BioAssist: Biomedical Web-Search Enabled RAG Chatbot

1. Project Overview

BioAssist is a domain-specific AI-powered Retrieval-Augmented Generation (RAG) chatbot built to assist users with biomedical, clinical, and healthcare-related queries. It combines knowledge retrieval from internal documents with real-time web search. The chatbot leverages transformer-based sentence embeddings and integrates them into a FAISS-based vector database for efficient similarity search. Its prompt-building strategy incorporates both past conversational history and dual-contextual grounding (local + web), enhancing the relevance and coherence of the generated answers.

---

2. Document Ingestion & Vectorization

- **Supported Formats**: PDF, DOCX, TXT, CSV
  - A total of 17 local documents were ingested in the vector index. 5 PDF, 4 CSV, 4 TXT and 4 DOCX
- **Text Chunking**: Token-based splitter with overlap
- **Embedding Model**: sentence-transformers/all-MiniLM-L6-v2
- **Vector DB**: FAISS

---

3. Architecture Overview

The BioAssist architecture follows a modular design enabling efficient ingestion, retrieval, generation, and interaction. It comprises the following key components:

- **Document Ingestion & Vector Indexing**
  Documents in formats like PDF, DOCX, TXT, and CSV are parsed, chunked using a token-based splitter (TokenTextSplitter), and embedded using HuggingFace's sentence-transformers/all-MiniLM-L6-v2. These embeddings are L2-normalized and stored in a FAISS index using IndexFlatIP for cosine similarity-based retrieval. Metadata is attached to each chunk to enable traceability and quality inspection. The pipeline uses LangChain methods to process data.
- **Retrieval & Query Processing**
  Each user query is first evaluated by a domain-specific guardrail to determine if it falls within the biomedical domain. If it does not, the chatbot responds with a fallback message indicating that the query is outside its scope. If the query is biomedical, it is passed to a query rewriter powered by an LLM, which resolves ambiguities (such as pronouns or incomplete references) by incorporating context from the ongoing conversation. The rewritten query is then processed through a conversational retrieval chain, ensuring that responses are both

context-aware and grounded in relevant biomedical knowledge. The system retrieves the top-5 most relevant document chunks from the FAISS vector store using similarity scores.
    - If the best similarity score is below 0.4, it enters web-fallback mode, where only top-5 web search results are considered.
    - Otherwise, it combines both local document chunks and web results for grounding.
- Web search is executed in every case, ensuring robustness even when local content is available. Retrieval logic is implemented in rag_pipeline.py.
- LangChain is used as a utility layer to format prompts, structure documents, and wrap Gemini model calls.
- **Web Search Integration**
  DuckDuckGo web results are fetched using the ddgs library and converted into Document objects via LangChain's schema. These are included in the final context for answer generation. In fallback mode (when local similarity is < 0.4), only web search results are used to generate the answer.
- **Prompt Construction & LLM Answer Generation**
  The final prompt includes:
    - User query
    - Chat history
    - Retrieved local document chunks
    - Web search snippets
- This is passed to Gemini Flash 2.0 Lite, configured with a temperature of 0.4 and max tokens of 2000. The model generates a grounded, context-aware response that is verified against the retrieved context for factual grounding (Hallucination Detection).

---

4. RAG Pipeline

- **Query Rewriting**: Ensures clarity and self-containment for web search
- **Biomedical Guardrail**: Binary classifier using Gemini to restrict queries to health-related topics
- **Dual Retrieval**:
    - From internal vector DB (using similarity search)
    - From DuckDuckGo web search (via web_search.py)
- **Prompt Construction**:
    - Chat history + Local + Web chunks used
    - Final prompt sent to Gemini model for answer generation
- **Answer Filtering**:
    - Verify grounding of each sentence using self-critique technique. Basically, the generated answer is compared with the provided context.

---

5. Web Search Integration

- **Library Used**: DuckDuckGo (ddgs) via LangChain schema

- **Fallback Trigger**: Local similarity score below threshold (e.g., 0.4)
- **Web Context**: Injected into prompt along with local context
- **Source Tracking**: All URLs and titles retained to be displayed as reference with the provided answer

---

6. Frontend Interface

- **Framework**: Streamlit
- **Key Features**:
    - Realtime input/output
    - Exportable chat history
    - End chat option
    - Source traceability (local vs. web)
    - Visual indicators for fallback cases
    - Chat input with auto-scroll
    - Export chat (.txt, .md)
    - Performance metrics (retrieval time, generation time, token count)
    - Past conversation management
    - Web fallback notice and content reference

---

7. Bonus Features Implemented

- **Use of LangChain**
- **DuckDuckGo-based web search fallback**
- **Query rewrite using chat context**
- **Biomedical safety guardrail**
- **Hallucination Detection**
- **Streamlit UI with export, metrics, chat persistence**
- **Logging through “loguru”**
- **Docker file**

---

8. Design Decisions

- **Gemini Flash 2.0 Lite**: Chosen for context length, speed and output quality.
- **MiniLM Embeddings**: Small, fast, and high-performing for semantic similarity
- **FAISSDB**: Configurable backend for flexibility and persistence
- **DDG Web Search**: Free and fast, avoids dependency on external API keys

---