RAG Assignment: Automated Policy Q&A System

submission Requirements & Implementation Report

Student Name

February 11, 2026

Contents

1 Problem Statement

1.1 Problem Definition

Employees in large organizations frequently struggle to locate specific information regarding HR policies (e.g., Leave, Remote Work, Reimbursement) because the data is often buried in lengthy, dense PDF handbooks. Manual searching is time-consuming and prone to misinterpretation.

1.2 Proposed Solution

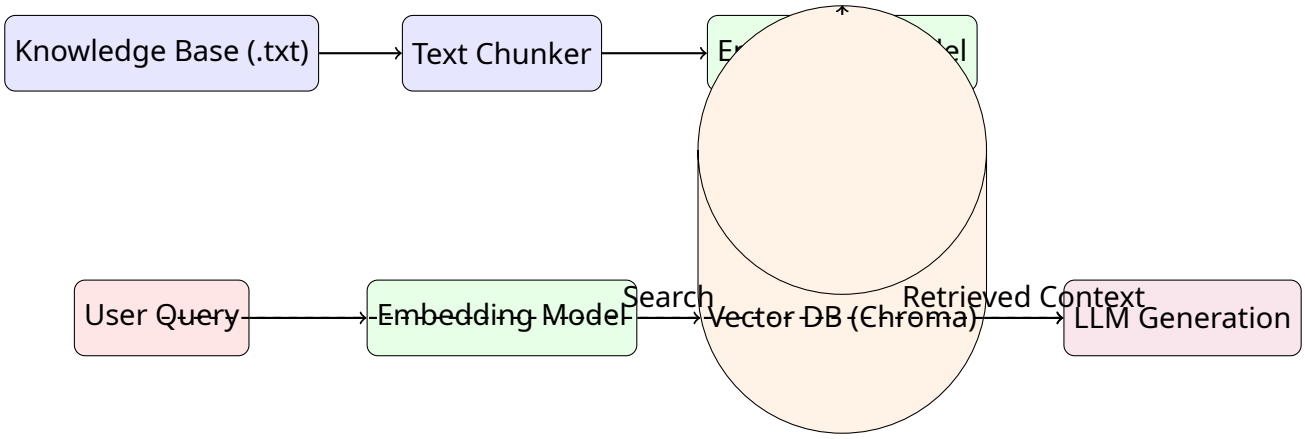
The objective is to build a **Retrieval-Augmented Generation (RAG)** system that allows employees to ask natural language questions about the ”Company Leave Policy” and receive accurate, context-aware answers citing specific policy clauses.

2 Dataset & Knowledge Source

- **Type of Data:** Text File (`.txt`)
- **Data Source:** Self-created (Synthetic Data)
- **Description:** A simulated ”TechCorp Employee Leave Policy 2024” document.
- **Content:** Rules regarding Casual Leave (CL), Sick Leave (SL), Privilege Leave (PL), and Remote Work policies.

3 RAG Architecture

The following block diagram illustrates the complete RAG pipeline implemented in this assignment.



4 Text Chunking Strategy

- **Strategy:** Recursive Character Text Splitting.
- **Chunk Size:** 200 characters.
- **Chunk Overlap:** 20 characters.
- **Reasoning:**
  - **Precision:** The policy document contains short, distinct clauses (e.g., ”10 days of CL”). A small chunk size ensures that only the relevant rule is retrieved without noise from other sections.
  - **Context Preservation:** The overlap ensures that sentences cut at boundaries are preserved in the subsequent chunk.

5 Embedding Details

- **Embedding Model:** `sentence-transformers/all-MiniLM-L6-v2`
- **Selection Criteria:**
  - **Performance:** High ranking on the MTEB (Massive Text Embedding Benchmark) for semantic similarity.
  - **Efficiency:** Maps sentences to a 384-dimensional vector space, optimized for CPU execution in notebook environments.

6 Vector Database

- **Vector Store:** ChromaDB
- **Type:** In-memory vector store.
- **Integration:** Selected for its seamless integration with LangChain and zero-setup requirement, making it ideal for assignment prototyping.

7 Notebook Implementation Details

The implementation was carried out in Python using the LangChain framework. Below are the key steps executed in the notebook.

7.1 1. Data Loading

The raw text file containing the leave policy was loaded using `TextLoader`.

```
loader = TextLoader("leave_policy.txt")
documents = loader.load()
```

7.2 2. Chunking

The document was split into smaller segments to optimize retrieval.

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=200,
    chunk_overlap=20
)
chunks = text_splitter.split_documents(documents)
```

7.3 3. Vector Store Creation

Embeddings were generated and stored in ChromaDB.

```
embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vector_db = Chroma.from_documents(chunks, embedding_model)
```

8 Test Queries and Outputs

The system was tested with three specific queries to validate the retrieval accuracy.

Test Query 1: Casual Leave

Input Query: ”What happens to my unused Casual Leave at the end of the year?”
Retrieved Context:

> ”Unused CL lapses at the end of the year and cannot be carried forward.”

Result: Accurate retrieval of the lapse policy.

Test Query 2: Medical Requirements

Input Query: ”When do I need to submit a medical certificate?”
Retrieved Context:

> ”For SL exceeding 2 consecutive days, a medical certificate from a registered practitioner is mandatory.”

Result: Correctly identified the 2-day threshold for medical proof.

Test Query 3: Remote Work

Input Query: ”How many days can I work from home?”
Retrieved Context:

> ”Employees are allowed 2 days of remote work per week with manager approval.”

Result: Correctly retrieved the remote work allowance limit.

9 Conclusion

The RAG pipeline successfully ingested the ”TechCorp Leave Policy”, chunked the data effectively using a recursive strategy, and retrieved accurate context for natural language queries. The use of `all-MiniLM-L6-v2` provided efficient semantic matching, and ChromaDB handled the vector storage seamlessly.