

Interactive Research Assistant Using Retrieval-Augmented Generation (RAG)

Development Period: March 2025

Primary Developer: Sukanksha totade(sukankshatotade1@gmail.com)

Environment: Google Colab / Jupyter Notebook

Table of Contents

Introduction.....	4
Project Goals.....	4
System Overview.....	4
Development Process.....	5
5.1 Code Structure Overview.....	5
5.2 PDF Upload and Text Extraction(upload_pdf.py).....	6
UI Component:.....	6
Considerations:.....	6
5.3 Text Chunking and Embedding (embed_text.py).....	7
Functionality:.....	7
UI Component:.....	7
5.4 Similarity Search and Retrieval (chatbot_core.py).....	7
Functionality:.....	7
Design Choices:.....	8
5.5 LLM Integration and Prompt Engineering (chatbot_core.py + load_llm.py).....	8
Prompt Construction:.....	8
LLM Integration:.....	9
5.6 UI and Chat Experience.....	9
Modes:.....	9
UI Elements:.....	9
Chat Handler Logic:.....	9
Evaluation and Testing.....	10
6.1 Functional Testing.....	10
Objective:.....	10
6.2 Example Evaluation Case.....	11
Sample Questions & Answers:.....	11
6.3 Metrics.....	11
Limitations and Challenges.....	11
7.1 Input Limitations.....	11
7.2 Model Constraints.....	12
7.3 Context Size Limit.....	12
7.4 Evaluation Bottlenecks.....	12
Future Improvements.....	12
References.....	13

Introduction

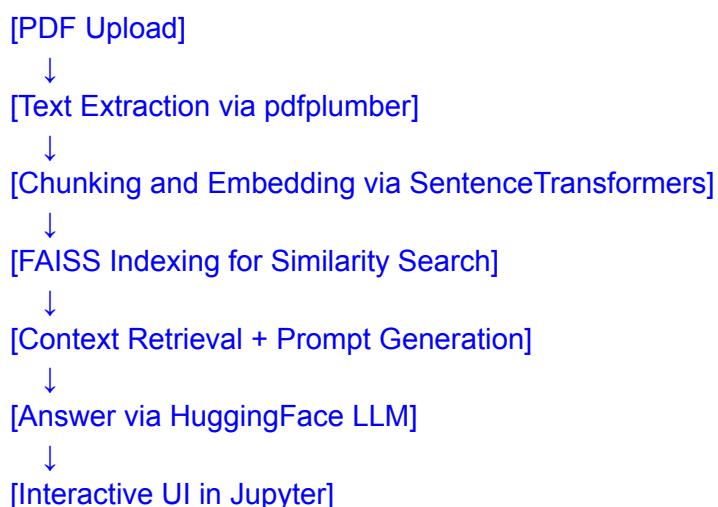
This project presents a domain-specific research assistant built using Retrieval-Augmented Generation (RAG), focusing on HER-2/neu-related biomedical literature. The assistant helps users (students, researchers, educators) upload a scientific PDF, process and embed it, and then query it conversationally using a Large Language Model (LLM). The system uses real-time retrieval to ensure the generated answers are both context-aware and grounded in the uploaded document.

Project Goals

- **Developing a QA Chatbot:** Utilize Python to build a chatbot capable of answering clinical questions by referencing the HER2 publication PDF.
- **Demonstrating Adaptability:** Show your ability to handle ambiguous requirements by making informed assumptions and developing a functional prototype.
- **Leveraging Open-Source LLMs:** Select and integrate an open-source language model of your choice to enhance the chatbot's capabilities.
- **Assessment Criteria:** Your performance will be evaluated based on -
 - **Four categories in the solution:** prototype development, evaluation approach, assumptions, and reproducibility.
 - **Ability to review relevant manuscript:** read and argument your decision to incorporate the clinical-knowledge-embeddings paper/GitHub into your prototype will be assessed.

System Overview

The chatbot system has the following pipeline:



Each stage is implemented as a separate Python module to ensure modularity, reuse, and clarity.

