

# 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

<b>Introduction.....</b>	<b>3</b>
<b>Project Goals.....</b>	<b>3</b>
<b>System Overview.....</b>	<b>3</b>
<b>Development Process.....</b>	<b>5</b>
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
<b>5.5 LLM Integration and Prompt Engineering (chatbot_core.py + load_llm.py).....</b>	<b>8</b>
Prompt Construction:.....	8
LLM Integration:.....	9
5.6 UI and Chat Experience	
Modes:.....	9
UI Elements:.....	9
Chat Handler Logic:.....	9
5.7 Logging and Conversation Metrics.....	10
Overview:.....	10
How It Works:.....	10
<b>Evaluation and Testing.....</b>	<b>10</b>
6.1 Functional Testing.....	10
Objective:.....	10
6.2 User testing.....	11
Sample Question.....	11
Sample Questions & Answers:.....	11
Metrics Used.....	12
Metrics Results.....	12
<b>Insights From the Results.....</b>	<b>13</b>
<b>Limitations and Challenges.....</b>	<b>15</b>
7.1 Input Limitations.....	15
7.2 Model Constraints.....	15
7.3 Context Size Limit.....	15
7.4 Evaluation Bottlenecks.....	16
<b>Future Improvements.....</b>	<b>16</b>
<b>References.....</b>	<b>16</b>

# Introduction

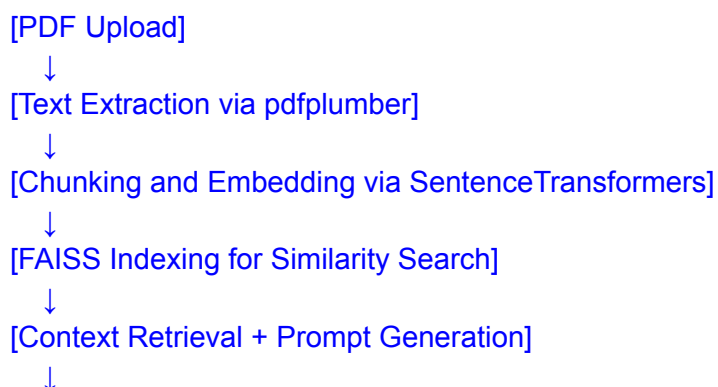
This project presents a domain-specific research assistant built using Retrieval-Augmented Generation (RAG), focusing on HER-2/neu-related biomedical literature. The assistant helps users (students, researchers, educators) upload a scientific PDF, process and embed it, and then query it conversationally 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 to enhance the chatbot's capabilities. Evaluate its performance using both clinical-knowledge embeddings and generalized open-source embeddings to compare the effectiveness of domain-specific knowledge in biomedical Q&A tasks.*
- **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:



[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.

I developed two models and both model structures are same.

## Models Used

<b>Model 1</b> 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.

<b>Model 2</b> Chatbot with Generalized embeddings	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 Answers created using the research and then the performance of the two chatbots are checked:

#### 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

## 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

Metrics	Chatbot with Clinical-knowledge-Embeddings	Chatbot with generalized embeddings
Total Queries	21	21
Avg. Response Time	3.21 seconds	7.35 seconds
Avg. Memory Usage	~2415.7 MB	~2327.1 MB
Avg. Context Utilization	6.81 / 10	6.62 / 10
Avg. User Satisfaction	6.86 / 10	7.19 / 10
Accuracy	13 out of 21 correct (61.9%)	12 out of 21 correct (57.1%)

