RAG-Powered PDF Chatbot

1. Introduction

In today's data-driven landscape, PDF documents - ranging from financial reports and scientific papers to technical manuals - hold a wealth of critical information. However, extracting insights from them manually is inefficient and error-prone. This project presents a **Retrieval-Augmented Generation (RAG)** based solution that enables **chat-based interaction with PDF documents**, preserving structured data like tables and ensuring local, private processing.

Powered by **DeepSeek LLMs**, **Qdrant** vector database, and a sleek **Chainlit** UI, the system facilitates intelligent, context-aware querying of documents. Importantly, it functions locally, respecting user privacy and ensuring fast performance.

2. System Overview

The system allows users to upload a PDF and interact with its content via a chatbot interface. It uses **RAG principles** to semantically retrieve relevant content chunks and generate coherent responses using an LLM.

Key Features:

- Local-first processing (no cloud dependency)
- Accurate handling of structured data (tables, lists)
- DeepSeek integration for context-sensitive generation
- Elegant chat UI using Chainlit
- Fast vector retrieval using Qdrant

3. Technology Stack

Component	Description
Chainlit	Real-time chat UI
Langchain	To utilize local LLMs with the help of Ollama
Qdrant	Vector similarity engine for retrieval
Docling	Parses and chunks PDFs (including tables)
HuggingFace Embeddings	Embedding generation using transformer models
Docker	Runs Qdrant locally
uv	Fast Python dependency manager

4. Architecture & Workflow

High-Level Workflow:

1. **PDF Ingestion**: The PDF is parsed using `DoclingLoader` which preserves tables and other structures.
2. **Chunking**: Hybrid chunking splits content into semantically meaningful units.
3. **Embedding**: Each chunk is converted into a dense vector using `sentence-transformers/all-MiniLM-L6-v2`.
4. **Vector Storage**: Chunks are stored in **Qdrant**, enabling fast semantic search.
5. **Chat Querying**:
    - User enters a question.
    - The system retrieves relevant chunks.
    - The prompt is composed with context and sent to DeepSeek LLM.

5. Code Walkthrough

5.1 PDF Ingestion & Vectorization

```
from docling.chunking import HybridChunker
from langchain_docling.loader import ExportType

loader = DoclingLoader(
    file_path=FILE_PATH,
    export_type=ExportType.DOC_CHUNKS,
    chunker=HybridChunker(tokenizer=EMBED_MODEL_ID),
)
docling_documents = loader.load()
```

- **Docling**, a specialized loader from `langchain-docling` designed for robust PDF ingestion.
- Capable of parsing both **textual content and structured elements** (e.g. tables, lists, headers).
- **Hybrid Chunker** combines **rule-based** and **token-based** chunking strategies.
- Uses the tokenizer of the embedding model (`all-MiniLM-L6-v2`) to **split text naturally along semantic boundaries**, preserving context.

```
vectorstore = QdrantVectorStore.from_documents(
    documents=docling_documents,
    embedding=HuggingFaceEmbeddings(model_name=EMBED_MODEL_ID),
    url=qdrant_url,
    collection_name=COLLECTION_NAME,
)
```

- An interface to the Qdrant vector DB, enabling **high-speed vector similarity search**.
- Each stored chunk is associated with metadata (page number, file name), which is used for attribution later.

5.2 Chainlit Initialization and Prompt Setup

```
template = """You are a document analysis expert. When tables are present in

Context:
{context}

Question: {question}
"""

prompt = ChatPromptTemplate.from_template(template)
```

- Defines a structured prompt that guides the model's behavior.
- Sets the **role** of the LLM as a "document analysis expert".
- Using `ChatPromptTemplate` standardizes prompt formatting for consistency.
- Encourages the model to **focus on table structure, clarity, and factual accuracy**.

```
@cl.on_chat_start
async def on_chat_start():
    retriever = vectorstore.as_retriever()
    runnable = ({
      "context": retriever | format_docs,
      "question": RunnablePassthrough()
    } | prompt | llm | StrOutputParser())
    cl.user_session.set("runnable", runnable)
```

- Triggered once per user session when the chat opens.
- `as_retriever()`: Turns Qdrant vectorstore into a retriever function.
- `format_docs`: Concatenates retrieved chunks into a coherent input.
- `RunnablePassthrough()`: Passes the raw user query directly.
- Final pipeline = `retrieval -> prompt injection -> LLM -> output parsing`.

