# Project Report: Semantic Search System with RAG Pipeline and Cache Implementation

Objectives:

The primary objectives of the project are as follows:

- Build a semantic search system employing the RAG (Embedding Layer, Search and Rank Layer, Generation Layer) pipeline for effective document retrieval.
- Extract pertinent information from PDF documents, organize it in a structured format, and create vector representations using the SentenceTransformerEmbedding's all-MiniLM-L6-v2 model.
- Integrate a cache layer to boost system performance by storing and retrieving previous queries and their results.

1. Design:

1.1. RAG Pipeline:

**Embedding Layer:** Extract text and tables from PDFs, convert them into a dataframe, and create vector representations using OpenAI's text-embedding-ada-002 model. Store these embeddings in ChromaDB.

**Search and Rank Layer:** Conduct a semantic similarity search on the knowledge base based on user queries, retrieving the top K most relevant documents or chunks.

**Generation Layer:** Use the results from the previous layer, along with the original user query and a carefully crafted prompt, to generate coherent answers with a language model.

1.2. Cache Implementation:

- Set a threshold of 0.2 for semantic similarity to determine if a query should be handled by the cache.
- Store queries and their results in a cache_collection within ChromaDB for efficient embedding and future searches.
- Utilize ChromaDB's utility functions to add documents, IDs, and metadata to the cache_collection.

2. Implementation:

Use Jupyter Notebook for development and leverage libraries such as pdfplumber, tiktoken, openai, ChromaDB, and sentence-transformers for document processing, embedding, and caching. Implement functions to extract text and tables from PDFs, create a dataframe, generate vector embeddings, and perform semantic searches using the RAG pipeline. Develop a cache system using ChromaDB to store and retrieve previous queries and their results.

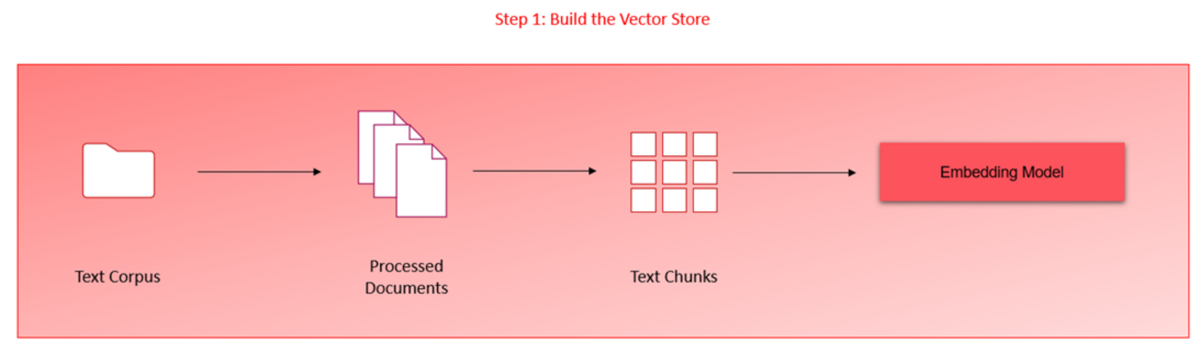
The three layers for RAG pipeline are:

1. Embedding Layer
2. Search Layer
3. Generation Layer

Embedding Layer:

**Processing and Chunking**: Investigate and assess various strategies for efficient PDF document processing, cleaning, and chunking. Analyze how different chunking approaches affect the quality of the retrieved results.

**Embedding Model Choice:** Select either OpenAI's embedding model or SentenceTransformers from HuggingFace. Evaluate how the choice of model affects the quality of vector representations.

