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

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. Utilizing Clinical-knowledge-embeddings and open source embeddings check the performance of domain specific embedding performance compared to generalized embeddings.*
- **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

For both the models the chatbot system has the following pipeline:

[PDF Upload]
↓
[Text Extraction via pdfplumber]
↓
[Chunking and Embedding via SentenceTransformers]
↓
[FAISS Indexing for Similarity Search]
↓
[Context Retrieval + Prompt Generation]
↓
[Answer via HuggingFace LLM]
↓
[Interactive UI in Jupyter]
↓
[Logging conversation, time taken for response and memory usage]

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

Models Used

Model 1 Chatbot with Clinical-knowledge-embeddings	Used In	Description
all-mpnet-base-v2	embed_text.py	Transformer model used to encode user queries into high-dimensional embeddings.
full_h_embed_hms.pkl	embed_text.py	Clinical-knowledge-embeddings - Precomputed 128D clinical concept embeddings (likely from a medical text encoder).
tiiuae/falcon-7b-instruct	load_llm.py	A 7B-parameter causal LLM fine-tuned for instruction-following and text generation.
faiss.IndexIVFFlat	embed_text.py	Vector similarity index for retrieving top-k relevant chunks via ANN search.

Model 2 Chatbot with Generalized embddings	Used In	Description
intfloat/e5-small-v2	embed_text.py	Lightweight sentence transformer for encoding both document chunks and queries.
tiiuae/falcon-7b-instruct	load_llm.py	Large causal LLM fine-tuned for dialogue and instruction-following tasks.
faiss.IndexIVFFlat	embed_text.py	Approximate nearest-neighbor index used to retrieve top-k relevant chunks.