5.3 Chat Interaction and Source Attribution

```
class PostMessageHandler(BaseCallbackHandler):
    def __init__(self, msg: cl.Message):
        self.msg = msg
        self.sources = set()

    def on_retriever_end(self, documents, *, run_id, parent_run_id, **kwargs):
        for d in documents:
            source = d.metadata.get('source', 'unknown')
            page = d.metadata.get('page', 'N/A')
            self.sources.add((source, page))

    def on_llm_end(self, response, *, run_id, parent_run_id, **kwargs):
        if self.sources:
            sources_text = "\n".join([
                f"{source}#page={page}" for source, page in self.sources
            ])
            self.msg.elements.append(
                cl.Text(
                  name="Sources",
                  content=sources_text, display="inline"
                )
            )
```

- `PostMessageHandler` is a subclass of `BaseCallbackHandler` from LangChain, designed to intercept specific events in the RAG pipeline:
    - `on_retriever_end`: Runs after the retriever finishes fetching relevant chunks.
    - `on_llm_end`: Executes after the LLM generates the final response.

```
async for chunk in runnable.astream(
    message.content,
    config=RunnableConfig(callbacks=[
        cl.LangchainCallbackHandler(),
        PostMessageHandler(msg)
    ]),
):
    await msg.stream_token(chunk)
```

- allows **asynchronous streaming** of the language model's output. Instead of waiting for the full answer, tokens are yielded incrementally and rendered in real-time

6. Features & Capabilities

- **Structured Data Recognition**: Tables and lists are retained, referenced by row/column.
- **Contextual Understanding**: LLM leverages context to answer deeply nested questions.
- **Interactive UI**: Chat-based interface with streaming tokens and inline sources.
- **Scalability**: Can support multiple documents with minimal modifications.

7. Security & Local Processing

This application prioritizes **data privacy**. All operations, including embedding, retrieval, and generation, are executed **entirely on local infrastructure**. No document content or query is ever sent to the cloud.

- **Qdrant** and the app run via Docker on localhost.
- **DeepSeek LLM** is served via `ollama` locally.
- **No internet dependency** ensures corporate confidentiality.

8. Challenges Faced

- Ensuring table structures remain intact post-chunking.
- Managing long-context retention across pages.
- Optimizing retrieval relevance for small documents.
- Making Chainlit session management robust and stateful.

