Question Answer

Question: What method or library did you use to extract the text, and why? Did you face any formatting challenges with the PDF content?
Answer:

I used the pytesseract library to extract text from the PDF. This library provides a Python wrapper for Google's Tesseract-OCR engine, which is effective for OCR (Optical Character Recognition) on scanned documents or image-based PDFs.

However, I faced some formatting challenges, especially with Bangla text. Some words were mismatched or altered during the extraction process. This is a common issue when using OCR with complex scripts like Bangla due to limitations in character recognition accuracy, font styles, and image quality in the source PDF.

To improve the results, techniques like preprocessing the image (e.g., binarization, resizing, or using the correct --oem and --psm settings) or using a language-trained model specific to Bangla can be helpful.

2. What chunking strategy did you choose (e.g. paragraph-based, sentence-based, character limit)? Why do you think it works well for semantic retrieval?
I applied two different chunking strategies to process the text for semantic retrieval:
Answer:

1. Fixed-size chunking with overlap
Chunk size: 1000 characters
Overlap: 200 characters
In this method, the text is split into chunks of 1000 characters, with a 200-character overlap between consecutive chunks.

Why it works well:
This approach ensures that important contextual information that may span across chunk boundaries is preserved, which improves the quality of semantic matching during retrieval. Overlapping helps maintain continuity in meaning and reduces the chance of cutting off important phrases or sentences.

2. Sentence-based chunking
In this method, the text is split based on sentence boundaries using a sentence tokenizer.
Answer:
Why it works well:
Sentence-based chunking aligns closely with natural language structure. It allows embeddings to capture the full meaning of each sentence, making it suitable for precise question-answering and semantic understanding. This method is especially effective for shorter documents or when each sentence contains a complete idea.

3. What embedding model did you use? Why did you choose it? How does it capture the meaning of the text?
Answer:

I used the models/embedding-001 embedding model via LangChain with PGVector as the vector store. This model is provided by Google through the langchain_google_genai integration.

I chose this model:
Strong multilingual support: models/embedding-001 is known for its ability to handle multiple languages effectively, including Bangla.

High-quality semantic representation: It captures the semantic meaning of text well, even for non-English content, making it suitable for Bangla-based semantic search and retrieval tasks.

Easy integration: The model works seamlessly with LangChain's embedding interface and PGVector, allowing efficient storage and retrieval using vector similarity search.

How it captures the meaning of the text:
The model converts text into high-dimensional vector embeddings that preserve semantic relationships. Texts with similar meanings generate similar vector representations, even if the exact words or phrases differ. This enables accurate retrieval of relevant information based on meaning rather than exact keyword matching.

4 How are you comparing the query with your stored chunks? Why did you choose this similarity method and storage setup?
Answer:

To compare the user query with the stored chunks, I use vector similarity search powered by PGVector in a Neon PostgreSQL database.

How the comparison works:
Embedding generation:
The input query is embedded using the same embedding model (models/embedding-001) that was used to embed the document chunks.
Similarity search:
The query embedding is then compared with the stored chunk embeddings using cosine similarity. PGVector allows efficient vector similarity search directly within the PostgreSQL database.

Top-k retrieval:
The most similar chunks (top-k) are retrieved based on their similarity scores for downstream tasks like question-answering or summarization.

5 How do you ensure that the question and the document chunks are compared meaningfully? What would happen if the query is vague or missing context?
Answer:

To ensure meaningful comparison between the question and document chunks:
I use the same embedding model (models/embedding-001) for both queries and chunks to represent them in the same semantic space.
I apply context-aware chunking strategies (fixed-size with overlap and sentence-based) to preserve meaning within each chunk.
Cosine similarity is used to find the most semantically relevant chunks during retrieval.
If the query is vague or lacks context, the results may be less accurate. In such cases, solutions include:
Asking the user for a more specific query,
Using conversational history to provide context,

Or applying fallback responses to guide clarification.