Agentic AI Assignment 1

B.Tech 3rd Year
Semester: 6th
Session: 2026-2027

Submitted By:
Ashmit Naik (2023428345)
(CS-I G1)

Submitted To **Mr. Ayush Kumar Singh**

Faculty Designation
Assistant Professor @ Sharda University
Noida, Uttar Pradesh, India

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING SHARDA SCHOOL OF COMPUTING SCIENCE & ENGINEERING SHARDA UNIVERSITY, GREATER NOIDA

Problem Statement:

Organizations store important information in lengthy PDF documents, such as policy manuals and reports. Finding specific information from these documents manually is time-consuming and inefficient. Traditional keyword-based search methods often fail to understand the context of user queries.
Although Large Language Models can generate answers, they do not have direct access to private documents and may produce incorrect or hallucinated responses.

Therefore, there is a need for an intelligent system that can retrieve relevant information from uploaded documents and generate accurate, context-based answers using a Retrieval-Augmented Generation (RAG) approach.

Dataset / Knowledge Source:

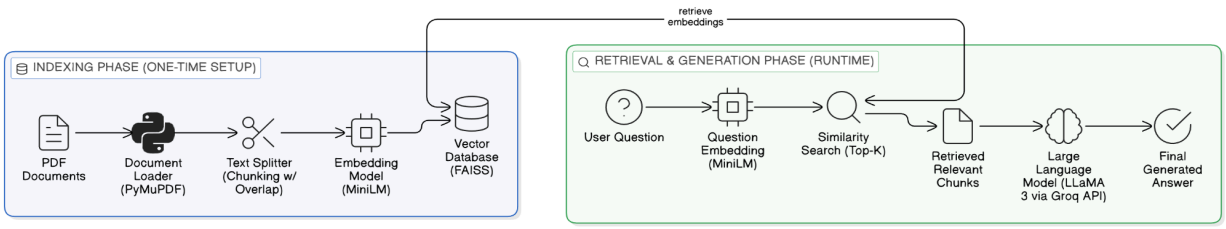
Type of Data:

The system uses unstructured text data in PDF format. The primary input consists of policy documents and manuals stored as PDF files. These documents contain textual information that is processed, split into smaller chunks, converted into embeddings, and stored in a vector database for retrieval.

Data Source:

The dataset used in this project is publicly available / self-provided policy documents. The documents are used solely for academic and research purposes to demonstrate the implementation of a Retrieval-Augmented Generation (RAG) system.

RAG Architecture:



Text Chunking Strategy:

In this project, a chunk size of 700 characters is used.

This means that the document text is divided into smaller segments where each chunk contains approximately 700 characters. This size is chosen to ensure that each chunk contains sufficient contextual information while remaining small enough for efficient embedding generation and retrieval.

A chunk overlap of 100 characters is implemented between consecutive chunks. This ensures that important contextual information at the boundaries of chunks is not lost. Overlapping helps maintain semantic continuity when the text is split.

Reason for Chosen Strategy: The selected chunk size and overlap provide a balance between context preservation and retrieval efficiency.

Larger chunks may reduce retrieval precision, while smaller chunks may lose contextual meaning. The 700-character chunk size with 100-character overlap ensures accurate semantic search and improves similarity matching in the vector database.

Embedding Details:

The project uses the Sentence Transformers model: `all-MiniLM-L6-v2` for generating embeddings.
This model converts text chunks into dense vector representations (numerical format) that capture semantic meaning. These embeddings are then stored in the FAISS vector database to enable efficient similarity-based retrieval.

Reason for selecting the model:

- It provides high-quality semantic embeddings suitable for document retrieval tasks.
- It is lightweight and computationally efficient, making it suitable for academic and small-scale deployments.
- It offers a good balance between performance and speed.
- It integrates easily with LangChain and FAISS.
- It performs well for semantic similarity search in Retrieval-Augmented Generation (RAG) systems.

This model ensures accurate matching between user queries and relevant document chunks, improving the overall effectiveness of the system.

Vector Database:

Vector Store used: FAISS

FAISS (Facebook AI Similarity Search) is used as the vector database for storing and retrieving text embeddings.

After converting document chunks into embeddings using the Sentence Transformer model, the vectors are stored in FAISS. When a user submits a query, the query is also converted into an embedding and compared against stored vectors using similarity search.

FAISS retrieves the top *k* most relevant chunks based on vector similarity, which are then passed to the Large Language Model for answer generation. (You can refer to the notebook.)

Future Improvement:

Better Chunking Strategy:
Currently, fixed-size chunking (700 characters with overlap) is used.
In future improvements, smarter chunking techniques such as semantic-based or paragraph-based chunking can be implemented.
This would ensure that chunks are split based on meaning rather than character count, improving retrieval accuracy and contextual understanding.

Reranking / Hybrid Search:
At present, similarity search retrieves the top-k chunks based purely on vector similarity. In future versions, a reranking mechanism can be added to reorder retrieved chunks based on relevance score using a cross-encoder model.
Hybrid search, which combines both keyword-based search and semantic search, can also improve accuracy by balancing exact term matching with contextual similarity.

Metadata Filtering:
Currently, all document chunks are treated equally.
In future enhancements, metadata such as document title, section name, page number, or category can be stored along with embeddings. This would allow filtered retrieval, such as searching within a specific section or document type, improving precision.

UI Integration:
The current implementation is notebook-based. A future improvement would be to integrate the system with a web-based user interface using frameworks such as Streamlit, Flask, or React. This would allow users to upload documents, ask questions interactively, and view responses in a user-friendly environment.