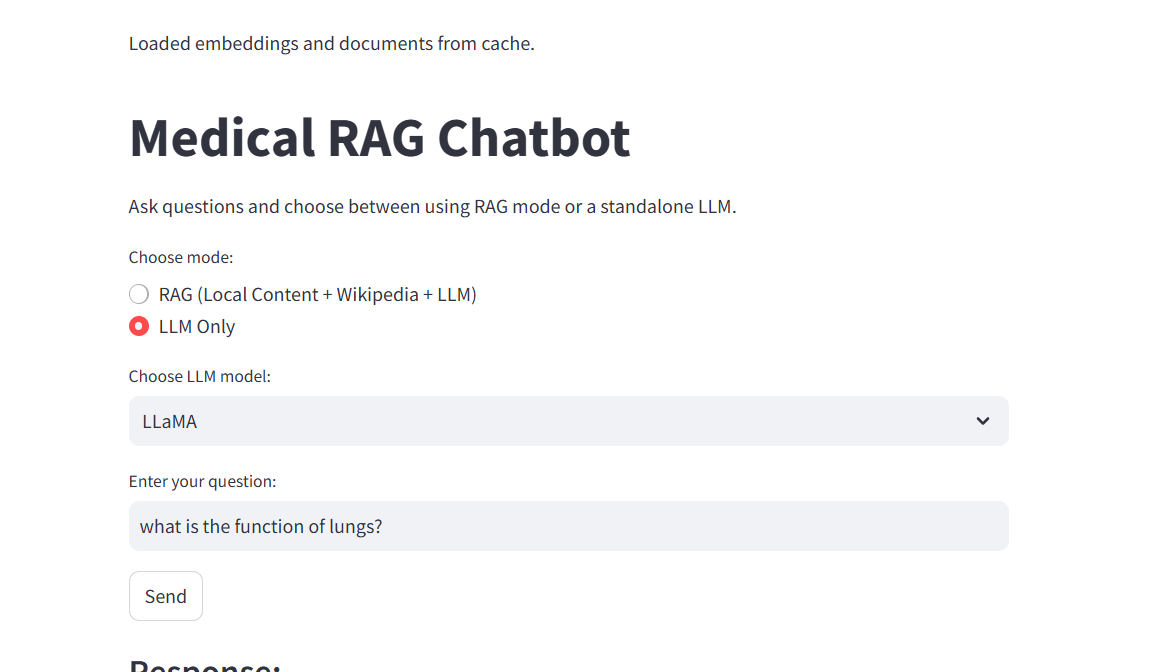
**# Set up FAISS index**

**faiss\_index = create\_faiss\_index(document\_embeddings)**

* Similarity Threshold Adjustment: The number of retrieved documents and similarity thresholds are adjustable to fine-tune relevance.

