AI Infrastructure Engineer - Coding Challenge

Overview

Welcome to the AI Infrastructure coding challenge! This challenge is designed to assess your ability to build an end-to-end product matching system that leverages Vector Databases, NoSQL Databases, Visual Language Models (VLMs) and Vision Foundation Models served using NVIDIA Triton Inference Server.

You are expected to mock a database, implement model quantization, deploy a model, and design a full matching pipeline that can efficiently retrieve the closest product match for a given input image.

Task Description

Your task is to create a product matching pipeline where an input image is compared against stored products in a vector database and a MongoDB-based metadata store to find the closest match.

Requirements

1. Mock a Vector Database (e.g., Qdrant, FAISS, Weaviate, Chroma, or Pinecone)
   - Store product embeddings (textual and visual) in the database.
   - Allow efficient nearest neighbor search for retrieval.
2. Mock a MongoDB Database to Store Product Metadata
   - Store product images and metadata (e.g., name, category, price).
   - Implement necessary query functions.
3. Quantize a Visual Language Model (VLM) using TensorRT
   - Select a VLM (preferably the popular ones like LLaVA, QwenVL, InternVL, Idefics …).
   - Select a text encoder model (e.g. BERT, SentenceEncoder, GPT….) to
   - Select a vision encoder model (e.g. DINOv2, CLIP, ….)
   - Perform model quantization/compilation using TensorRT.
4. Deploy the Quantized Model Using NVIDIA Triton Inference Server
   - Serve the quantized model via an HTTP endpoint.
   - Ensure it supports both image and text inputs.
5. Build the Product Matching Pipeline
   - Given an input image, extract text+visual embeddings using the deployed model as well as DB info.
   - Perform a nearest neighbor search in the vector database.
   - Retrieve the best match along with metadata from MongoDB.
6. Mock a MongoDB for Logging
   - Store logs, errors, and execution results.
   - Ensure error handling and tracking.

Bonus Points

- Use Docker to containerize the application.
- Optimize latency by applying batching or caching strategies.
- Implement multi-threading or async processing for faster inference.
- Write clean, modular code with API documentation.

Submission Guidelines

- Provide a GitHub repo with a README explaining:
- Include sample product data in the vector DB and MongoDB.
- (Optional) Provide a live demo using a cloud-hosted instance, maybe via Gradio or any WebUI.

Evaluation Criteria

- Architecture Design – How well you structure the components.
- Scalability & Efficiency – Use of optimized inference and retrieval.
- Code Quality – Clean, modular, and well-documented code.
- Correctness – The system correctly matches products.
- Robustness – Proper handling of errors and logging.