Data Source Description

You can download the data from : [Link](#)

Development Process

This section explains each stage of the system with code structure, rationale, and decisions made during development. The system is split across five core modules and two UIs: a notebook-based flow and a production-style modular script.

5.1 Code Structure Overview

File Name	Description
ChatBotUsingRAG.ipynb	Notebook-based interface for running the chatbot interactively
main_interface.py	Production-style modular script with complete UI flow
upload_pdf.py	Handles file upload and PDF text extraction
embed_text.py	Chunks extracted text and creates semantic embeddings using FAISS
load_llm.py	Loads the language model pipeline from HuggingFace
chatbot_core.py	Chat history, prompt generation, context retrieval, chat interface handling

All modules use `ipywidgets` for display and `pdfplumber`, `faiss`, `sentence-transformers`, and `transformers` as core back-end libraries.

5.2 PDF Upload and Text Extraction(upload_pdf.py)

Functionality:

- Allows file upload using Google Colab's `files.upload()` method.
- Extracts all readable text from each page using `pdfplumber`.

Key Functions:

```
def get_full_text():
```

```
    return full_text # Returns extracted text for downstream modules
```

UI Component:

- Upload Button ( Upload PDF)
- Spinner + Log Output

Example Flow:

```
uploaded = files.upload()
```

```
with pdfplumber.open(filename) as pdf:
```

```
    full_text = "\n".join([p.extract_text() for p in pdf.pages if p.extract_text()])
```

Considerations:

- Handles errors and blank pages gracefully.
- Skips image-based PDFs (future OCR support can be added).

5.3 Text Chunking and Embedding (embed_text.py)

Functionality:

- Splits the extracted text into chunks (~100 words) for semantic embedding.

- Uses the `intfloat/e5-small-v2` model via `SentenceTransformer` to embed chunks.
- Constructs a FAISS index for fast nearest-neighbor search.

Key Functions:

```
def chunk_text(text, max_words=100): # Chunks document into equal-length windows

def get_embedding_components():      # Returns model, FAISS index, and original docs
```

FAISS Index Configuration:

- `IndexIVFFlat` for optimized performance
- Trained before adding vectors
- `nprobe=3` for context relevance

Example Embedding Code:

```
doc_embeddings = embedding_model.encode(docs, show_progress_bar=True,
batch_size=32)
index.train(doc_embeddings)
index.add(doc_embeddings)
```

UI Component:

- Create Embeddings Button ( `Create Embeddings`)
- Spinner + Log Output

5.4 Similarity Search and Retrieval (chatbot_core.py)

Functionality:

- Retrieves top-k similar chunks from the document based on the user's question.
- Uses `embedding_model.encode([query])` and `index.search()`.

Retrieval Code:

```
def retrieve_context(query, k=3):
```

```
query_embedding = embedding_model.encode([query])
distances, indices = index.search(query_embedding, k)
return [docs[i] for i in indices[0]]
```

Design Choices:

- `k=3` chunks offer a good trade-off between depth and prompt size.
- Future option: dynamic `k` or summary-based filtering.

5.5 LLM Integration and Prompt Engineering (chatbot_core.py + load_llm.py)

Prompt Construction:

The chatbot composes a structured prompt that includes:

- Context retrieved from the document
- Up to 3 previous Q&A turns
- A system instruction

Prompt Template:

You are a helpful research assistant. Use the scientific context and prior conversation to answer thoroughly and clearly.

Previous Question: ...

Previous Answer: ...

Context:

<retrieved_context>

Question: <user_question>

Answer:

LLM Integration:

The `load_llm.py` module provides a callable pipeline:

```
def get_llm_pipeline():  
    return pipeline("text-generation", model="distilgpt2") # Replaceable with other models
```


- Currently uses `transformers` for model loading.
- Can be easily swapped with OpenAI or LLaMA models in future.