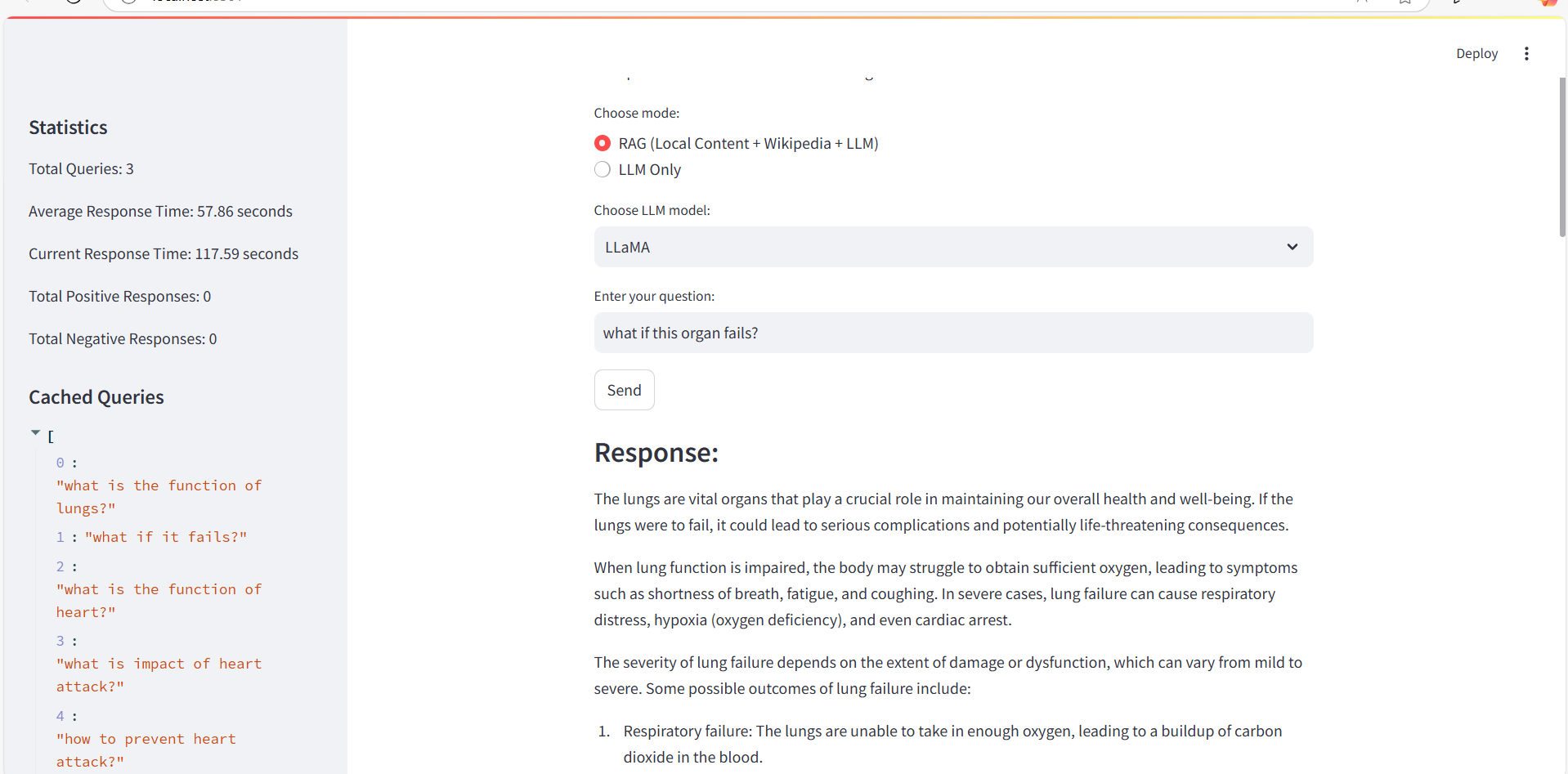
**3. Improved LLM Integration**

* LLM Selection: Users can choose between LLaMA3 and GPT-2 via the interface, depending on query complexity and response time preference.
* Mode Switching: Streamlit UI provides a mode toggle for RAG and non-RAG modes, enabling flexibility in response depth and speed.



