Build a RAG System

### **Authors:**

**Nishanth Fiona [fiona.nf12@gmail.com]**

**Arnav Sinha [arnavonline@gmail.com]**

**Akshay Sharma [akshaysharma6293@gmail.com]**

#### **Submission Date: April 29, 2025**

### **Problem Statement:**

Modern organizations often store critical information in unstructured documents such as PDF policy files, manuals, legal contracts, or support guidelines. For example, a user may need to ask questions about an insurance policy PDF like *"icici-bharat-griha-raksha-policy.pdf"*, and expect instant, accurate answers grounded in that document.

Traditional keyword-based search fails to:

* Understand the context of the question,
* Match semantically similar but differently worded content,
* And synthesize answers across multiple sections.

This leads to inefficient information retrieval, frustrated users, and missed insights.

### **Why LangChain (or LlamaIndex) is an Ideal Framework:**

LangChain offers a modular **Retrieval-Augmented Generation (RAG)** pipeline that solves this problem by:

* **Reading and splitting PDFs** into manageable chunks.
* **Embedding** those chunks into vector space using LLM-powered embeddings.
* **Storing** and **retrieving** relevant chunks efficiently using vector databases like **Chroma**.
* **Re-ranking** them using powerful cross-encoders for relevance.
* **Generating human-like answers** with Google’s Gemini (or other LLMs) using only the retrieved context, ensuring factuality.

LangChain handles all these steps **end-to-end**, with powerful connectors and easy customization.  
**It is ideal for document Q&A systems where precision and context matter.**

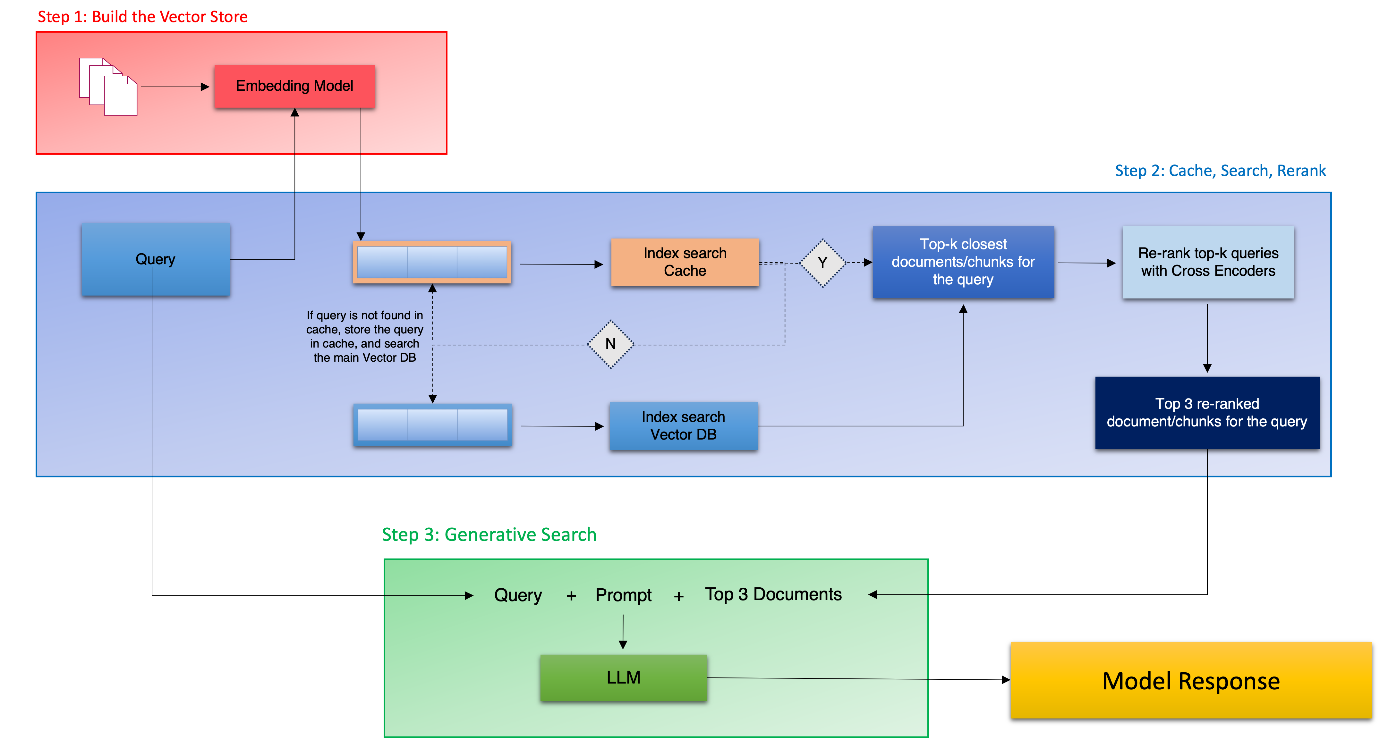
### **Data Sources**

* **Primary data source:**  
  Text-based PDF documents such as [icici-bharat-griha-raksha-policy.pdf](https://github.com/Nishanthfiona/Helpmate-AI/blob/main/icici-bharat-griha-raksha-policy.pdf)
* **Format:**  
  These are multi-page PDFs loaded directly from Google Drive (/content/drive/MyDrive/HelpMate AI Codes/).
* **Preprocessing:**  
  PDFs are split into overlapping text chunks to retain context between chunks **(chunk size = 1000, overlap = 150).**

### **System Design & Layers:**

* **Chunking with Overlap**: To preserve semantic flow across pages and ensure relevant context is not lost.
* **Embedding with Google’s Model**: Used text-embedding-004 for high-quality dense representations.
* **Vector Store: Chroma**: Chosen for its persistent local storage, quick indexing, and LangChain support.
* **CrossEncoder for Reranking**: A second filtering layer ensures better precision of context retrieved.
* **Gemini 1.5 Flash Model**: Fast, reliable response generation with zero temperature for factual consistency.
* **LangChain Expression Language (LCEL)**: Clean chaining of components to ensure maintainability and modularity.