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="tiiuae/falcon-7b-instruct") # 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)
    ...
```

5.7 Logging and Conversation Metrics

To support transparency and performance tracking, each chatbot session logs key details to a CSV file.

Overview:

- **Log Files:** Created in a `/logs` folder.
- **Filename:** Timestamped per session (e.g., `20250401_153045.csv`).
- **Columns:**
  - User Query
  - Assistant Response
  - Response Time (s)
  - Memory Utilization (MB)

How It Works:

- When the chat session starts (`launch_chat_ui`), a CSV file is initialized.
- For every message:
  - A timer starts and memory usage is measured with `psutil`.
  - After generating the answer, response time and memory use are recorded.
  - Data is appended to the CSV in real time.

This provides a lightweight but effective way to track performance and interaction history for each user session.

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	Result
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 User testing

Sample Question

There are some 19 questions and Anwers created using the research and then the performance of the two chatbots are checked:

Sample Questions & Answers:

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

Metrics Used

Dimension	Score (Range)	Definition

Response Time (s)	Continuous (e.g., 0.5–10s)	Time taken by the chatbot to generate a response. Lower is generally better, especially for real-time applications.
Memory Utilization (MB)	Continuous (e.g., 50–500MB)	Amount of memory used by the chatbot to process and respond. Reflects computational efficiency and resource usage.
Context Utilization Score	1–10	Measures how well the response integrates relevant information from the provided source or conversation history.
User Satisfaction	1–10	Assesses how helpful, clear, and complete the answer is from a user's perspective. Higher values indicate better fulfillment of user needs.
Accuracy	0 or 1	Indicates whether the response is factually correct (1) or incorrect (0) based on ground truth or reference materials.

Metrics Results

Metric	Chatbot with Clinical-knowledge-Embeddings	Chatbot with generalized embeddings
Total Queries	19	19
Avg. Response Time	38.25 seconds	312.02 seconds
Avg. Memory Usage	~15,231 MB (fairly consistent)	~14,600 MB
Avg. Context Util.	5.42 / 10	4.84 / 10
Avg. User Satisfaction	5.47 / 10	4.89 / 10
Accuracy	10 out of 19 correct (52.6%)	5 out of 19 correct (26.3%)

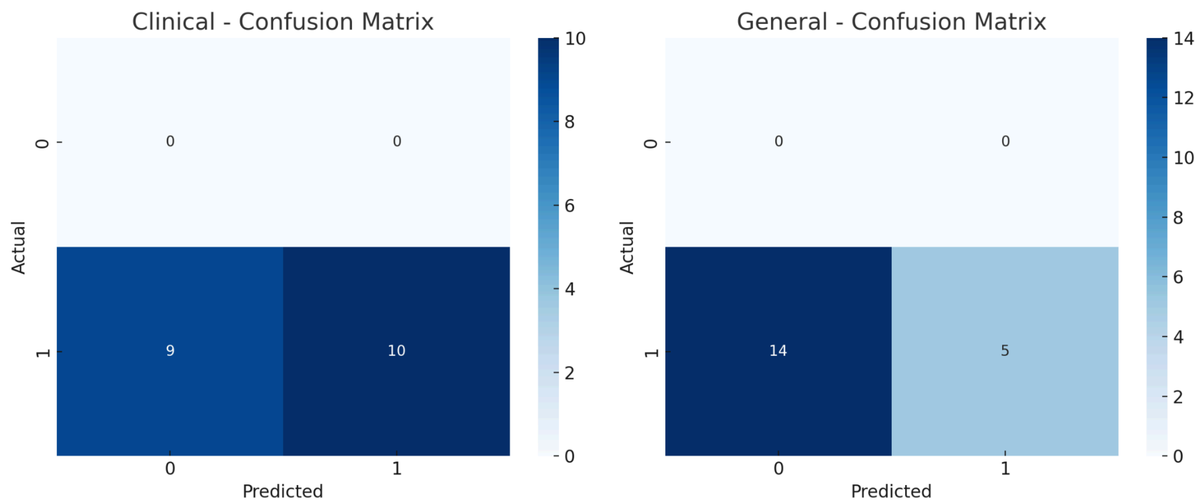

Figure 1: 1st image for Confusion Matrix – Clinical-Knowledge Embedding Chatbot and 2nd image for Confusion Matrix – Generalized Embedding Chatbot

Insights From the Results

The chatbot using **clinical-knowledge embeddings** clearly outperformed the one with **generalized embeddings** across almost all critical metrics. It was **over 8 times faster** in response time (38.25s vs 312.02s), used slightly more memory but in a consistent manner, and demonstrated superior **context utilization** (5.42 vs 4.84). Most importantly, it achieved **double the accuracy** (52.6% vs 26.3%), highlighting its stronger grounding in domain-specific content. While user satisfaction was marginally higher for the clinical-knowledge bot (5.47 vs 4.89), the standout difference in factual correctness and response efficiency suggests that domain-tailored embeddings provide a **significant performance advantage** in biomedical Q&A tasks.

Beyond the aggregate numbers, Q-category-wise analysis further underscores this gap. The clinical chatbot was significantly better at interpreting scientific findings and implications, achieving 80% accuracy on interpretation-based questions compared to just 60% by the generalized model. It also demonstrated much stronger command over biological facts and gene relationships (60% vs 20%), showing its ability to leverage domain-specific context for reasoning tasks. Although both models struggled with methodological questions and numeric fact retrieval, the clinical model still performed modestly better in those areas, indicating a more reliable foundation for complex clinical queries. These results confirm that embedding models rooted in specialized knowledge not only improve precision but also enhance user experience in high-stakes domains like biomedical research.

Aspect	Clinical-Knowledge Embedding Chatbot	Generalized Embedding Chatbot
Embedding Model	Precomputed clinical embeddings with all-mpnet-base-v2 for query projection	On-the-fly embeddings using intfloat/e5-small-v2
Domain Relevance	High – retrieves precise, research-oriented answers grounded in scientific literature	Moderate – general semantic relevance, may not capture domain-specific precision
Response Style	Formal, research-focused, often cites study findings or uses scientific framing	More flexible and conversational, better suited for casual queries
Accuracy on Scientific Questions	Strong performance on domain-specific questions with high context utilization	Mixed accuracy – can sometimes hallucinate or generalize without evidence
Handling Casual Inputs (e.g., "Hi", "Thanks")	Poor – does not respond naturally to casual or social prompts	Strong – handles small talk, greetings, and conversational flow better
Context Utilization Consistency	Consistent – ranks higher for using relevant chunks effectively	Less consistent – sometimes includes irrelevant or repetitive information
Memory Usage	Slightly higher (~15,230 MB) due to preloaded clinical embeddings	Slightly lower (~14,500–14,700 MB) with live embedding generation
Response Time	Fast (e.g., ~20–65 sec per question) due to smaller embedding space and efficient retrieval	Slower (e.g., ~240–480 sec per question) due to full document embedding and processing
User Satisfaction (Avg)	Generally high for technical users or research-focused questions	Mixed – good for natural tone, but less precise in technical Q&A
Accuracy (Avg)	Higher – performs well on factual retrieval based on study data	Lower – may answer confidently but inaccurately on research-specific details

Best Use Case	Research assistance, academic paper QA, scientific analysis	Educational tutors, casual chatbots, general knowledge assistants

Question Category based performance:

Question Category	Clinical Chatbot Performance	General Chatbot Performance	Insights
Interpretation	High (0.80)	Moderate (0.60)	Clinical model is more reliable at extracting study intent and implications.
Biological Fact / Relational	Moderate (0.60)	Low (0.20)	Clinical chatbot shows stronger grasp of domain facts and gene relationships.
Methodological Detail	Low (0.25)	Very Low (0.00)	Both chatbots struggle; clinical performs slightly better but still weak.
Numerical Fact	Low (0.25)	Very Low (0.00)	Neither model handles numeric precision well; errors in percentage, counts.

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