Guru Nanak Dev Engineering College

Training Diary – TR-102 Report

Name: Divanshi Goyal
URN: 2302513
CRN: 2315056
Day 11

Training Summary

On the eleventh day of training, we studied and implemented **Retrieval-Augmented Generation (RAG)**—an advanced technique that combines document retrieval with generative language models to provide context-aware responses. We worked on creating persistent chunk-based knowledge systems using ChromaDB, OpenAI's gpt-3.5-turbo, and GenAI for querying both **single and multiple PDFs**.

Understanding RAG (Retrieval-Augmented Generation)

RAG enhances LLMs by injecting **external knowledge** into the prompt pipeline. Instead of relying only on model memory, it retrieves relevant information from documents and then generates a tailored response.

Core Components of RAG:

1. **Indexing:**
   - Converts text data into numerical representations called **embeddings**.
   - Helps organize and store information in vector databases.
2. **Retrieval:**
   - Searches relevant documents or chunks based on a user query.
   - Uses similarity scoring to fetch the most appropriate content.
3. **Augmentation & Generation:**
   - Fetched content is used as **context** in the prompt.
   - The model then generates a well-informed answer using this added context.

Hands-On: RAG with Multiple PDFs

We followed a step-by-step process to implement RAG with OpenAI's GPT and ChromaDB:

Steps Performed:

- Used gpt-3.5-turbo to **create clean and manageable text chunks** from the PDF.

- Stored those chunks in **ChromaDB**, a lightweight vector database, for persistent and efficient querying.
- Provided a **user query**, retrieved relevant context using embedding similarity, and passed both query + context to the GenAI model for a coherent response.
- Supported both **single** and **multiple PDF** sources, allowing broader knowledge coverage.

This allowed us to simulate building a **personal document Q&A system**, enabling efficient interaction with large sets of content.

Key Tools & Libraries Used

- **OpenAI gpt-3.5-turbo –** for generation and chunking
- **ChromaDB –** for storing vector embeddings and retrieval
- **Langchain / GenAI functions –** for managing context + prompt injection
- **PDF loader –** for reading and chunking data

Learning Outcome

By implementing RAG, we gained a practical understanding of:

- How **indexing, retrieval, and augmentation** work together to boost LLM accuracy.
- Chunking text for optimal embedding and persistent storage.
- Building systems that can interact with PDFs and respond to queries meaningfully.
- Using vector databases (like ChromaDB) for scalable retrieval solutions.