**Research Paper Based Smart Semantic Search System**

**By Tejas, Tushar and Pranit**

**1.** **Major Objectives and Considerations**:

The primary objectives of the project are as follows:

* Develop a **semantic search system** using the **RAG pipeline** (Embedding Layer, Search & Rank Layer, Generation Layer) for efficient document retrieval.
* Extract relevant information from **a sample research paper from a University of Poland**, store it in a structured format, and generate vector representations by embedding the text using the **all-MiniLM-L6-v2** model. The embeddings are stored in a **ChromaDB collection** for retrieval.
* Implement a **cache layer** to enhance system performance by storing and retrieving previous queries and their results.
* Build a **robust generative search system** capable of effectively and accurately answering questions from the research document.
* Use the **Gemini model** for prompt-based response generation, ensuring well-structured and contextually relevant answers.

**2. Project Architecture**

**2.1. RAG Pipeline**

* **Embedding Layer**: Extract text and tables from **a sample research paper from a University of Poland**, convert them into a structured dataframe, and apply an **overlapping chunking strategy** to maintain context. Generate vector representations using the **all-MiniLM-L6-v2** embedding model and store them in **ChromaDB**.
* **Search and Rank Layer**: Perform **semantic similarity search** on the knowledge bank based on user queries, retrieving the **top 3 closest document chunks**. Apply **re-ranking using a cross-encoder** to improve relevance.
* **Generation Layer**: Use the retrieved results, the original user query, and a **well-structured prompt** to generate coherent and well-cited responses using the **Gemini model**.

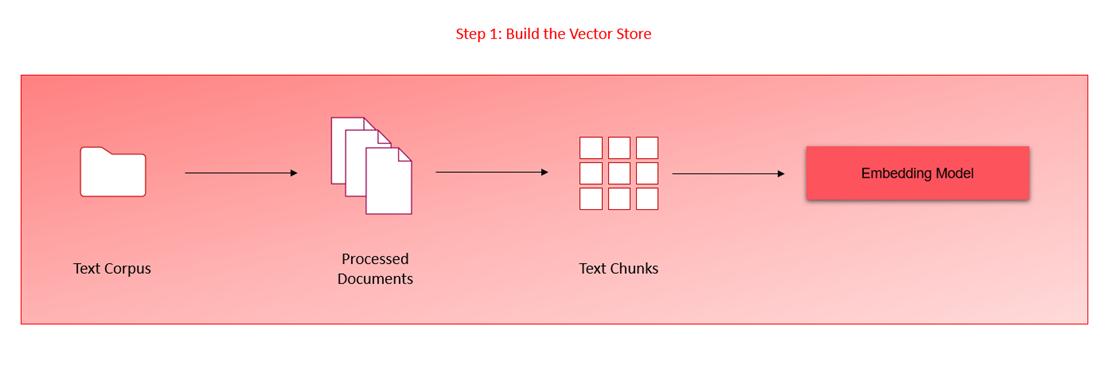
**2.2. Cache Implementation**

* Set a **semantic similarity threshold of 0.2**.
* Store queries and results in a **cache\_collection** in **ChromaDB** for efficient embedding and retrieval.
* Use **ChromaDB’s utility functions** to manage documents, IDs, and metadata in the cache.

