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

1. Objectives:

The primary objectives of the project are as follows:

- Develop a semantic search system using the RAG (Embedding Layer, Search and Rank Layer, Generation Layer) pipeline for efficient document retrieval.
- Extract relevant information from PDF documents, store them in a structured format, and generate vector representations using SentenceTransformerEmbedding's all-MiniLM-L6-v2 model.
- Implement a cache layer to enhance system performance by storing and retrieving previous queries and their results.

2. Design:

2.1. RAG Pipeline:

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

Search and Rank Layer: Perform a semantic similarity search on the knowledge bank based on user queries, retrieving the top K closest documents or chunks.

Generation Layer: Utilize the results from the previous layer, including the original user query and a well-constructed prompt, to generate coherent answers using a language model.

2.2. Cache Implementation:

Set a threshold of 0.2 for semantic similarity.

Store queries and results in a cache_collection in ChromaDB for easy embedding and searching.

Use ChromaDB's utility functions to add documents, ids, and metadata to the cache_collection.

3. Implementation:

Use Google Colab for development and leverage libraries such as pdfplumber, tiktoken, openai, chromaDB, and sentence-transformers for document processing, embedding, and caching.

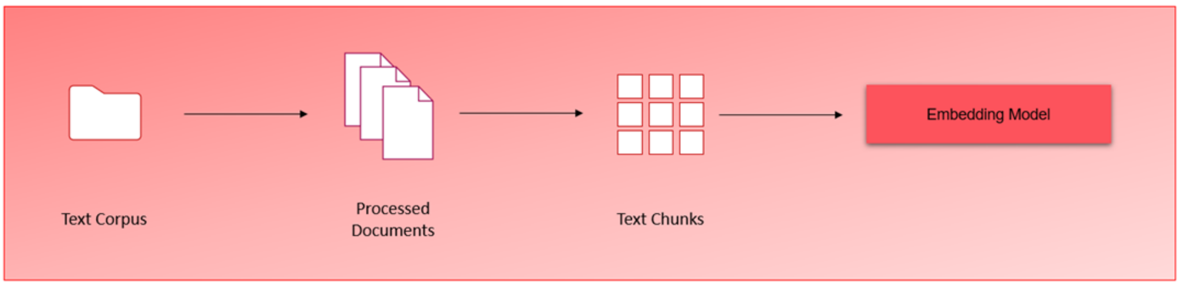
Implement functions to extract text and tables from PDFs, create a dataframe, generate vector embeddings, and perform semantic searches using the RAG pipeline.
Develop a cache system using ChromaDB to store and retrieve previous queries and their results.

The three layers for RAG pipeline are:

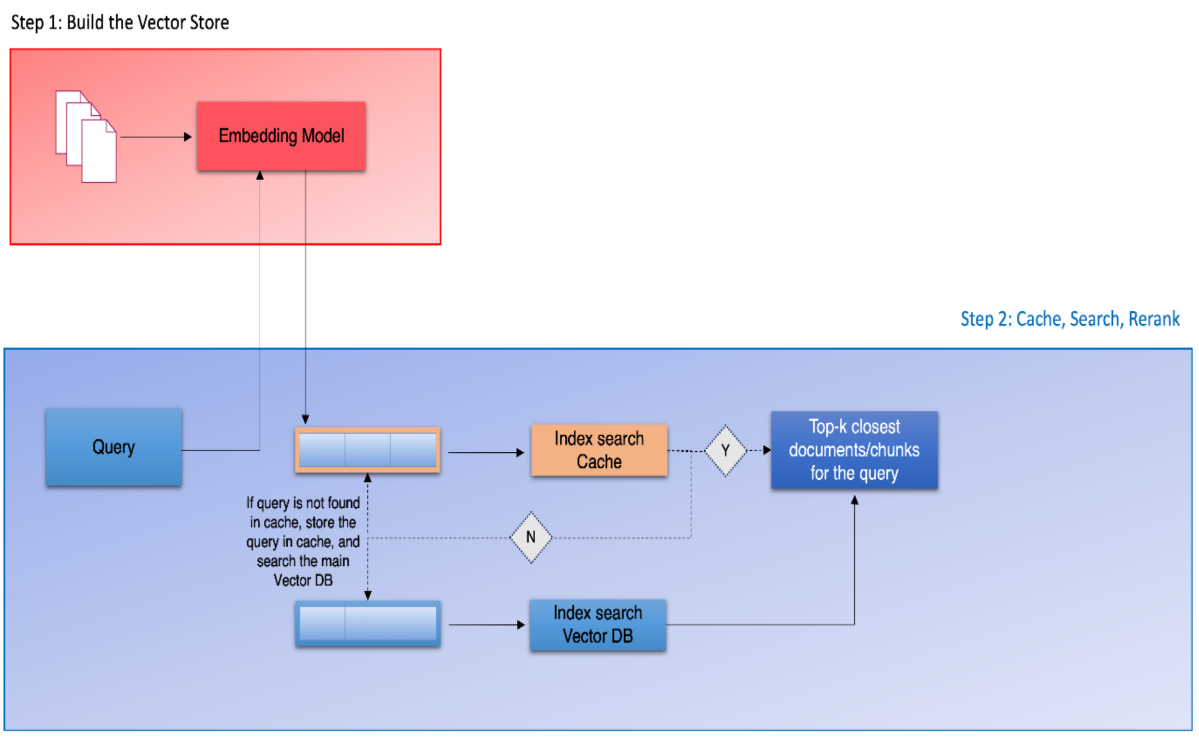
1. Embedding Layer
2. Search Layer
3. Generation Layer