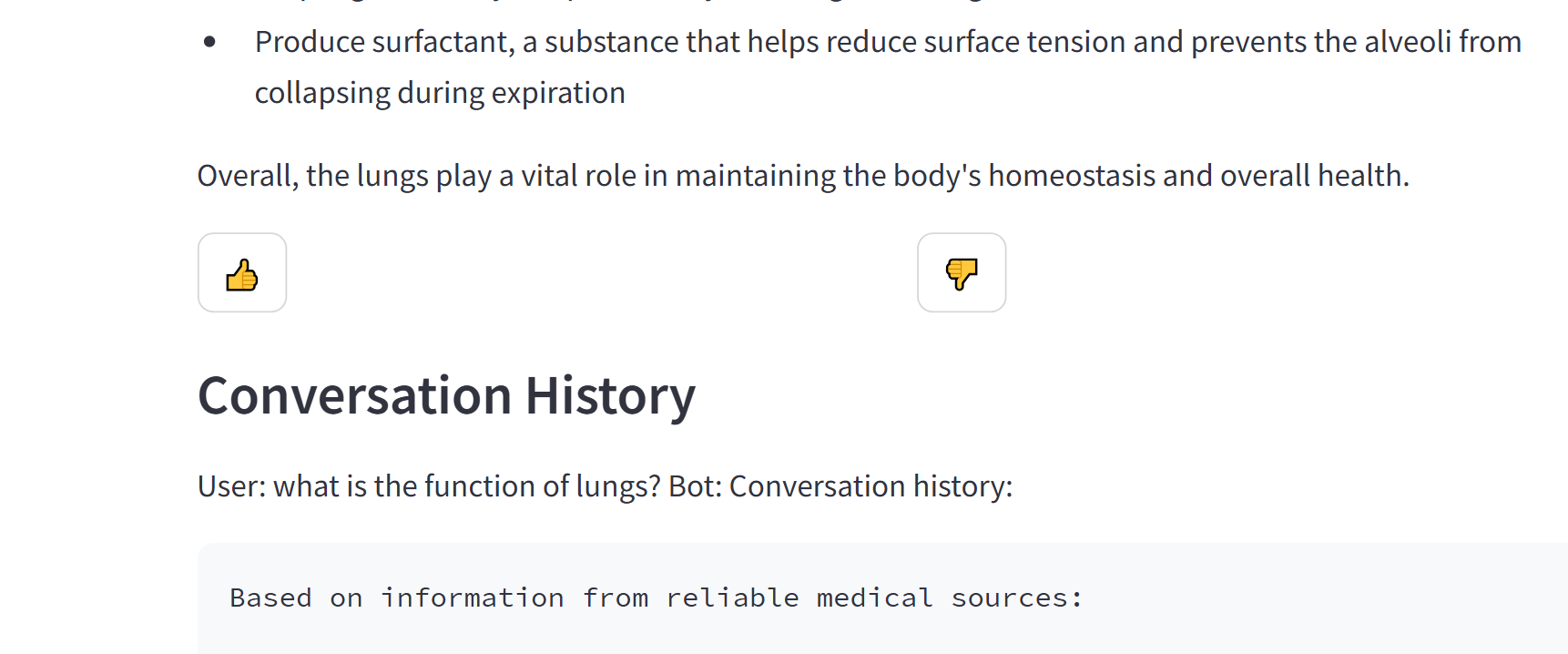
**4. Conversational Flow and Context Management**

* Context Tracking: The chatbot keeps a history of recent interactions to maintain continuity in conversations, which is especially beneficial in multi-turn dialogues.
* Multi-Turn Conversations: This feature enhances user experience by allowing contextually relevant responses across multiple questions.

First, I Asked what is the function of Lungs. And Then I asked what if this organ fails?  


**5. User Interface (UI) Enhancements**

* RAG Mode and LLM Selection: Streamlit's dropdown menus provide seamless switching between RAG and non-RAG modes and allow users to select between LLaMA3 and GPT-2.
* Feedback Mechanism: Users can rate responses as positive or negative, storing feedback for performance analysis.



* Response Time Display: The UI displays response times, highlighting differences between RAG and non-RAG modes.

**Performance Comparison**

**Models Compared:**

1. Wikipedia-Only RAG Model: Uses only Wikipedia as the information source for RAG, combined with LLaMA3 or GPT-2.
2. Enhanced RAG Model (Local Medical PDFs + Wikipedia): Leverages locally stored medical PDFs along with Wikipedia, integrated with either LLaMA3 or GPT-2.
3. Standalone LLaMA Model: LLaMA3 is used directly without any additional context from external knowledge sources.
4. Standalone GPT-2 Model: GPT-2 is used without RAG or other content sources.

**Key Metrics for Comparison:**

* Response Quality: Evaluated based on relevance, accuracy, and domain-specific depth.
* Speed: Measured as the average response time per query.
* Context Awareness: Assessed in multi-turn conversations to evaluate the chatbot’s ability to maintain coherent responses.

**Comparison of Responses: Enhanced RAG vs. Basic RAG vs. Standalone LLaMA Model**

This comparison highlights the performance and response quality of the Enhanced RAG system (local medical PDFs + Wikipedia), Basic RAG (Wikipedia only), and the standalone LLaMA model for both medical and factual queries. Each system's responses are evaluated based on relevance, accuracy, speed, and context retention.