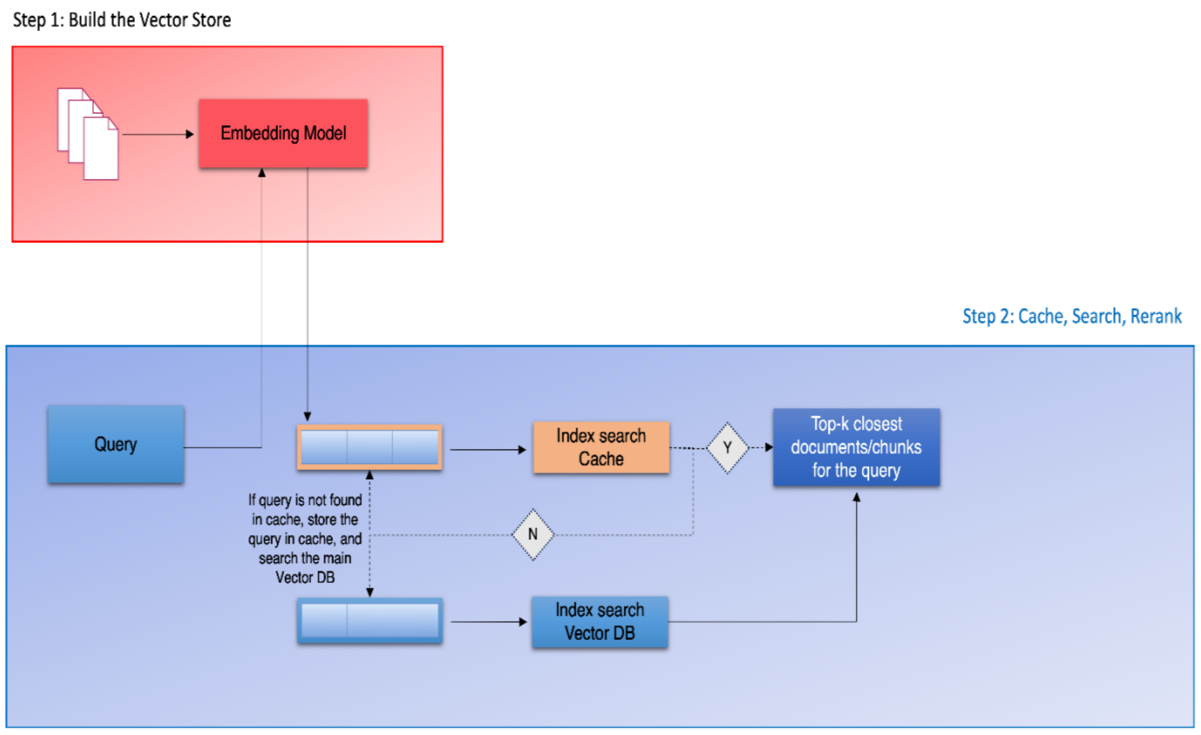


Search Layer:

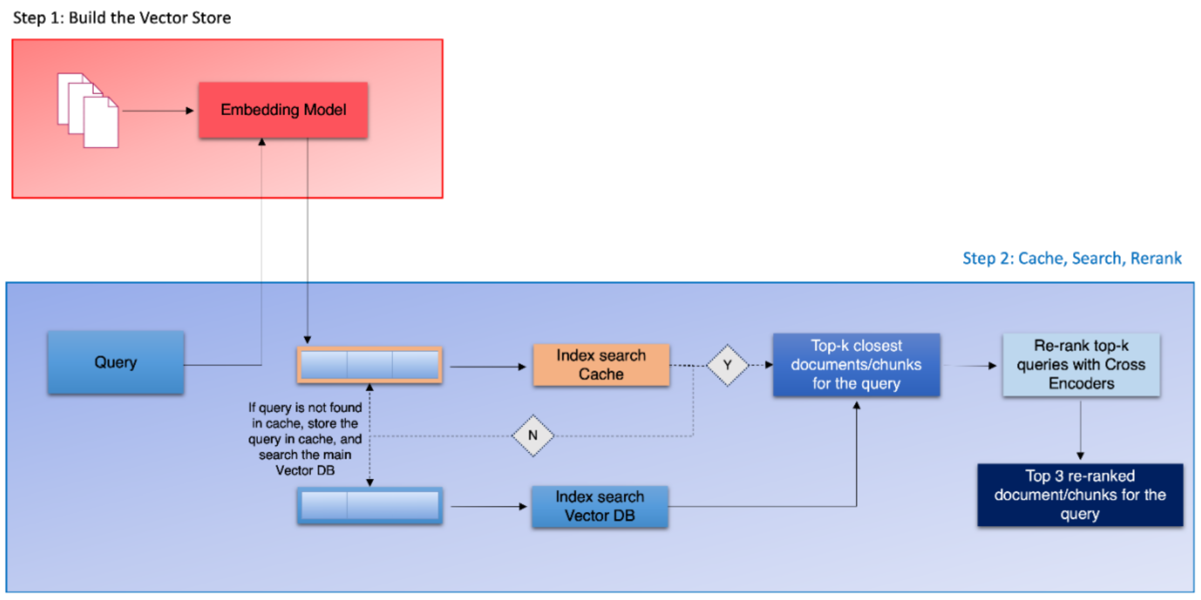
**Query Design:** Formulate at least three diverse queries that reflect potential information-seeking questions related to the policy document. Ensure these queries cover various aspects of the document to comprehensively evaluate the system's effectiveness.

**Vector Database Search:** Embed the queries and perform searches using the ChromaDB vector database. Implement a caching mechanism to efficiently store and retrieve previously processed queries and their results.

**Re-ranking Block:** Integrate a re-ranking block that uses cross-encoding models from HuggingFace to refine search results, improving their relevance and accuracy.


Step 1: Build the Vector Store
Embedding Model
Step 2: Cache, Search, Rerank
Query
Index search Cache
Y
Top-k closest documents/chunks for the query
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB

When using cross_encoder:


Step 1: Build the Vector Store
Embedding Model
Step 2: Cache, Search, Rerank
Query
Index search Cache
Y
Top-k closest documents/chunks for the query
Re-rank top-k queries with Cross Encoders
Top 3 re-ranked document/chunks for the query
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB

**Generation Layer:**

- **Prompt Design:** Develop a detailed and instructive prompt for the Language Model (LM) to ensure it effectively conveys the necessary context for coherent answer generation.

- **Few-shot Examples:** Improve the LM's performance by incorporating few-shot examples into the prompt, helping the model generate responses that are more contextually accurate.

**Overall Project Insights:**

- **Performance Evaluation:** Conduct comprehensive evaluations of each layer to assess the impact of various strategies, models, and components on the overall system performance.
- **Scalability:** Address scalability concerns by planning for increases in document volume or user queries. Implement strategies such as scaling vector databases and adjusting compute resources accordingly.
- **Documentation and Codebase:** Maintain thorough documentation of the codebase, including detailed explanations of strategies, models, and mechanisms. Provide clear guidelines for future developers or collaborators.

3. Challenges:

**Performance Scaling:** Tackle performance issues that arise with growing numbers of documents or users by utilizing vector databases and scaling computational resources.

**Cache Storage:** Enhance the cache collection to ensure efficient storage and retrieval of queries and their results.

4. Lessons Learned:

**Efficient Document Processing:** Efficiently processing PDFs is essential, with libraries like pdfplumber and well-chosen data structures being critical for effective storage and retrieval.

**Semantic Search Optimization:** Refine semantic search parameters and thresholds to achieve the best possible results.

**Cache Management:** Develop a robust cache management strategy to optimize both storage and retrieval efficiency.

5. Conclusion:

The project effectively delivers a semantic search system utilizing the RAG pipeline and a cache layer. It achieves its objectives and addresses challenges with valuable lessons for future enhancements. The system offers a scalable and efficient solution for document retrieval and information extraction.