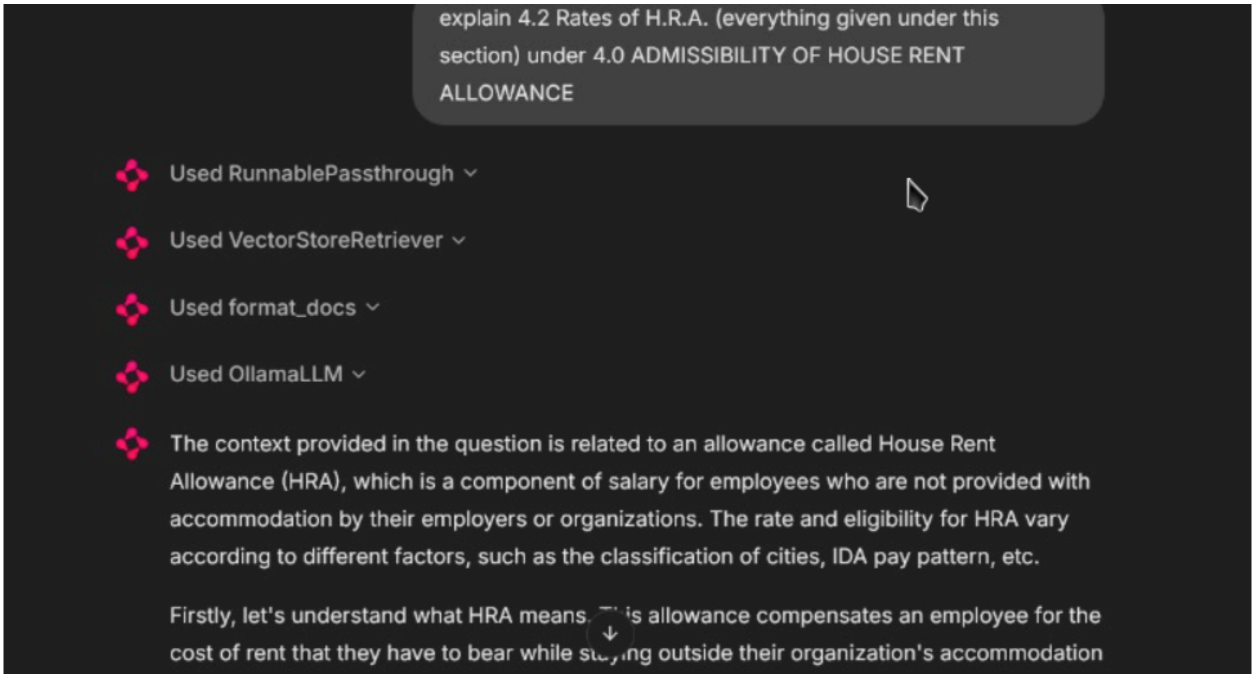
9. Future Improvements

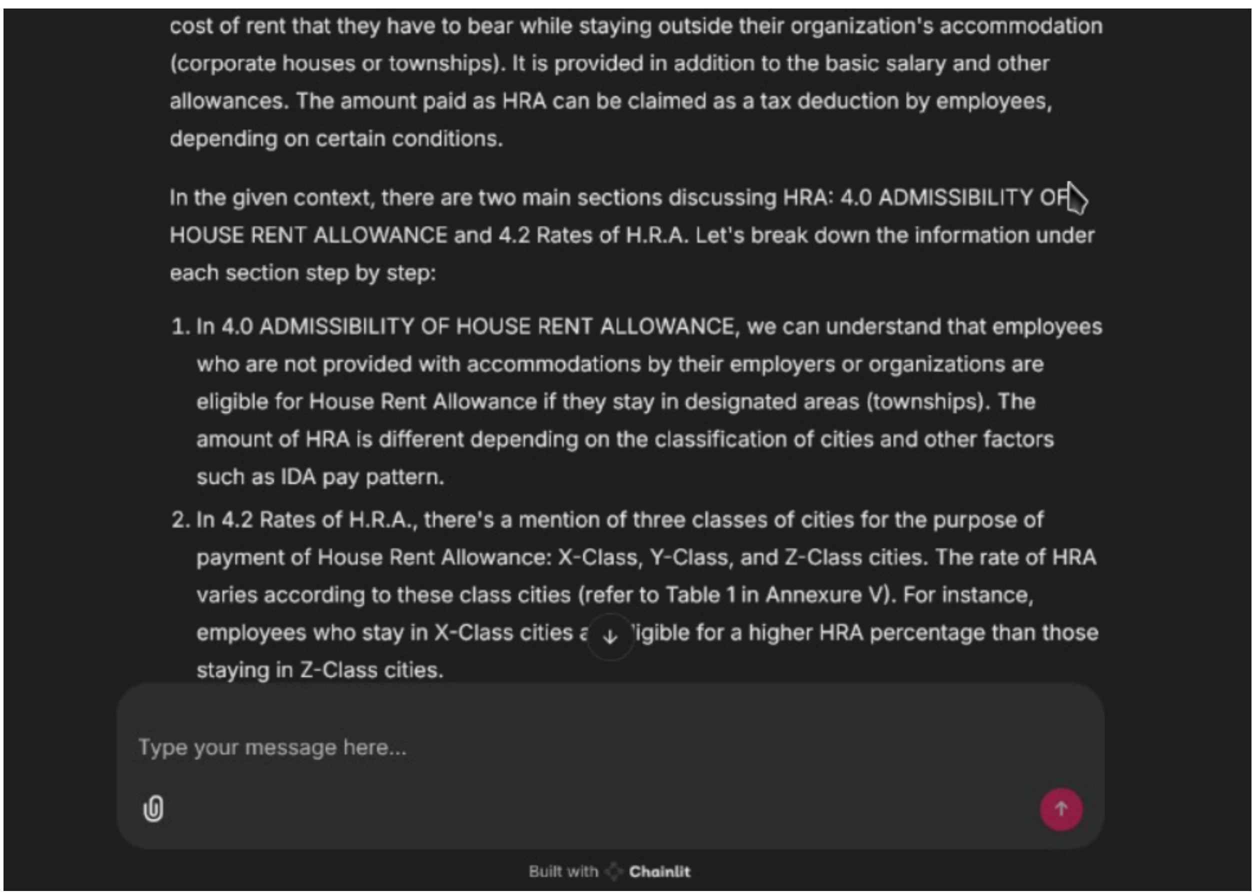
Several enhancements are already envisioned:

- **Auto Ingestion:** Currently, PDFs need to be manually ingested via `ingest.py`. A future version will include a drag-and-drop or monitored folder for automatic embedding.
- **Multi-Doc Search:** Enable querying across multiple documents with document-specific citation.
- **Visual Table Rendering:** Render extracted tables visually in the chat with CSV/Excel export support.
- **OCR Fallback:** Integrate OCR for scanned PDFs using tools like Tesseract or PaddleOCR.
- **Model Optimization:** Allow users to switch between models (DeepSeek, LLaMA, Mistral) based on performance/accuracy tradeoffs.
- **Fine-Tuned Prompts:** Introduce role-based personas (e.g., Financial Analyst, Legal Advisor) with prompt engineering.