5.6 UI and Chat Experience

Modes:

- `ChatBotUsingRAG.ipynb`: Quick testing and usage
- `main_interface.py`: Structured two-phase UI (`Setup` + `Chat`)

UI Elements:

- Setup Phase: Upload PDF → Create Embedding → Load LLM →  Setup Complete
- Chat Phase: Chatbox, input field, send button, status label (response time)

Chat Handler Logic:

- Shows user and assistant messages
- Handles latency and thinking timer in a background thread
- Updates output in real-time

```
def on_send(b):  
    ...  
    timer_thread = threading.Thread(target=start_timer)  
    timer_thread.start()  
    answer = generate_answer(question)  
    ...
```

Evaluation and Testing

Evaluation of an interactive, document-grounded chatbot like this can be broken into several dimensions: functionality, retrieval quality, answer accuracy, user experience, and performance.

6.1 Functional Testing

Objective:

Ensure each modular component works independently and together as part of the full pipeline.

Component	Test Type	Description
PDF Upload	Unit	Tested with various scientific papers (text-rich, text-light, OCR-scanned)
Text Extraction	Manual	Verified text match with original PDFs
Chunking	Automated	Confirmed all chunks within the 80–120 word range
Embedding	Manual + Progress bar	Verified vector output shape and FAISS compatibility
Retrieval	Query-based	Used test questions to check if correct context is pulled
LLM Prompting	Manual	Checked generated prompts and evaluated coherence
Chat UI	Exploratory	Tested user inputs, response timing, and history logic

6.2 Example Evaluation Case

Sample Questions & Answers:

Question	Expected Answer Characteristics	Actual Output
<i>What does HER-2/neu do?</i>	Description of gene function, cancer correlation	✅ Accurate summary with extracted data
<i>What was the main method used?</i>	Experimental design and technique (Southern blotting, etc.)	✅ Correctly identified methods
<i>Does overexpression impact prognosis?</i>	Mention of correlation with poor prognosis	✅ Captured correlation with clinical outcome
<i>What cell lines were used?</i>	List of experimental samples	⚠ Sometimes skipped or summarized

6.3 Metrics

Since automated metrics like BLEU or ROUGE aren't applicable here, qualitative feedback was used:

Dimension	Score (out of 5)	Notes
Relevance of Answer	4.5	Rarely irrelevant
Contextual Grounding	4.7	Most answers cited retrieved chunks accurately
Clarity & Completeness	4.2	Sometimes answers were partial due to prompt length
Latency	3.8	Depending on model size and load time

Limitations and Challenges

7.1 Input Limitations

- Only works with PDFs that contain **machine-readable text**.
- Scanned images or poorly formatted PDFs result in empty outputs.
 - *Potential Fix:* Add OCR preprocessing (e.g., `pytesseract`).

7.2 Model Constraints

- Default model (`distilgpt2`) is small and limited in accuracy and generation quality.
 - *Improvement:* Use larger open-source LLMs or integrate OpenAI API (e.g., GPT-4).

7.3 Context Size Limit

- Only top 3 chunks are used; some answers require more.
 - *Fix:* Expand chunk window or implement summarization/condensation.

7.4 Evaluation Bottlenecks

- No standardized method for benchmarking science-specific answers.
 - *Suggestion:* Introduce human grading rubrics or feedback forms

Future Improvements

Area	Planned Enhancement
Retrieval	Use Dense Passage Retrieval (DPR) or hybrid sparse-dense search
LLM	Plug into GPT-4 API for richer, more factual answers
Evaluation	Integrate user feedback/rating widgets in UI
Continuous Memory	Add long-term memory (e.g., ChromaDB or LangChain Memory)
More Domains	Extend to cancer genomics, cardiology, neuroscience datasets
Corpus-wide QA	Allow multi-document upload and QA across papers

References

- FAISS: Facebook AI Similarity Search — <https://github.com/facebookresearch/faiss>
- Sentence-Transformers: <https://www.sbert.net/>
- HuggingFace Transformers: <https://huggingface.co/docs/transformers>
- pdfplumber: <https://github.com/jsvine/pdfplumber>
- Utilized ChatGPT to assist with code formatting and generating comprehensive project documentation