



## **Agentic AI LAB**

**Submitted By:**

**Aryan Raj(2023393710)**

**Ayush Nangia(2023497017)**

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

# Overview

This project implements a **Retrieval-Augmented Generation (RAG)** system for document-based question answering. The system allows users to upload a PDF document, retrieve relevant content using semantic similarity, and generate answers based strictly on the retrieved information.

The project demonstrates how modern Natural Language Processing (NLP) techniques such as **vector embeddings** and **similarity search** can be combined to build an intelligent document analysis system.

## Executive Summary

### What is this Project?

This project builds a **RAG pipeline** that enables efficient information extraction from documents. Instead of manually reading long PDFs, users can ask questions and receive answers derived from the document content.

## PROJECT GOAL

Convert document content → Generate embeddings → Retrieve relevant chunks → Build context → Generate answers

## Main Steps (Key Implementation Stages)

1. **Setup & Environment** - Install required libraries and dependencies
  2. **Load PDF Document** - Parse "How Apple is Organized for Innovation" PDF
  3. **Document Processing** - Split document into chunks for semantic processing
  4. **Embedding Generation** - Convert text chunks into vector embeddings
  5. **Vector Database Setup** - Store embeddings in FAISS for retrieval
  6. **Query Processing** - Process user queries and retrieve relevant context
  7. **Answer Generation** - Generate responses based on retrieved chunks
  8. **System Testing** - Execute test queries and validate outputs
- 

## Detailed Step-by-Step Explanation

### Dataset / Knowledge Source

### Explanation

The dataset used in this project is a **PDF document** uploaded by the user during runtime. This document serves as the **knowledge base** for the Retrieval-Augmented Generation (RAG) system. Instead of training a model on fixed data, the system dynamically extracts information from the uploaded document.

Once the PDF is uploaded, its content is extracted and processed further. This allows the system to answer queries based strictly on the document, ensuring accuracy and reducing hallucinations.

#### **Purpose of the code:**

- Allows the user to upload a PDF file
  - Stores the uploaded filename for further processing
  - Acts as the entry point of the dataset into the system
- 

## **RAG Architecture**

### **Explanation**

The Retrieval-Augmented Generation (RAG) architecture combines **information retrieval** with **answer generation**. Instead of directly generating answers, the system first retrieves the most relevant parts of the document and then uses them to generate a response.

This architecture improves reliability and transparency because answers are always grounded in retrieved document content.

### **RAG Pipeline Flow**

User Query  
→ Query Embedding  
→ FAISS Vector Search  
→ Relevant Text Chunks  
→ Context Construction  
→ Answer Generation

This modular design also makes the system easy to extend in the future.

---

## **Document Loading and Text Extraction**

### **Explanation**

After uploading the PDF file, the system extracts text from it. Each page of the PDF is read and converted into plain text. This raw text is then used for further processing such as chunking and embedding.

Text extraction is a crucial step because all downstream operations depend on clean and complete document text.

#### **Purpose of the code:**

- Opens the PDF file
  - Extracts text page by page
  - Combines all text into a single string
  - Displays a preview to verify successful extraction
- 

## **Text Chunking Strategy**

#### **Explanation**

Large documents cannot be processed efficiently as a single block of text. Therefore, the extracted text is divided into smaller overlapping chunks. Chunking improves retrieval accuracy and ensures that embeddings capture meaningful context.

Overlap is added to prevent important information from being lost between chunk boundaries.

#### **Chunking Configuration**

- **Chunk size:** 700 characters
  - **Chunk overlap:** 150 characters
- 

## **Embedding Generation**

#### **Explanation**

Embeddings are numerical representations of text that capture semantic meaning. In this project, each text chunk is converted into an embedding vector so that similar content can be retrieved using similarity search.

The same embedding model is used for both document chunks and user queries to ensure consistent comparison.

#### **Model Used**

- sentence-transformers/all-MiniLM-L6-v2
  - 384-dimensional embeddings
-

## Vector Database (FAISS)

### Explanation

FAISS (Facebook AI Similarity Search) is used to store embeddings and perform fast similarity search. It allows the system to efficiently find the most relevant document chunks for a given query.

This project uses **L2 (Euclidean) distance** for similarity comparison.

---

## Retrieval and Context Building

### Explanation

When a user submits a query, the system retrieves the most relevant chunks from FAISS. These chunks are then combined into a single context while respecting a maximum length limit.

This context is later used for answer generation.

---

## Answer Generation using RAG Pipeline

### Explanation

In this project, a **stub-based answer generation** approach is used to demonstrate the RAG concept. Instead of calling a real LLM API, the system constructs a response using the query and retrieved context.

This approach clearly shows how RAG works while keeping the system simple and API-free.

---

## Test Queries and Results

### Explanation

To verify system performance, multiple test queries are executed. These queries check factual understanding, conceptual explanation, and summarization based on document content.

#### Code Used (from your Colab)

```
test_queries = [
    "Question 1: Ask something factual based on your document.",
    "Question 2: Ask for an explanation of a concept mentioned.",
    "Question 3: Ask for a summary of a specific section or topic."
]
```

```
for i, q in enumerate(test_queries, start=1):
```

```
    print("=" * 70)
    print(f"Test Query {i}: {q}")
    print("=" * 70)
```

```
response = rag_pipeline(q, top_k=4)
print(response)
print("\n")
```