SYSTEMS DESIGN – TAKE-HOME ASSIGNMENT

Congratulations on advancing through the previous rounds! In this assignment, you will demonstrate your hands-on abilities by implementing a simple information retrieval system with a focus on deployment and scalability. Please take some time to work on this. The system design interview will be a discussion of your assignment.

INSTRUCTIONS

- In this assignment, you would be deploying a question-answering system with focus on two areas,
    1. backend setup (Vector DB, embedding, and ingestion), and
    2. a question-answering API.
- Feel free to use LLMs to expedite your work.
- Assume any missing information that you might require to successfully finish the assignment.
- The assignment has been kept open ended on purpose.
- Avoid using platforms or packages that require payment; try to rely on open-source solutions.
- Aim to complete the entire assignment on your personal device.
- Test your solution using dummy data.
- Share you completed assignment via a Git platform (e.g., GitHub, GitLab, etc.) of your choice.

PART 1

MUST-HAVES

- Create a Python pipeline to parse, embed, and index PDF documents in a vector database.
- Use open-source, lightweight, in-memory embedding libraries (e.g., Universal Sentence Encoder) and vector databases (e.g., Chroma DB).

GOOD TO HAVE

- Reusable and modular code.

BONUS POINTS

- Create an index with multiple fields and use complex queries that utilize vector embeddings and text fields together to perform hybrid search (lexical + semantic) in one go.

PART 2

MUST-HAVES

- Create an API to query the index built in Part 1, to enable question answering.
- Implement data validation and error handling in the API.
- Improve API performance using async-wait, multithreading, or a parallel processing method of your choice.

GOOD TO HAVE

- Setup a CI pipeline on a free Git platform (e.g., GitHub, GitLab) with at least one dummy test.
- Use the dummy test to generate a release branch via the CI pipeline.
- Analyze average latency for the various steps in your API.

BONUS POINTS

- Use Locust (or a similar tool) to estimate the optimal throughput and latency of the API.
- Fine tune the degree of parallelization from **"Must-Haves"** to maximize throughput while minimizing (or holding) latency and error rates.

Happy Coding! We look forward to discussing your solution soon!