AI Engineering Project

Project Overview

For this project, you will be designing, building, and evaluating a Retrieval-Augmented Generation (RAG) LLM-based application that answers user questions about a corpus of company policies & procedures. You will then deploy the application to a free-tier host (e.g., Render, Railway) with a basic CI/CD pipeline (e.g., GitHub Actions) that triggers deployment on push/PR when the app builds successfully. Finally, you will demonstrate the system via a screen-share video showing key features of your deployed application, and a quick walkthrough of your design, evaluation and CI/CD run. You can complete this project either individually or as a group of no more than **three** people.

While you can fully hand code this project if you wish, you are highly encouraged to utilize leading AI code generation models/AI IDEs/async agents to assist in rapidly producing your solution, being sure to describe in broad terms how you made use of them. Here are some examples of very useful AI tools you may wish to consider. You will be graded on the quality and functionality of the application and how well it meets the project requirements—no given proportion of the code is required to be hand coded.

Learning Outcomes

When completed successfully, this project will enable you to:

- Demonstrate excellent AI engineering skills
- Demonstrate the ability to select appropriate AI application design and architecture
- Implement a working LLM-based application including RAG
- Evaluate the performance of an LLM-based application
- Utilize AI tooling as appropriate

Project Description

First, assemble a small but coherent **corpus** of documents outlining company policies & procedures - about 5–20 short markdown/HTML/PDF/TXT files totaling 30–120 pages. You may author them yourself (with AI assistance) or use policies that you are aware of from your own organization that can be used for this assignment. Students must use a corpus they can legally include in the repo or load at runtime (e.g., your own synthetic policies, your organization's employee policy documents etc.)—no private/paid data is required. Additionally, you should define **success metrics** for your application (see the "Evaluation" step below), including at least one information-quality metric (e.g., groundedness or citation accuracy) and one system metric (e.g., latency).

Use free or zero-cost options when possible e.g., OpenRouter's free tier (https://openrouter.ai/docs/api-reference/limits), Groq (https://console.groq.com/docs/rate-limits), or your own paid API keys if you have them. For embedding models, free-tier options are available from Cohere, Voyage, HuggingFace and others

Complete the following steps to fully develop, deploy, and evaluate your application:

1. Environment and Reproducibility
    - Create a virtual environment (e.g., venv, conda).
    - List dependencies in requirements.txt (or environment.yml).
    - Provide a README.md with setup + run instructions.
    - Set fixed seeds where/if applicable (for deterministic chunking or evaluation sampling).
2. Ingestion and Indexing
    - Parse & clean documents (handle PDFs/HTML/md/txt).
    - Chunk documents (e.g., by headings or token windows with overlap).
    - Embed chunks with a free embedding model or a free-tier API.
    - Store the embedded document chunks in a local or lightweight **vector database** (e.g. Chroma or optionally a cloud-hosted vector store like Pinecone, etc.)
    - Store vectors in a local/vector DB or cloud DB (e.g., Chroma, Pinecone, etc.)
3. Retrieval and Generation (RAG)
    - To build your **RAG pipeline** you may use frameworks such as LangChain to handle retrieval, prompt chaining, and API calls, or implement these manually.
    - Implement Top-k retrieval with optional re-ranking.

- Build a prompting strategy that injects retrieved chunks (and citations/sources) into the LLM context.
- Add basic guardrails:
  - Refuse to answer outside the corpus (“I can only answer about our policies”),
  - Limit output length,
  - Always cite source doc IDs/titles for answers.

4. Web Application
   - Students can use Flask, Streamlit or alternative for the Web app. LangChain is recommended for orchestration, but is optional.
   - **Endpoints/UI:**
     - / - Web chat interface - text box for user input
     - /chat - API endpoint that receives user questions (POST) and returns model-generated answers with citations and snippets (link to source and show snippet).
     - /health - returns simple status via JSON.
5. Deployment
   - For production hosting use Render or Railway free tiers; students may alternatively use any other free-tier providers of their choice.
   - Configure environment variables (e.g. API keys, model endpoints, DB related etc.).
   - Ensure the app is publicly accessible at a shareable URL.
6. CI/CD
   - Minimal automated testing is sufficient for this assignment (a build/run check, optional smoke test).
   - Create a **GitHub Actions** workflow that on push/PR:
     - Installs dependencies,
     - Runs a build/start check (e.g., python -m pip install -r requirements.txt and python -c "import app" or pytest -q if you add tests),
     - On success in main, deploy to your host (Render/Railway action or via webhook/API).
7. Evaluation of the LLM Application
   - Provide a **small evaluation set** of 15–30 questions covering various policy topics (PTO, security, expense, remote work, holidays, etc.). Report:
     - Answer Quality (required):
       1. Groundedness: % of answers whose content is factually consistent with and fully supported by the retrieved evidence—i.e., the answer contains no information that is absent or contradicted in the context.

