RAG Chatbot: Technical Report

Project Overview

This Retrieval-Augmented Generation (RAG) chatbot project provides an interactive platform for querying user-supplied documents. The system uses **TinyLlama** as the language model, **MiniLM** for embeddings, **FAISS** for vector search, and **Streamlit** for the web UI. It follows a modern, modular architecture allowing seamless interaction from raw document upload to answer generation and streaming display[1].

Document Structure and Chunking Logic

Raw Data Handling

- **PDF Extraction**: Source documents are supplied as PDFs and placed in the `data/` directory.
- **Preprocessing**: `raw_data_preprocessing.py` reads each PDF, extracts page-wise text, removes headers, footers, HTML tags, and special characters, and concatenates the cleaned text into a single `dataset.txt` file[2].

Chunking

- The script `preprocessing.py` loads the cleaned corpus and splits it into **overlapping text chunks**.
    - Chunks are built using **`RecursiveCharacterTextSplitter`** with **`chunk_size`**=300 and **`chunk_overlap`**=50.
    - This overlap ensures context is preserved over chunk boundaries, reducing the chance of losing information between splits
- Both the chunks and their semantic embeddings are saved for later retrieval.

Embedding Model and Vector Database

Embedding Model

- **Model Used:** **`all-MiniLM-L6-v2`** (from SentenceTransformers)
    - **Why:** This model is selected for its balance between computational efficiency and semantic accuracy, making it ideal for fast, local similarity search and scalable deployment under hardware constraints.

Vector Database

- **Database:** FAISS (Facebook AI Similarity Search)[3][4].
    - Text chunks' embeddings are stored as dense float vectors in a FAISS **IndexFlatL2** index.
    - When a query is received, its embedding is compared to all stored vectors to efficiently find the nearest (most similar) document chunks, using L2 (Euclidean) distance.

Prompt Format and Generation Logic

Prompt Template

- **Prompt Construction:** On each user query, the system retrieves the top-K (typically 3) most similar chunks.
- The input to the LLM **(TinyLlama-1.1B-Chat-v1.0)** is formatted as:

```
You are a helpful assistant. Use the context to answer the user question.
Context: [Retrieved Chunks]
Question: [User Query]
```

- This approach ensures answers are always grounded in retrieved document content, minimizing hallucination risk.

Response Generation

- The LLM generates responses in streaming mode (sentence-by-sentence) for a responsive chat experience.
- Both the generated answer and the source chunks used are displayed to the user for transparency[5][6].

Example Queries and Responses

Below are sample questions (run after indexing documents with relevant content):

Query	Success/Failure	Answer Summary	Shown Sources
What is the refund policy mentioned in the document?	Success	Provides exact policy details found in retrieved chunk.	Yes (policy text)

List three actions that are prohibited when using eBay's Services.	Success	Lists the main eligibility rules.	Yes
What is the deadline for submitting the form?	Success	Returns the deadline date if present in document.	Yes (date text)
How do I apply if I'm overseas?	Partial	May answer if guidance is in document; else, states not found	Yes/No
What are the parking fees policy for 2025? (if not in doc)	Failure	States cannot find relevant information in sources.	Yes (unrelated)

Success case: Responses accurately reflect retrieved content; relevant source snippets are displayed.
Partial/failure case: System explicitly indicates if no related content could be found, helping manage expectations.

Limitations and Notes

- **Hallucination:** The TinyLlama model is prompted with retrieved context, which reduces hallucinations. However, if the retrieved chunks are not relevant, the model may produce plausible-sounding but incorrect details.
- **Model Limitations:** As a lightweight LLM, TinyLlama may:
    - Sometimes miss nuanced context or produce generic answers when insufficient information is present.
    - Struggle with very long or multi-part queries requiring synthesis across multiple documents.
- **Slow Responses:** On CPUs or low-resource GPUs,:
    - Embedding calculation during indexing may be slow for very large corpora.
    - Generating long answers may introduce perceptible delay.
- **Streaming Experience:** While answers are streamed for better UX, there may occasionally be a delay before the first sentence appears, owing to model and hardware limits.

Transparency Feature: All source chunks used to generate answers are shown to the user for verification, making it easy to detect hallucinations or edge-case reasoning failures[6].

Conclusion

This project delivers a robust, efficient RAG-based chatbot able to index and answer questions from custom document collections, offering source transparency and efficient API usage. It is ideal for lightweight deployments and can be easily extended or scaled as needed.

Github Repo Link - https://github.com/SachinMosambe/RAG-Chatbot-with-Streaming-Responses-Amlgo-Labs

DEMO Link- https://drive.google.com/drive/folders/15fNe1jU799DyMxaKBZVwF80yMpA3-0Pu?usp=sharing