10. Screenshots

Explanations of certain topics given in the provided pdf.





in this document, explain the section 10.1 WORKMEN AND SUPERVISORS (A. Project/Power Stations)

Used RunnablePassthrough

Used VectorStoreRetriever

Used format_docs

Using OllamaLLM

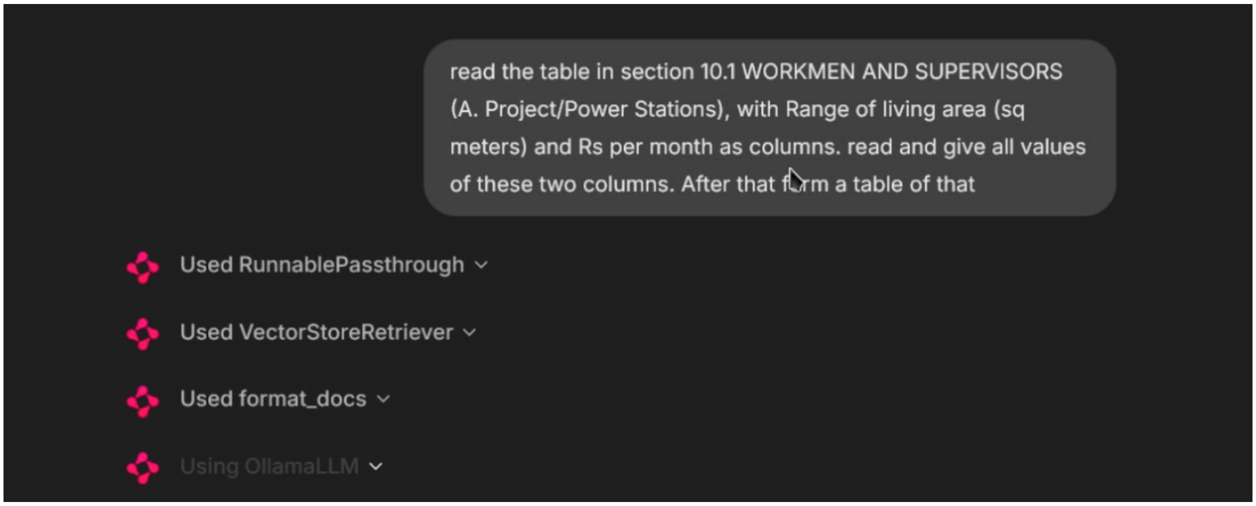
In this document, Section 10.1 pertains to "Workmen and Supervisors" for "Project/Power Stations." This section outlines the conditions under which workmen and supervisors are eligible for House Rent Allowance (HRA). The HRR rates for Corporation's accommodation in Project/Power Stations are provided as follows:

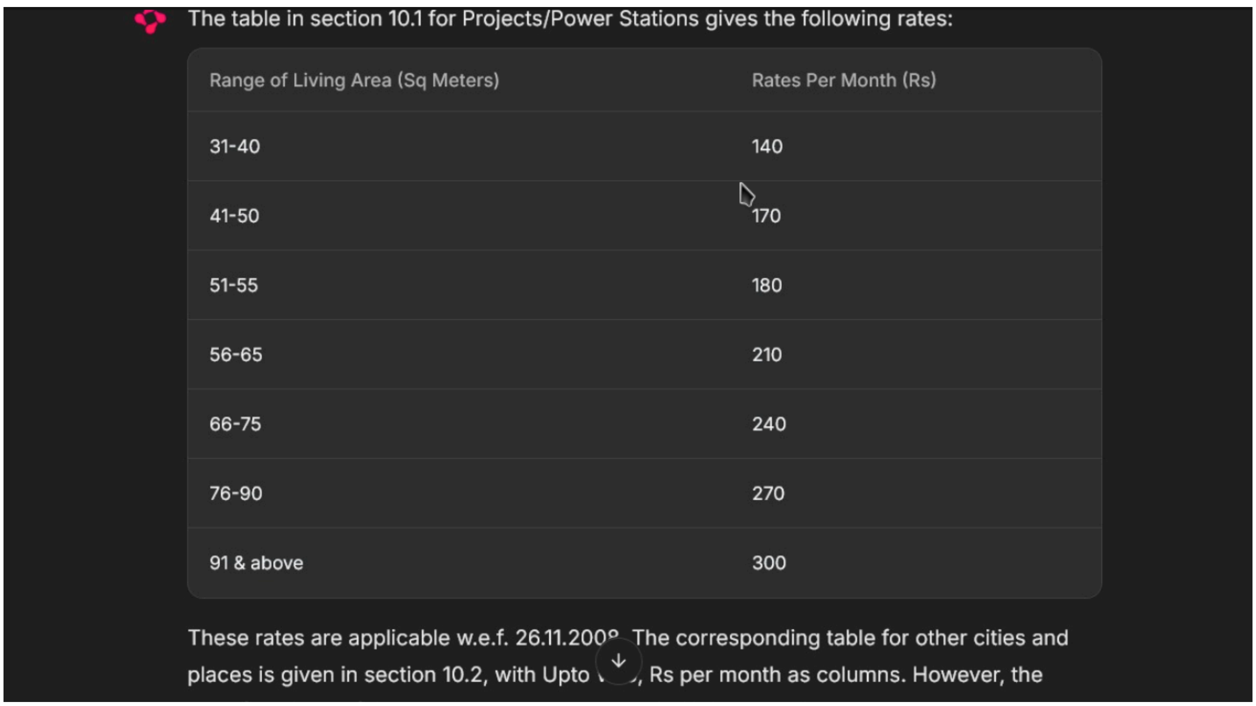
- For rent ranges from Rs. 31-40, HRA per month = Rs. 140
- For rent ranges from Rs. 41-50, HRA per month = Rs. 170
- For rent ranges from Rs. 51-55, HRA per month = Rs. 180
- For rent ranges from Rs. 66-75, HRA per month = Rs. 210
- For rent ranges from Rs. 76-90, HRA per month = Rs. 270
- For rent above Rs. 91, HRA per month = Rs. 300

Sources

./data/test2.pdf#page=N/A

Table related prompts answered by the model.





Range of Living Area (Sq Meters)	Rates Per Month (Rs)
31-40	140
41-50	170
51-55	180
56-65	210
66-75	240
76-90	270
91 & above	300

now read 10.2 EXECUTIVES(amended vide Part-I O/O No.46/10 dated 02.11.10)
Revised Rates of House Rent Recovery for Accommodation provided by
the Corporation
Grade, X(Existing A-1), Y (Existing A,B-1,B-2), Z (Existing C and
Un-classified)
give values in all these 4 columns and then explain the table to me

Used RunnablePassthrough

Used VectorStoreRetriever

Used format_docs

Used OllamaLLM

Grade	X(Existing A-1)	Y(Existing A,B-1,B-2)	Z(Existing C and Un-classified)
E-1 to E3	1780 Rs/pm	1340 Rs/pm	890 Rs/pm
E-4 & E-5	2180 Rs/pm	1640 Rs/pm	1090 Rs/pm
E-6 & Above	2750 Rs/pm	2060 Rs/pm	1380 Rs/pm

the above table you just gave can you now explain it to me

Used RunnablePassthrough

Used VectorStoreRetriever

Used format_docs

Used OllamaLLM

Sure, I'd be happy to explain! The information provided in the table is related to House Rent Recovery for Corporation-owned accommodations and the revised rates based on different classification of places or cities. Here's a breakdown of the key points from the context given:

1. There are different categories of places or cities classified as E1-E3 (X) and E4-E5 (Y), each with specific rates for house rent recovery. These rates vary depending on whether they fall under Existing A-1, Existing A, & B-2, or Unclassified category.

10. Conclusion

This project demonstrates the power of Retrieval-Augmented Generation in unlocking the insights trapped in complex PDF documents. By combining DeepSeek's language understanding, Qdrant's semantic retrieval, and Chainlit's interactive UI - all within a private, local-first environment - we deliver a fast, secure, and intelligent document interaction system.

With its modular architecture and clearly defined enhancement paths, the system lays the foundation for robust enterprise-grade document analytics.

Submitted by: Sumedha Singh Paliwal
Date: August 2025