Embedding Layer:
Processing and Chunking: Explore and compare various strategies for effective PDF document processing, cleaning, and chunking. Evaluate the impact of different chunking strategies on the quality of the retrieved results.
Embedding Model Choice: Choose between OpenAI's embedding model and SentenceTransformers from HuggingFace. Assess the impact of the selected model on the quality of vector representations.

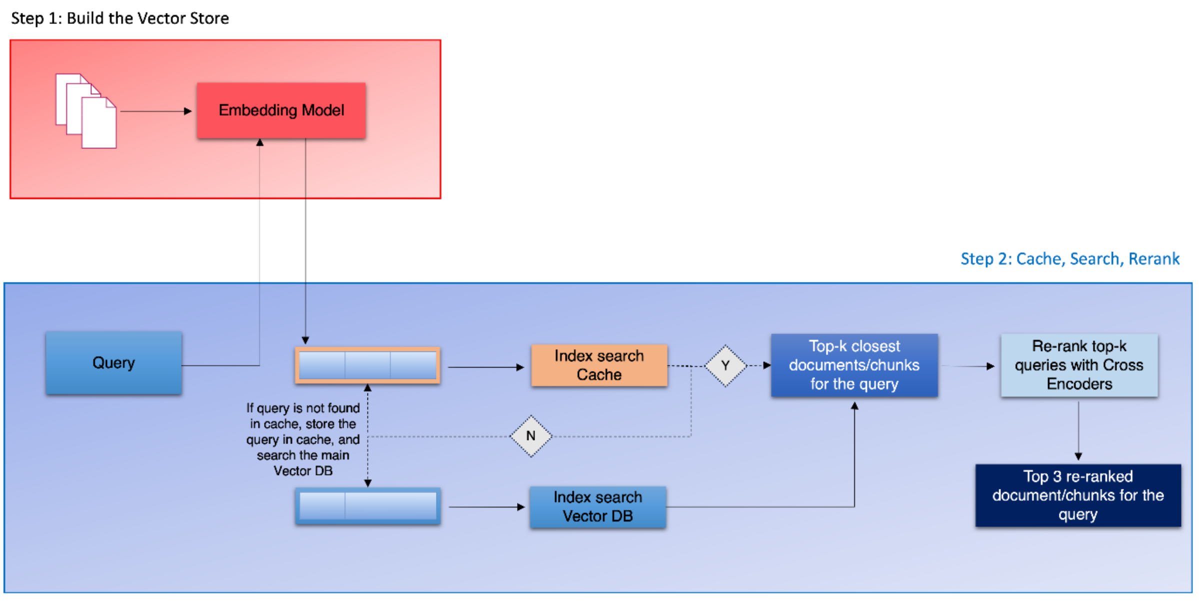
Step 1: Build the Vector Store



Search Layer:
Query Design: Design a minimum of three queries that reflect information seekers' potential questions in the policy document. Ensure queries cover diverse aspects of the document to thoroughly test the system.
Vector Database Search: Embed queries and perform searches against the ChromaDB vector database. Implement a cache mechanism to store and retrieve previous queries and their results.
Re-ranking Block: Integrate a re-ranking block utilizing cross-encoding models from HuggingFace to enhance the relevance and accuracy of search results.


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
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB
Top 3 re-ranked document/chunks for the query

Generation Layer:

Prompt Design: Focus on designing a comprehensive and instructive prompt for the Language Model (LM) in the generation layer. Ensure the prompt effectively conveys relevant information to the LM for coherent answer generation.

Few-shot Examples: Enhance LM performance by providing few-shot examples in the prompt to guide the model in generating more contextually accurate responses.

Overall Project Insights:

Performance Evaluation: Conduct thorough evaluations for each layer, considering the impact of different strategies, models, and components on the system's performance.

Scalability: Address concerns about system scalability by considering potential increases in document numbers or user queries. Implement measures such as vector database scaling and compute unit adjustments.

Documentation and Codebase: Ensure a well-documented codebase that includes detailed explanations of implemented strategies, models, and mechanisms. Provide clear instructions for potential future developers or collaborators.

4. Challenges:

Performance Scaling: Address concerns about system performance with an increased number of documents or users by implementing vector databases and scaling up compute units.

Cache Storage: Optimize the cache collection to efficiently store and retrieve queries and results.

5. Lessons Learned:

Efficient Document Processing: Processing PDFs efficiently is crucial; libraries like pdfplumber and suitable data structures for storage 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. 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.