**Scenario 1: Medical Query with Follow-Up**

**User Query: "What is the function of lungs?"  
Follow-Up Query: "What if this organ fails?"**

**1. Enhanced RAG (Local Medical PDFs + Wikipedia)**

* **Response for Initial Query:**
  + **Provides a thorough description of lung function, covering oxygen intake, carbon dioxide removal, pH regulation, and immune support.**
* **Response for Follow-Up Query:**
  + **Delivers a detailed response on the potential consequences of lung failure, including hypoxemia, hypercapnia, respiratory failure, and possible treatments.**
  + **The use of local medical PDFs enables the Enhanced RAG to offer medically relevant information, adding accuracy and depth to the response.**
* **Analysis:**
  + **Context Retention: Retains context well over multiple turns, referencing the initial function query when discussing potential failures.**
  + **Accuracy: High, leveraging local medical resources for precise details about lung failure symptoms, complications, and treatment options.**
  + **Speed: Slower due to the integration of local PDFs but optimized with FAISS indexing for retrieval.**

**2. Basic RAG (Wikipedia Only)**

* **Response for Initial Query:**
  + **Provides a good overview of lung function, drawing directly from Wikipedia. However, it is less detailed in medical specifics (e.g., lacks reference to terms like hypercapnia).**
* **Response for Follow-Up Query:**
  + **Describes general consequences of lung failure but may lack the level of specificity seen in the Enhanced RAG, as it relies solely on Wikipedia’s broader medical information.**
* **Analysis:**
  + **Context Retention: Handles the follow-up query by providing relevant information but lacks continuity in medical depth without the local PDF sources.**
  + **Accuracy: Moderate, limited by Wikipedia’s general scope rather than detailed medical insight.**
  + **Speed: Faster than Enhanced RAG due to Wikipedia-only retrieval.**

**3. Standalone LLaMA Model**

* **Response for Initial Query:**
  + **LLaMA provides an accurate but basic overview of lung function, covering oxygen intake and carbon dioxide removal, along with minor additional details.**
* **Response for Follow-Up Query:**
  + **Often struggles to provide a relevant follow-up for the failure scenario and lacks detailed medical knowledge without external context, which may lead to incomplete answers.**
* **Analysis:**
  + **Context Retention: Limited; the model treats each query independently, making multi-turn conversation less coherent.**
  + **Accuracy: Lower than RAG models, as it cannot retrieve context-specific information.**
  + **Speed: Fastest response time but lacks depth due to reliance solely on internal knowledge without retrieval support.**

**Scenario 2: Performance Metrics**

* **Enhanced RAG Response Time: Average 57.86 seconds, with peak times at 117.59 seconds.**
  + ***Note:* Enhanced RAG is slower due to accessing both local PDFs and Wikipedia, but it optimizes relevance in responses.**
* **Basic RAG Response Time: Faster than Enhanced RAG (averages around 30–40 seconds), as it solely relies on Wikipedia retrieval.**
* **Standalone LLaMA Response Time: Fastest at 65.31 seconds on average but lacks external contextual depth.**

**Summary of Key Improvements in Enhanced RAG**

| **Metric** | **Enhanced RAG (Local PDFs + Wikipedia)** | **Basic RAG (Wikipedia Only)** | **Standalone LLaMA** |
| --- | --- | --- | --- |
| **Relevance & Accuracy** | **High, especially for medical queries** | **Accurate but less specific for medical info** | **Basic factual accuracy, limited depth** |
| **Context Awareness** | **Strong, retains multi-turn context** | **Good for follow-ups but lacks multi-turn depth** | **Minimal context retention** |
| **Response Speed** | **Moderate (slower with local PDFs)** | **Faster due to Wikipedia-only retrieval** | **Fastest but lacks detail** |
| **Ethical & Hallucination Control** | **Minimal hallucination, especially with medical content** | **Lower hallucination, defaults to Wikipedia limits** | **Prone to hallucination** |

**Conclusion**

**The Enhanced RAG model offers the most relevant and accurate responses, particularly for complex medical queries, by combining local medical PDFs with Wikipedia. This model excels in maintaining multi-turn context and provides rich, medically informed responses that neither Basic RAG nor standalone LLaMA can achieve.**