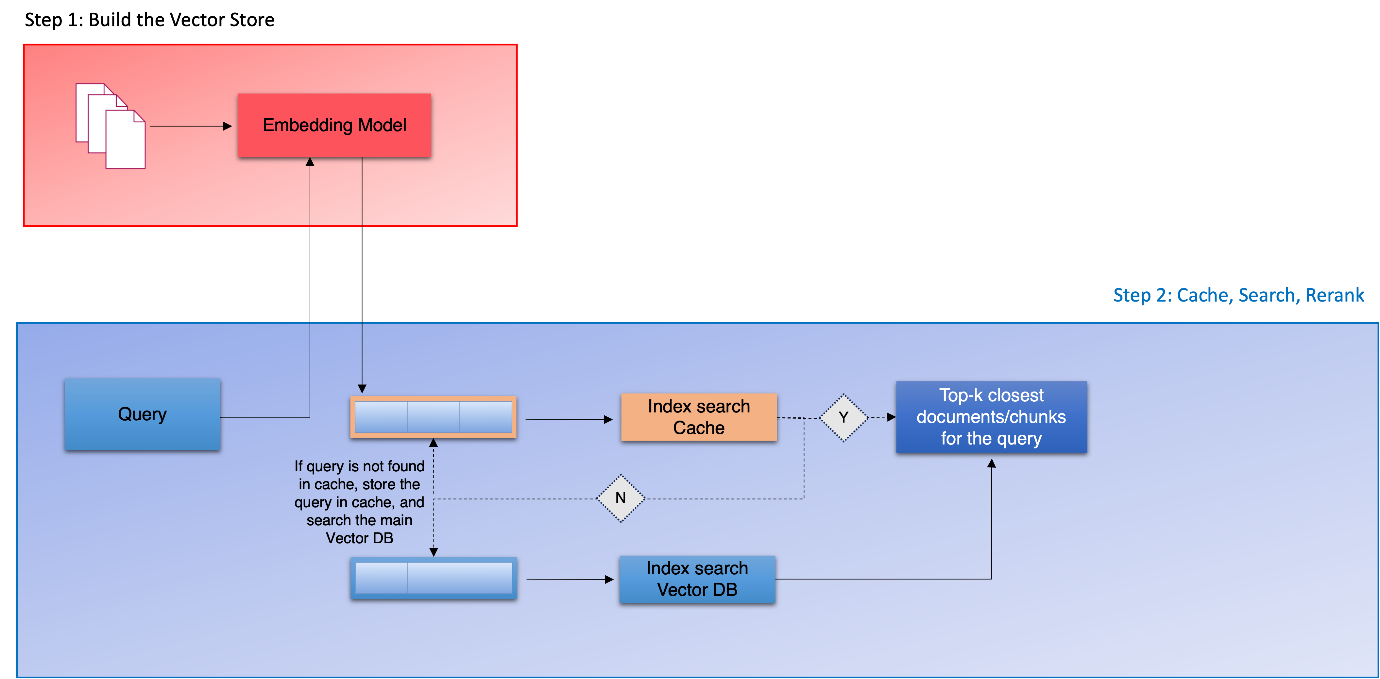
**3. Implementation**

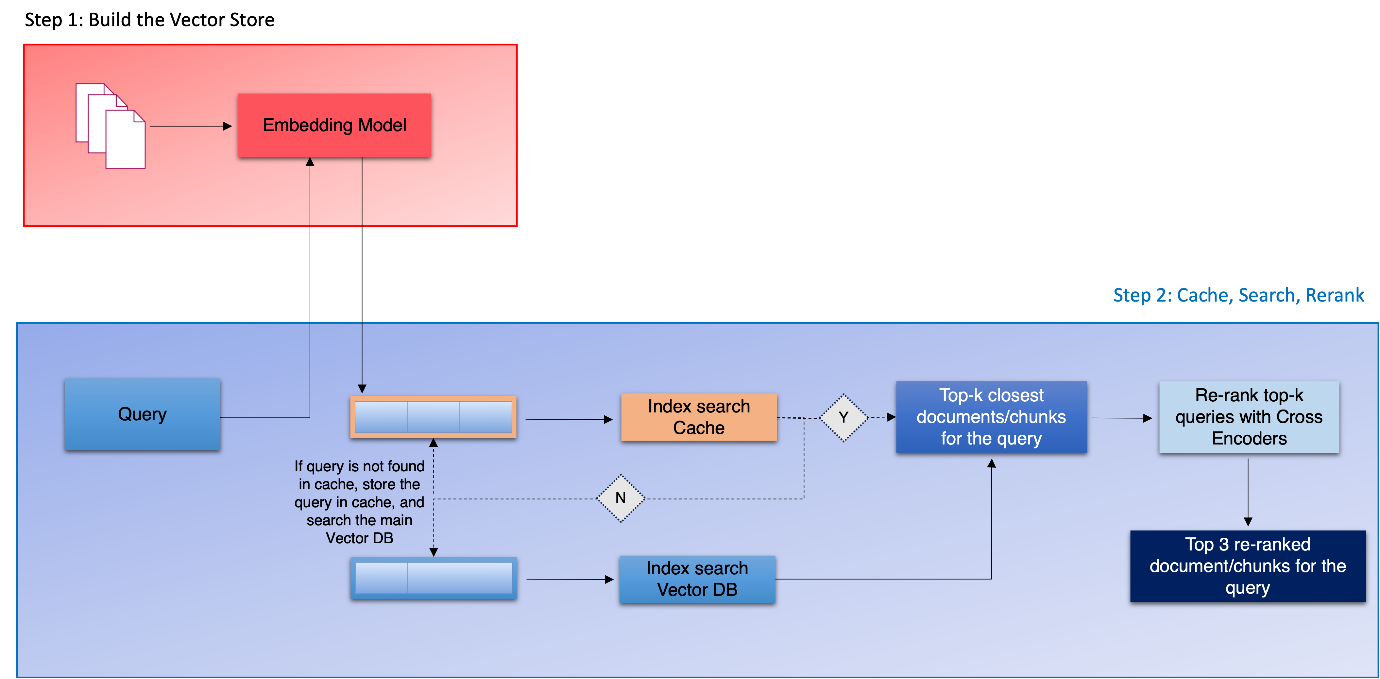
* Use **Jupyter Notebook** for development, leveraging libraries like **pdfplumber, sentence-transformers, ChromaDB**, and **Gemini API** for document processing, embedding, caching, and response generation.
* Implement functions to **extract text and tables from PDFs**, **chunk text using overlap chunking**, **create a dataframe**, **generate vector embeddings**, and **perform semantic searches** using the **RAG pipeline**.
* Design a **robust prompt** to structure the LLM response properly, ensuring relevant information is passed effectively. Utilize **few-shot examples** to enhance the quality of the generated answers.

Embedding Layer:



Search Layer:



When using cross\_encoder:  


**4.Challenges Faced:**

* Data Quality and Preprocessing
* Extracting relevant information from insurance documents, especially from complex text structures, is challenging.
* Effective Chunking Strategies
* Chunk Overlap and Size: Determining the optimal chunk size and overlap to capture meaningful context without losing coherence is challenging.
* Query Understanding and Matching
* Designing queries which are relevant to the document and that require sophisticated understanding and reasoning

**5.Lessons Learned:**

Efficient Document Processing: Processing PDFs efficiently is crucial; libraries like pdfplumber play a vital role.

Semantic Search Optimization: Fine-tune semantic search parameters and thresholds for optimal results.

Cache Management: Implement an effective cache management strategy to balance storage and retrieval efficiency.

**6.Conclusion:**

The project successfully implements a semantic search system with the RAG pipeline and cache layer and Generative search. The objectives are met, and the challenges are overcome with lessons learned for future improvements. The system provides a scalable and efficient solution for document retrieval and information extraction.