Submitted by: Dharansh Neema
Email: dharanshneema@gmail.com

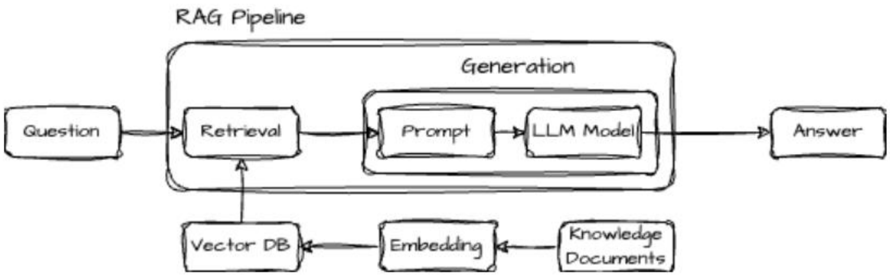
**Task 2 - Optimising RAG:** *

Detail two innovative techniques for optimising the RAG model developed in Task 1.

Task 2 should be submitted in **PDF Format.**

## Solution:

*Since RAG pipeline look like this:*




We can optimize each of this component to optimize our RAG i.e. from retrieval to generation.

## 1st Technique: CHUNKING the document

1. Chunking can play very crucial role while optimizing the RAG because we don't have infinite tokens, so chunking can be useful to reduce the **token head.**
2. All the embedding model have a fixed sized context window either (example:512 tokens), if the passed document is greater than that we started losing data, so chunking the document before embedding can prevent data loss.

**Chunking Methods:**

1. **Fixed Size Chunking:** divide the text into chunks of a fixed number of tokens, words, or character.
   Example: chunks = [text[i:i+300] for i in range(0, len(text), 300)]
   # 300 tokens
2. **Sentence-Based Chunking:** Split the document into chunks based on complete sentences. We can NLP framework NLTK to do so
   Example :from nltk.tokenize import sent_tokenize

sentences = sent_tokenize(text)

3. **Paragraph-Based Chunking:** Split the text by paragraphs, treating each paragraph as a chunk.
   Example : paragraphs = text.split("\n\n")

4. **Dynamic Chunking with Overlap:** Create chunks of a fixed size with overlapping tokens to preserve context between adjacent chunks.
   Example:
   overlap = 50
   chunk_size = 300
   chunks = [text[i:i+chunk_size] for i in range(0, len(text), chunk_size - overlap)]

5. **Semantic Chunking:** Use NLP techniques to divide the text into semantically coherent chunks.
   Example : from transformers import pipeline
   summarizer = pipeline("summarization")
   chunks = summarizer(text, max_length=300, min_length=100)

## 2$^{nd}$ Technique: Fine Tuning:

Improve the generation step by fine-tuning the LLM on domain-specific knowledge.

**Develop Training Data for fine tuning**

1. Manual prompt: Use the help of any Expert of that domain for evaluating and preparing the training data.
2. Programmatically prompt : Use metadata to scale larger training set.
3. Generate Synthetic prompt: Take all chunks and use and an LLM to synthesize prompts.

**Fine-Tuning the Embedding Set**

Embedding models are responsible for converting textual data into dense vector representations. Fine-tuning the embedding model for your specific use case can enhance retrieval accuracy by aligning embeddings with domain-specific semantics.

**Fine-Tuning Embedding Models:**

1. **Curate Training Data**:

    1. **Positive Examples**: Pairs of text chunks that should be similar in embedding space (e.g., movie descriptions of the same genre).

    2. **Negative Examples**: Pairs of text chunks that should be dissimilar (e.g., Sci-Fi vs. Comedy movie plots).

2. **Model Training**:
    1. Use techniques like contrastive learning or triplet loss to adjust the embedding space.
    2. Train on domain-specific datasets to optimize embedding quality.
3. **Evaluation**:
    1. Test embeddings by running similarity queries and comparing the results to expected outputs.
    2. Refine the model as needed.

**Benefits:**

1. Improves retrieval precision for queries in your domain.
2. Reduces noise by ensuring embeddings capture the most relevant features.