RAG System: Methods and Reasoning

1. Introduction

This project builds a system that can read PDF documents and answer questions about them. The system can handle different types of content in PDFs - regular text, tables with data, and images with text in them. It works by extracting all the information from PDFs, storing it in a searchable way, and then using an AI model to answer questions based on what it found in the documents.

2. Problem Understanding

What the System Needs to Do

The system has to process PDF documents and answer user questions about three different kinds of content:

- Regular text and written information
- Data organized in tables
- Information in pictures, charts, and scanned text

Main Challenges

PDFs are tricky to work with because they can store information in many different ways. A single document might have regular text, complex tables, and important details hidden in images. No single tool can extract all these different types of content well. Tables can have complicated layouts that break normal text extraction. Pictures and charts often contain crucial information that requires special processing to read. The system also needs to keep track of how different pieces of information relate to each other while making everything searchable.

3. Data Processing / Preprocessing

Tools Used to Extract Content from PDFs

The system uses three different tools because each one is good at handling different types of content:

- **PyMuPDF** reads regular text from PDF pages. It works well with most PDF formats and can handle different types of text encoding.

- **Camelot** finds and extracts tables from PDFs. It can identify where tables begin and end, understand cell boundaries, and convert table data into a format that's easier to work with.
- **Tesseract OCR** reads text from images. When the system finds pictures or charts in a PDF, it uses this tool to extract any text that might be embedded in them.

Cleaning Up the Extracted Content

After extracting content, the system does basic cleanup to remove obvious errors and format issues. This includes fixing spacing problems and handling special characters that might cause issues later. When working with tables, the system converts them into readable text while keeping the structure that shows which information belongs in which columns.

4. Document Representation

Breaking Documents into Smaller Pieces

Large documents get split into smaller chunks that are easier to process:

- **Chunk size**: 1024 characters - big enough to include meaningful context but small enough for the system to handle efficiently
- **Overlap**: 256 characters between chunks so important information doesn't get cut off at boundaries

Different types of content get handled appropriately, tables are converted to text that still shows their structure, and text from images gets mixed in with regular content.

Converting Text to Embeddings for Search

The system uses a tool called BAAI/bge-base-en-v1.5 to convert all text into numerical representations that capture meaning. This was chosen because:

- It performs better than other options when searching for relevant information
- It's designed specifically for finding related content rather than general text understanding
- It can run locally without sending data to external services
- It creates number patterns that effectively represent what text means

5. Indexing and Storage

The system uses ChromaDB to store all the converted text. This was chosen because:

- It's open source, so there are no licensing concerns

- It stores information locally, keeping documents private
- It works well with other tools the system uses
- It provides fast searching through large amounts of information

How Information Is Organized

Each piece of text gets stored along with details about where it came from:

- Which original document it came from
- What piece number it is within that document
- What type of content it is (regular text, table data, or text from images)
- What page it was found on

This organization allows the system to find relevant information and tell users exactly where it came from.

6. Query Processing

Handling User Questions

When users ask questions, the system does minimal processing to prepare them for searching. It keeps the natural language structure while making sure the question format works with the search system.

Finding Relevant Information

The system converts user questions into the same type of numerical representation used for documents. It then searches for the 10 most similar pieces of content using mathematical similarity calculations. This provides enough context to answer questions thoroughly without overwhelming the answer generation process.

7. Answer Generation

AI Model Choice

The system uses DeepSeek-R1, a 1.5 billion parameter AI model that runs locally. This model was chosen because it's good at reasoning through information and staying factual, while being small enough to run efficiently on available hardware.

Ensuring Accurate Responses

The system uses carefully written instructions that tell the AI model exactly how to behave:

- Only use information from the documents provided
- Structure answers in a specific format with sources
- Clearly state when information isn't available rather than guessing
- Don't use outside knowledge or make assumptions

This approach prioritizes getting the right answer over always having an answer, which is important for business applications where accuracy matters more than completeness.

Keeping Track of Conversations

The system remembers the last 5 exchanges with users so it can handle follow-up questions naturally, but limits memory use to keep the system running efficiently.

8. Handling Different Content Types

Working with Tables

Tables need special handling to preserve their meaning:

- The extraction tool maintains relationships between rows and columns
- Tables get converted to text format that still shows their structure
- Table content gets mixed with surrounding text context
- The system keeps track of which content came from tables

Processing Images and Figures

Visual content focuses on extracting readable text:

- Images are pulled out of PDFs during initial processing
- OCR software converts any text in images into searchable content
- Poor quality results get filtered out to avoid adding garbage to the system
- Text from images gets labeled so the system knows where it came from

Combining Everything

All different types of content end up in the same searchable database while keeping information about what type each piece originally was. This lets the search system find relevant information regardless of whether it came from text, tables, or images.

9. System Architecture

The system follows a clear step-by-step process:

1. **Getting content**: PDF files → Extract text, tables, images → Combine everything → Split into chunks
2. **Making it searchable**: Text chunks → Convert to numbers → Store in database
3. **Answering questions**: User question → Search for similar content → Get relevant pieces → Generate answer

Component Integration

The system components integrate through standardized interfaces and data contracts:

- PDF processing modules output LangChain Document objects with consistent metadata schemas
- Vector store implements retriever interface for uniform similarity search operations
- Language model integration maintains stateful conversation context through memory buffers
- FastAPI layer provides RESTful endpoints with structured request/response models

Information Flow

Information moves in one direction from source documents through processing steps to final answers. The system stores processed information so it doesn't have to redo work when restarting.

10. Implementation Details

Technologies Used

- **Python** as the main programming language
- **FastAPI** for creating web interfaces that other programs can use
- **LangChain** for connecting all the pieces together
- **ChromaDB** for storing and searching information
- **Ollama** for running the AI model locally

How Code Is Organized

The system is built as separate modules that each handle specific tasks:

- pdf_processor.py: Extracts content from PDFs
- tesseract_ocr.py: Processes images to find text
- llm_loader.py: Sets up the AI model
- rag_pipeline.py: Coordinates the main search and answer process
- main.py: Provides the web interface

The system is built using proven tools and keeps different parts clearly separated for easier maintenance and development.