Assignment 03 — Google NotebookLM RAG

Problem Statement

Build your own version of **Google NotebookLM** — a RAG-powered application where a user can upload any document and have a conversation with it.

The user uploads a file. The system processes it, stores it intelligently, and allows the user to ask natural language questions — getting back answers grounded in the document's actual content.

What You Are Building

A full RAG pipeline with a working interface where:

- A user can upload a document (PDF or plain text)
- The system chunks, embeds, and indexes the document into a vector database
- The user can ask questions about the document
- The system retrieves the most relevant chunks and generates a grounded answer using an LLM
- Answers must come from the document — not from the LLM's general knowledge

What is Expected

- A working application — CLI or simple web UI, your choice
- The full RAG pipeline must be implemented end to end: ingestion → chunking → embedding → storage → retrieval → generation
- At least one chunking strategy must be implemented and clearly documented
- A vector database must be used for storing and retrieving embeddings
- The LLM must use retrieved context to answer — not answer from memory alone
- The app must handle a document it has never seen before and answer questions correctly from it

Submission

Submit both of the following on the course portal:

1. **GitHub Repository Link** — must be public
2. **Live Project Link** — deployed and accessible without any local setup

Submissions missing either will not be evaluated.

Marking Scheme — 10 Points

Criterion	Marks
GitHub Repository	2
Live Project	2
RAG Pipeline (chunking → embedding → retrieval → generation)	3
Answer Quality — grounded in document, not hallucinated	2
Code Quality & Documentation	1

Code File 👏

```
import "dotenv/config";
import { PDFLoader } from "@langchain/community/document_loaders/fs/pdf"
import { OpenAIEmbeddings } from "@langchain/openai";
import { QdrantVectorStore } from "@langchain/qdrant";
import { OpenAI } from "openai";

const filePath = "./node-js.pdf"

async function indexing() {
   const loader = new PDFLoader(filePath);

   // chunking
   const docs = await loader.load();

   // embeddings
```

```
    const embeddings = new OpenAIEmbeddings({
        model: "text-embedding-3-large",
    });

    const vectoreStore = await QdrantVectorStore.fromDocuments(docs, embeddings, {
        url: "http://localhost:6333",
        collectionName: "SEC-B"
    });

    console.log("Indexeding Completed")

}

// indexing();

async function retrival() {
    const userQuery = "Explain me how to do debugging in node js and also provide me some examples";

    // embeddings
    const embeddings = new OpenAIEmbeddings({
        model: "text-embedding-3-large",
    });

    const vectoreStore = await QdrantVectorStore.fromExistingCollection(embeddings, {
        url: "http://localhost:6333",
        collectionName: "SEC-B"
    });

    const retrival = await vectoreStore.asRetriever({
        k: 3
    });

    const searchedChunks = await retrival.invoke(userQuery);

    const client = new OpenAI();

    const system_prompt = `You are an AI Assistant who helps resolving the user query based on the avaliable context provided to you from PDF file with the content and page number.

        Rule :
```

```
        - Only answer based on the avaliable context from the file only.

        context : ${JSON.stringify(searchedChunks)}`

    const response = await client.chat.completions.create({
        model : 'gpt-4.1-mini',
        messages : [
            {
                role : 'system',
                content : system_prompt
            },
            {
                role : 'user',
                content : userQuery
            }
        ]
    });

    console.log(response.choices[0].message.content);

}

retrival();
```