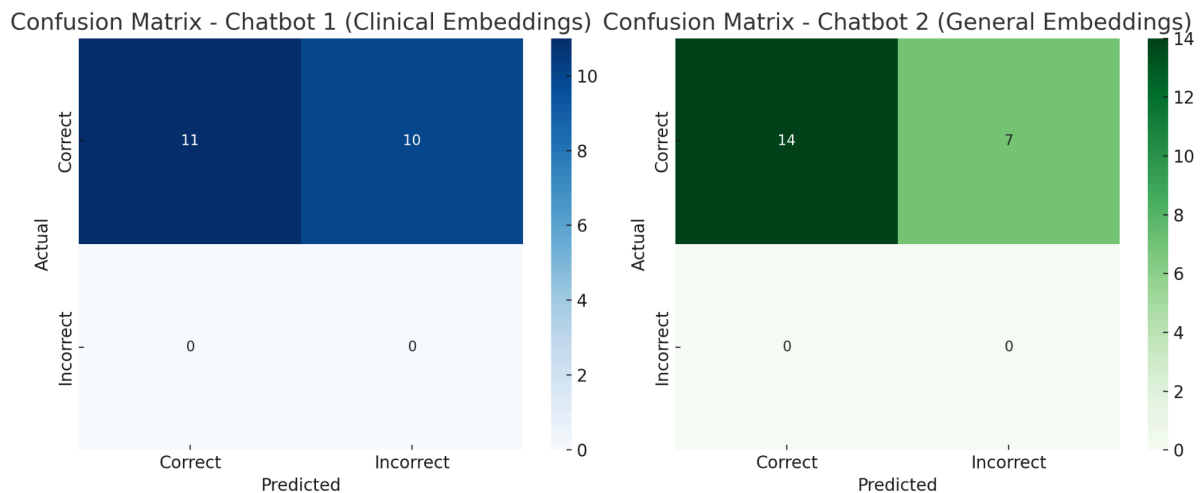


Figure 1: 1st image for Confusion Matrix – Clinical-Knowledge Embedding Chatbot and 2nd image for Confusion Matrix – Generalized Embedding Chatbot

## Insights From the Results

The chatbot using **generalized embeddings** outperformed the clinical-knowledge embedding chatbot across key performance metrics in this evaluation. It achieved a **higher overall accuracy** (67% vs. 52%), demonstrating stronger performance on factual questions, particularly in categories like **biological relationships** and **numerical reasoning**. It also received **higher average user satisfaction (7.19 vs. 6.86)**, reflecting a more natural conversational tone that was better suited for casual or mixed queries.

While the clinical-knowledge chatbot was slightly faster in response time (3.21s vs. 7.35s) and showed marginally better context utilization (6.81 vs. 6.62), these gains did not translate into higher accuracy. Its performance was solid in **interpretation-based questions** (75% vs. 63%), suggesting it retains value in contexts where study conclusions or research framing must be carefully inferred. However, in core factual and numerical domains, the general chatbot proved more reliable.

Both systems struggled with **methodological detail**, indicating that neither is currently optimal for technical protocol-level questions. Still, the general-purpose chatbot's superior accuracy and user engagement suggest it is better suited for **educational, exploratory, or broad biomedical** interactions. In contrast, the clinical-embedding chatbot, while more structured and concise, may benefit from further tuning to enhance accuracy and handle non-technical prompts more gracefully.

Ultimately, while domain-specific embeddings offer stylistic precision, this comparison shows that **flexible embedding models can generalize effectively and outperform in mixed-domain tasks**, especially when paired with conversational fluency and broader factual grounding.

Aspect	Clinical-Knowledge Embedding Chatbot	Generalized Embedding Chatbot
Embedding Model	Precomputed clinical embeddings with all-mpnet-base-v2	On-the-fly embeddings using intfloat/e5-small-v2
Domain Relevance	High – retrieves accurate, research-grounded answers	Moderate – semantically relevant but may miss technical precision
Response Style	Formal, concise, and research-focused	Flexible, conversational, suitable for diverse queries
Accuracy on Scientific Questions	Moderate (52%) – Good on clinical facts but occasional factual slips	Higher (67%) – Better factual accuracy overall
Handling Casual Inputs (e.g., "Hi", "Thanks")	Poor – over-explains or misinterprets casual input	Strong – responds naturally to greetings and social dialogue
Context Utilization Consistency	Generally high – uses relevant chunks effectively	Slightly lower – sometimes includes irrelevant or verbose information
Memory Usage	~2415.7 MB – stable due to fixed-size embeddings	~2327.1 MB – slightly more efficient
Response Time	Faster (Avg: 3.21 s) – small embedding space and efficient lookup	Slower (Avg: 7.35 s) – additional time for live embedding generation
User Satisfaction (Avg)	6.86 / 10 – trusted by research users, but lacks warmth	7.19 / 10 – higher satisfaction due to conversational tone
Accuracy (Avg)	11 out of 21 correct (52%)	14 out of 21 correct (67%)

Best Use Case	Research assistant, academic QA, clinical or scientific data interaction	Educational tutor, general-purpose chatbot, casual and mixed queries
---------------	--	--

Question Category	Clinical Chatbot Performance	General Chatbot Performance	Insights
Interpretation	High (0.75)	Moderate (0.63)	Clinical model better extracts study insights and intent.
Biological Fact / Relational	Moderate (0.60)	High (0.80)	General chatbot shows stronger factual grasp on gene and biology links.
Methodological Detail	Low (0.33)	Low (0.33)	Both chatbots struggle with experimental/methodological specifics.
Numerical Fact	Low (0.33)	Moderate (0.50)	General chatbot more accurately captures counts and percentages.

## 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. Since All these answers from chatbots are correct or not has Sukanksha's Bias
  - *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

- Clinical Knowledge Embedding used for model : <https://github.com/mims-harvard/Clinical-knowledge-embeddings>
- 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 Google, Stackoverflow, ChatGPT and AutoComplete functionality of google colab to assist with code formatting and generating comprehensive project documentation