### **System Architecture**



### **Implementation Details**

**1. Data Loading and Preprocessing**

* The system reads and processes documents (like insurance PDFs) using PyPDFDirectoryLoader from langchain\_community.document\_loaders.
* These documents are stored in a Google Drive folder (/content/drive/MyDrive/HelpMate AI Codes/) and loaded into the notebook environment.
* Each PDF page is loaded as an individual Document object, forming the base for chunking.

**2. Text Chunking**

* The loaded text is split into overlapping chunks using the RecursiveCharacterTextSplitter with chunk\_size=1000 and chunk\_overlap=150.
* This ensures contextual continuity between segments while avoiding loss of meaning.
* The chunking logic is crucial to maintain the semantic relevance of individual pieces fed to the embedding model.

**3. Embeddings and Vector Store**

* Each text chunk is embedded using **Google’s Generative AI embeddings** (text-embedding-004 model) via langchain\_google\_genai.GoogleGenerativeAIEmbeddings.
* These embeddings convert the chunks into high-dimensional vector representations.
* The embeddings and metadata are stored using **Chroma Vector Store**, with persistence enabled through Google Drive (chroma\_db\_langchain directory), ensuring data remains intact between sessions.

**4. Retriever and Reranker**

* A retriever is created from the vector store using .as\_retriever() with k=20 to fetch the top 20 semantically similar documents based on user queries.
* To improve the relevance and precision of retrieved chunks, the retriever is wrapped with a **ContextualCompressionRetriever**, which includes a **CrossEncoderReranker** powered by the cross-encoder/ms-marco-MiniLM-L-6-v2 model from HuggingFace.
* This reranker intelligently scores and selects the top 3 most contextually appropriate chunks from the initial 20 candidates, improving answer quality.

**5. Answer Generation**

* The final prompt is composed using ChatPromptTemplate to format the top documents and instruct the language model to generate a context-aware answer.
* The answer is generated using **ChatGoogleGenerativeAI** with the **gemini-1.5-flash-latest** model.
* The temperature is set to 0.0 to ensure factual, deterministic answers grounded in the retrieved content.
* The entire prompt-response interaction is orchestrated using LCEL, enabling seamless chaining of context retrieval and answer generation.

**6. Storage and Runtime Management**

* All outputs, including embeddings, vector databases, and logs, are saved in **Google Drive**, ensuring durability and minimizing reprocessing time.
* The notebook is designed for **reproducibility**, allowing future users or evaluators to run each step independently without manual configuration.

### **Challenges Faced**

* **Chunk Quality**: Overlap needed to be tuned carefully — too little caused loss of context, too much created redundancy.
* **Reranking Efficiency**: Integrating the CrossEncoder re-ranker required additional computation but significantly improved result quality.
* **Context Management**: Keeping responses grounded strictly to the source documents needed robust prompt templates.
* **Colab Storage Paths**: Ensuring persistence and reusability required managing custom file paths inside Google Drive.

### **Lessons Learned:**

* RAG pipelines are powerful but require tuning of chunking and context retrieval
* Re-ranking adds major boost to answer relevance
* LangChain simplifies orchestration but needs careful prompt engineering
* Persistent storage is critical in Colab-based projects

### **Results & Output**

The **HelpMate AI** RAG system successfully answers user queries with high contextual accuracy, providing **factual, document-grounded responses** extracted from complex insurance PDFs.

Example:   
  
Query:   
**Explain the Home Building Cover?**Model Reply:   
**The Home Building Cover's sum insured is the prevailing cost of construction at**

**the policy's start date, as declared by the policyholder and accepted by the**

**insurer**

**icici-bharat-griha-raksha-policy.pdf, Page: 5]. This is the maximum payable**

**amount if the building is a total loss. For policies longer than one year, the**

**sum insured increases by 10% annually on each policy anniversary, up to a**

**maximum of 100% of the initial sum insured, without extra premium**

**icici-bharat-griha-raksha-policy.pdf, Page: 5]. For annual policies, the sum**

**insured increases daily by 1/365th of 10% of the initial sum insured**

**icici-bharat-griha-raksha-policy.pdf, Page: 5]. Loss of rent coverage is**

**available for a reasonable time (maximum three years from when the home becomes**

**uninhabitable) to repair the building to a livable state, requiring a**

**certificate from an architect or local authority confirming unsuitability for**

**living**

**icici-bharat-griha-raksha-policy.pdf, Page: 6]. Loss of rent claims are only**

**accepted if the physical damage claim for the home building is accepted**

**icici-bharat-griha-raksha-policy.pdf, Page: 6].**