2. Citation Accuracy: % of answers whose listed citations correctly point to the specific passage(s) that support the information stated—i.e., the attribution is correct and not misleading.
3. Exact/Partial Match (optional): % of answers that exactly or partially match a short gold answer you provide.

- System Metrics (required):
  1. Latency (p50/p95) from request to answer for 10–20 queries.
- Ablations (optional): compare retrieval k, chunk size, or prompt variants.

8. Design Documentation
   - Briefly justify **design choices** (embedding model, chunking, k, prompt format, vector store).

Submission Guidelines

Your final submission should consist of **two links:**

- A link to an accessible **software repository** (a GitHub repo) containing all your developed code and the items listed below. You must share your repository with the GitHub account, **quantic-grader**.
  - The GitHub repository should include a link to the **deployed version** of your RAG LLM-based application (in file deployed.md)
  - The GitHub repository must include a README.md file indicating setup and run instructions
  - The GitHub repository must also include a brief **design and evaluation document** (design-and-evaluation.md) listing and explaining:
    i) design and architecture decisions made - and why they were made, including technology choices
    ii) summary of your evaluation approach and results for your RAG system
  - The GitHub repository must include an ai-use.md file that briefly describes which AI code tools you used and how.
- A link to a recorded screen-share **demonstration video** of the working RAG LLM-based application, involving screen capture of it being used with voiceover
  - All group members must speak and be present on camera.
  - All group members **must show their government ID**.
  - The demonstration/presentation should be between 5 and 10 minutes long.

To submit your project, please click on the "Submit Project" button on your dashboard and follow the steps provided. If you are submitting your project as a group, **please**

Project Rubric

Scores 2 and above are considered passing. Students who receive a 1 or 0 will not get credit for the assignment and must revise and resubmit to receive a passing grade.

Score	Description
5	• Addresses ALL of the project requirements, but not limited to: ○ Outstanding RAG application with correct responses with matching citations, ingest and indexing works ○ Excellent, well-structured application architecture ○ Public deployment on Render, Railway (or equivalent) fully functional ○ CI/CD runs on push/PR and deploys on success ○ Excellent documentation of design choices. ○ Excellent evaluation results, which includes groundedness, citation accuracy, and latency ○ Excellent, clear demo of features, design and evaluation
4	• Addresses MOST of the project requirements, but not limited to: ○ Excellent RAG application with correct responses with generally matching citations, ingest and indexing works ○ Very good, well-structured application architecture ○ Public deployment on Render, Railway (or equivalent) almost fully functional ○ CI/CD runs on push/PR and deploys on success ○ Very good documentation of design choices. ○ Very good evaluation results which includes groundedness, citation accuracy, and latency ○ Very good, clear demo of features, design and evaluation
3	• Addresses SOME of the project requirements, but not limited to: ○ Very good RAG application with mainly correct responses with generally matching citations, ingest and indexing works ○ Good, well-structured application architecture ○ Public deployment on Render, Railway (or equivalent) almost fully functional ○ CI/CD runs on push/PR and deploys on success ○ Good documentation of design choices.

	○ Good evaluation results which includes most of groundedness, citation accuracy, and latency ○ Good, clear demo of features, design and evaluation.
2	• Addresses FEW of the project requirements, but not limited to: ○ Passable RAG application with limited correct responses with few matching citations, ingest and indexing works partially ○ Passable application architecture ○ Public deployment on Render, Railway (or equivalent) not fully functional ○ CI/CD runs on push/PR and deploys on success ○ Passable documentation of design choices. ○ Passable evaluation results which includes only some of groundedness, citation accuracy, and latency ○ Passable demo of features, design and evaluation
1	• Addresses the project but MOST of the project requirements are missing, but not limited to: ○ Incomplete app; not deployed, ○ No CI/CD, ○ No to very limited evaluation ○ No design documentation ○ No demo of application
0	• The student either did not complete the assignment, plagiarized all or part of the assignment, or completely failed to address the project requirements.