Research Queries with RAG-Powered Chatbot

Aman, MDS202305
Himanshu, MDS202327
Keshav Kumar, MDS202331
Utpalraj Kemprai, MDS202352

Agenda

- Motivation
- System Architecture
- Technical Implementation
- Key Components
- Data Flow and RAG Pipeline
- What all we tried!
- Challenges and Solutions
- Future Enhancements

START

Motivation

WHAT?

Help users research the latest academic papers and relevant information across domains.

WHY?

Traditional search is slow and fragmented!

NEED?

Interactive assistant combining updated research and general knowledge.

System Architecture

1. **Backend Framework:** Flask
2. **Information Retrieval:** Wikipedia API, ArXiv API, Quadrant (Vector DB)
3. **NLP:** KeyBERT, SentenceTransformers, Transformers (distilbart-cnn-12-6)
4. **LLMs:** Phi-3-mini and Google Gemini API
5. **Frontend:** HTML/CSS/JavaScript
6. **File Processing:** PyMuPDF (fitz)
7. **Data Storage and Processing:** JSON, UUID
8. **Python Libraries:** TQDM, NumPy
9. **Deployment:** Docker

How it all work together

1. **User Query:** The user submits a query via the frontend.
2. **Keyphrase Extraction:** KeyBERT extracts keyphrases from the query.
3. **Content Fetching:** Wikipedia and/or ArXiv content is fetched based on the keyphrases.
4. **Chunking and Embedding:** The content is chunked, embedded using SentenceTransformers, and stored in Qdrant.
5. **Retrieval:** Relevant chunks are retrieved from Qdrant using cosine similarity.
6. **Summarization:** Retrieved chunks are summarized using DistilBART.
7. **Response Generation:** The summarized content is passed to the LLM (Phi-3-mini or Google Gemini) to generate a response.
8. **Frontend Display:** The response, along with citations, is streamed back to the user in real-time.

Key Components

Data Sources

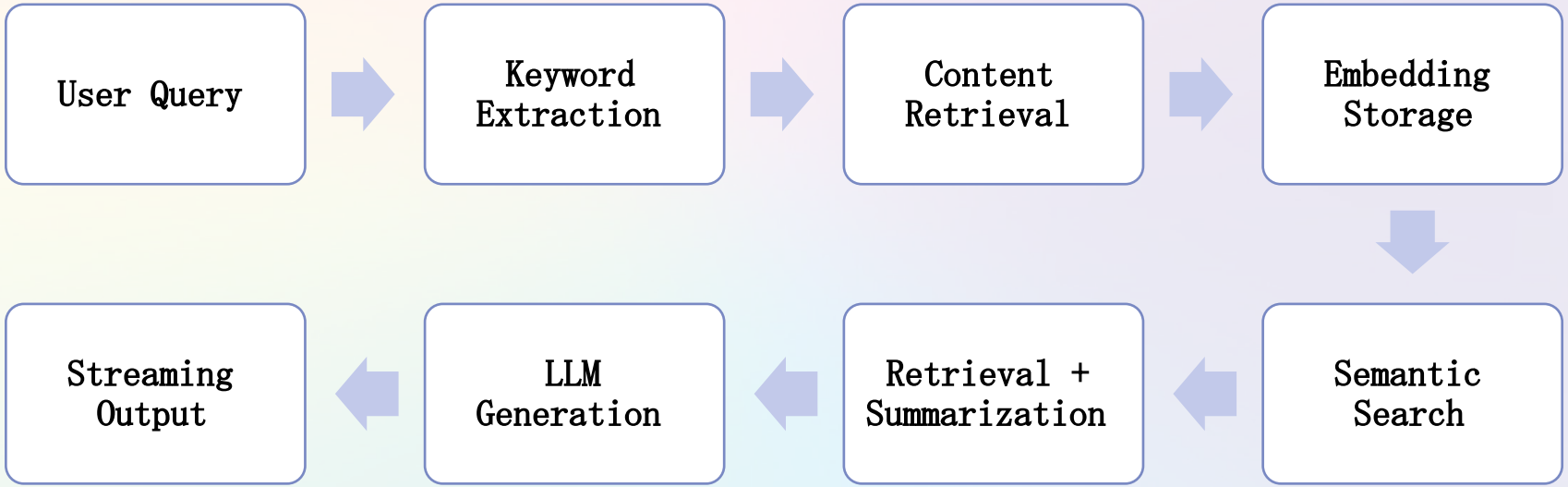
- Wikipedia API for general knowledge retrieval with section chunking
- ArXiv API for accessing latest research papers filtered by subject/subtopic
- File upload capability for custom document context (.pdf, .txt support)

Processing Pipeline

- Keyword extraction using KeyBERT + KeyPhrase Vectorizer for identifying search terms
- Text chunking and storage in VectorDB followed by vector-based semantic search for finding relevant content
- Text summarization to manage token limitations
- Citation tracking to maintain academic integrity

START

Data Flow and RAG Pipeline



START

What all we tried!

- Different LLMs (Phi-3-Mini, Mistral 7B; Gemini-2.0-Flash)
- Different models for keyword/key phrase extraction (Stat: Yake, Rake, TextRank, DL:KeyBERT, KeyPhrase Vectorizers)
- Dump all fetched content for RAG context later used vector db, chunking and text summarization of all the retrieve chunks
- Allowed users to decide whether to use wiki, ArXiv (relevancy, recency or both) with .pdf, .txt file support for more sources of information content for RAG
- Used QDrant as our vector db. For demo we are storing all fetched content in RAM, can be deployed in the cloud too.

Challenges and Solutions

- **PDF Capability Inefficient**
  Not able to extract tables, latex chunks and diagrams
- **Time Latency**
  Downloading the content from ArXiv and Wikipedia
- **Token limitations**
  Solved with chunking and summarizing
- **Information freshness**
  Addressed by integrating ArXiv's recent papers option

Future Enhancements

- Multi-modal support for images and diagrams
- Integration with additional academic databases
- Improved PDF parsing and understanding
- User feedback loop for continuous improvement
- Fine-tuning the language model on academic content

Thank you

Team

Breaking RAG

https://github.com/U1K_emp/RAG-powered-search_-engine-for-Research_

START