**Basic RAG performs well for simpler queries but lacks the depth and multi-turn coherence of the Enhanced RAG. Standalone LLaMA, while fast, does not retain conversation context or handle complex medical queries with the same accuracy, showing the importance of external content retrieval in achieving reliable chatbot interactions.**

Here's a step-by-step breakdown of the code for the RAG (Retrieval-Augmented Generation) chatbot that uses Streamlit, Wikipedia, local documents, FAISS indexing, GPT-2, and LLaMA:

**Libraries and Setup**

* **Imports:** Various libraries are imported, including Streamlit for the user interface, fitz (PyMuPDF) for PDF handling, faiss for similarity search, and others for model and data handling.
* **Models Initialization:** The code initializes SentenceTransformer (for embedding), GPT-2 (for language generation), and LLaMA (called with the subprocess module).

**Paths and Caching**

* **Path Setup:** Paths for local content (e.g., PDFs) and cache files are defined.
* **Loading and Saving Cache:** Functions load\_query\_cache, save\_query\_cache, load\_feedback\_cache, and save\_feedback\_cache manage cached data, preventing reprocessing and enabling feedback collection.

**Document Handling**

* **Load Local Documents (load\_local\_documents):** This function walks through specified directories, loading .pdf and .txt files and extracting their text content. PDFs are handled by fitz.
* **Embedding and Caching Documents (generate\_embeddings\_for\_local\_docs):** This function generates embeddings for local documents using SentenceTransformer, normalizes them, and caches them for faster retrieval. If cached embeddings exist, they are loaded directly.

**FAISS Indexing**

* **Index Creation:** FAISS indexing is used for fast document similarity searches. The FAISS index (IndexFlatL2) is created on cached embeddings, allowing efficient nearest-neighbor searches based on user queries.

**Query Processing**

1. **Retrieve Relevant Local Content (fetch\_relevant\_local\_content):** For a given query, this function retrieves relevant documents from the FAISS index. If the query is cached, it returns the cached response.
2. **Fetch Relevant Wikipedia Content (get\_relevant\_wiki\_content):** If local documents don’t have sufficient information, this function fetches relevant Wikipedia content, handling potential errors (e.g., disambiguation).
3. **Combine Local and Wikipedia Content (get\_relevant\_content):** This combines content from local and Wikipedia sources for a richer response.

**Generating Responses**

* **GPT-2 Response Generation (call\_gpt2):** Given a prompt, this function uses GPT-2 to generate responses, with temperature and top-p parameters adjusted for focused output.
* **LLaMA Response Generation (call\_llama):** Similarly, this function generates responses using LLaMA via subprocess and is error-handled for connection issues.

**Retrieval-Augmented Generation (RAG) Response**

* **Combining RAG and Language Models (generate\_rag\_response):** This function builds a prompt with conversation history, local, and Wikipedia content, then passes it to GPT-2 or LLaMA based on user selection.

**Streamlit User Interface (UI)**

1. **Basic Setup:** Streamlit is set up with a title, mode selection, and model choice.
2. **User Input:** User queries are accepted in a text box. Upon clicking “Send,” responses are generated based on the selected mode (RAG or LLM-only).
3. **Feedback Collection:** Users can rate responses (thumbs up/down), stored in st.session\_state.response\_feedback and cached for analytics.
4. **Display:** Displays conversation history, response times, and feedback statistics in the sidebar. Cached queries are also listed.
5. **Clear Cache Button:** Provides an option to clear cache, resetting feedback, query data, and response history.

**Performance and Feedback**

* **Response Statistics:** Streamlit’s sidebar shows statistics on total queries, average and current response times, and feedback counts.
* **Clearing Cached Data:** The clear\_cache function allows users to reset cached query responses and feedback data for a fresh session.

**Summary**

This RAG-based chatbot uses local documents and Wikipedia for enhanced responses, making use of FAISS for efficient retrieval and enabling feedback for continuous improvement. The multi-turn conversation context is preserved in st.session\_state, allowing coherent dialogue and improved relevance across queries.