

# **Financial QA RAG Model Documentation**

# Implementation Overview

This document details the implementation of a RAG system for financial document processing, specifically focusing on P&L data analysis using FAISS as the vector database. The system processes documents through a sequence of operations that transform raw financial documents into a queryable knowledge base. Let's examine each component and its implementation details.

## Model Architecture:

### 1. Loading PDF File

The process begins by loading the desired PDF file(s). This step involves using Python libraries such as PyPDF to read the content of the PDF documents. Each PDF is loaded in its entirety to enable subsequent processing and analysis.

### 2. Extracting Text from PDF and Creating Chunks

Once the PDF file is loaded, the text is extracted using PyPDF. To ensure efficient handling and embedding of the data, the text is divided into smaller chunks. This chunking process helps maintain a balance between granularity and contextual coherence, making it easier to manage and retrieve specific segments of the content.

### 3. Converting Chunks to Embeddings

Each chunk of text is converted into a vector representation using Google embeddings. These embeddings encode the semantic meaning of the text, allowing the system to perform similarity-based retrieval tasks effectively. The embedding process ensures that the textual data is transformed into a numerical format suitable for storage and comparison.

### 4. Storing Embeddings in a Vector Database

The vector representations of the text chunks are stored in a vector database. FAISS (Facebook AI Similarity Search) is used for this purpose, offering efficient indexing and querying capabilities. This database enables fast retrieval of relevant text chunks based on their semantic similarity to a given query.

### 5. Retrieving Text Chunks Based on User Query

When a user submits a query, the system calculates the cosine similarity between the query and the stored embeddings in the vector database. The most relevant text chunks are retrieved based on their similarity scores. This retrieval step ensures that the language model has access to the most pertinent information to answer the user's question.

### 6. Sending Text Chunks and Question to the LLM

The retrieved text chunks and the user's query are sent to the language model, specifically the Gemini-1.5-Flash model, for generating a response. The model processes the provided context along with the query to deliver an accurate and contextually relevant answer.

# Challenges Faced During Development

## 1. Extracting Text from PDFs

- **Challenge:** Initially used LLAMA Parser to extract text from PDFs, but it occasionally failed to parse data from tables accurately.
- **Solution:** Switched to PyPDF for text extraction. This provided better results in extracting text but introduced other challenges when dealing with tables.

## 2. RAG System Implementation

- **Challenge:** Experimented with various RAG systems, including simple RAG, semi-structured RAG, multi-modal RAG, agentic RAG, and re-ranking chunks using both LangChain and LlamaIndex. However, none achieved satisfactory accuracy. The primary issue was the inability to extract relevant information from the vector database.
- **Solution:** After multiple attempts, revisited the initial simple RAG implementation. Surprisingly, it worked well because the chunk size was well-optimized for data extraction. I experimenting with different chunk sizes to be effective.

## 3. Inappropriate Chunks

- **Challenge:** When using PyPDF for chunk extraction, the table structure and full table content were often missing due to text splitting. This resulted in missing table headers, leading to false or incomplete answers from the LLM.
- **Solution:** To mitigate this, provided the LLM with specific instructions to understand and interpret columns better, ensuring more accurate answers.

## 4. Relevant Table Segments

- **Challenge:** The lack of a proper table structure in the extracted data from PyPDF made it difficult to identify and extract relevant table information. This led to issues when querying the vector database.
- **Solution:** Sent the extracted data and the user's question to the LLM to reconstruct and infer the table structure. This approach helped overcome the structural gaps in the extracted data.

## **Examples:**

**1. What is the total revenue from operations for the year ending March 31, 2024?**

**Answer:** The total revenue from operations for the year ending March 31, 2024, was 153,670.

**2. What was the operating profit (profit before tax minus other income) for the quarter ending March 31, 2024?**

**Answer:** The operating profit for the quarter ending March 31, 2024 was 7,621

**3. What is the carrying value of goodwill as of March 31, 2024?**

**Answer:** The carrying value of goodwill as of March 31, 2024, was 7,303.

**4. What was the depreciation and amortization expense for the year ending March 31, 2024?**

**Answer:** The depreciation expense for the year ending March 31, 2024, was ₹2,685. The provided text does not specify a separate amortization expense; the ₹2,685 figure represents depreciation only.

**5. How much cash and cash equivalents are reported as of March 31, 2024?**

**Answer:** As of March 31, 2024, the company reported ₹14,786 in cash and cash equivalents.

**6. How much was spent on the buyback of equity shares (including transaction costs and tax) for the year ending March 31, 2023?**

**Answer:** The buyback of equity shares resulted in a cash outflow of ₹9,300 crore, excluding transaction costs and tax on the buyback. The provided text does not specify the amount of transaction costs and tax, therefore the total amount spent cannot be calculated.

**7. What is the earnings per equity share (basic) for the quarter ending March 31, 2024?**

**Answer:** The earnings per equity share (basic) for the quarter ending March 31, 2024, was ₹19.25 per share.