

(TR-103) PROMPT ENGINEERING –

Training Day 12 Report:

Retrieval-Augmented Generation (RAG) Pipeline

RAG stands for **Retrieval-Augmented Generation**, a pipeline designed to produce precise and context-based responses by combining document retrieval with generative AI models.

The RAG system is highly relevant in scenarios where even the smallest error or assumption cannot be tolerated, such as in legal, medical, or scientific applications. It acts as a **personalized agent**, generating fact-based answers from user-provided documents rather than relying on general training knowledge.

Stages of RAG Pipeline:

1. Indexing

- The input document (e.g., textbook, article, or PDF) is divided into smaller, meaningful text units called **chunks**.
- Each chunk is converted into a **numerical vector** using a **sentence embedding model**.
- These vectors represent the semantic meaning of each chunk.
- The chunks and their corresponding vectors are stored in a **vector database** (e.g., ChromaDB) to enable fast and meaningful search.

2. Retrieval

- The user's question is also converted into a **vector** using the same embedding model.
- The vector database compares this question vector with the stored chunk vectors.
- It identifies and retrieves the **most semantically relevant chunks** from the document.
- These retrieved chunks contain the most useful information related to the question.

3. Augmentation

- The retrieved chunks are combined with the user's original question to form a **single, structured prompt**.
- This prompt provides the necessary background and context from the document.
- The purpose is to **guide the language model** effectively by giving it relevant content to base its answer on.

4. Generation

- The constructed prompt is sent to a **generative language model** (e.g., Google Gemini or ChatGPT).
- The model processes the prompt, reads the context, understands the question, and generates a **clear, relevant, and accurate** answer.
- The output is a **human-like response** based entirely on the retrieved and relevant content.

Technical Highlights

- **Persistent Storage:** Ensures that once chunks are created and indexed, they can be reused without reprocessing.
- **Overlapping Chunks:** Chunks are created with overlaps (e.g., 1–100, 50–150) to maintain continuity between related sections.
- **Semantic Search:** The retrieval is based on meaning, not keywords, which improves relevance and accuracy.
- **Relevance First:** The main goal is always relevance and factual correctness — assumptions are avoided entirely.

Task: Implementation of RAG Pipeline on a Single PDF Document.


Steps:

- Upload the PDF document
- Extract and clean the text content.
- Split the text into smaller chunks.
- Generate embeddings for each chunk using a sentence transformer.
- Store the chunks and embeddings in ChromaDB.
- Accept user query input.

- Retrieve the top relevant chunks using semantic search.
- Combine the query and retrieved chunks to form the prompt.
- Send the prompt to a language model (e.g., Gemini or ChatGPT).
- Generate and display the final answer.

Output:

```
[ ] query = "What are the key topics discussed in the document?"  
    answer = generate_answer(query)  
    print("Answer:\n", answer)
```

 Answer:

Based on the provided text, the key topics discussed are:

- * **Machine Learning:** The text explicitly mentions machine learning and the intended audience includes students pursuing doctoral research i
- * **Book Writing/Content:** The text focuses on the creation of a book on machine learning. It describes the book's target audience, guiding p

Task: Implementation of RAG Pipeline on Multi-PDF Document.

Steps:

- Upload the PDF document
- All uploaded PDFs were read in a loop.
- Text from each file was extracted, chunked, and embedded.
- All embeddings were added to a shared vector store, allowing unified retrieval across all documents.
- This enabled users to ask questions that could span multiple files or topics, simulating a personalized knowledge base.