Report for LLM-Powered QA System

Implementation Choices:

FastAPI Framework:

Choice: Used FastAPI for its async support, automatic OpenAPI docs, and Pydantic validation, ideal for a scalable API serving LLM and analytics.

Reason: Simplifies endpoint definition, handles JSON parsing/validation (e.g., QuestionRequest), and supports high concurrency for future scaling.

LLMInterface:

Choice: generate_response, relying on VectorStore context and LLM inference.

Reason: Meets your requirement for flexibility users can ask any question, and the LLM interprets dataset context dynamically, avoiding rigid precomputed answers.

Vector Store with ChromaDB:

Choice: Used ChromaDB with SentenceTransformer embeddings (all-MiniLM-L6-v2) to store yearly/monthly summaries and booking details.

Reason: Enables retrieval-augmented generation (RAG) by providing relevant context to the LLM, balancing speed and accuracy over raw DataFrame queries.

JSON Logging:

Choice: Appended /ask responses to responses.json with response times.

Reason: Provides a persistent record of user interactions, useful for debugging, auditing, or later analysis, while keeping the API response lightweight.

Analytics with Visuals:

Choice: Precomputed aggregates (e.g., revenue trends) and included a base64-encoded plot.

Reason: Offers quick insights without LLM overhead for common metrics, with visuals enhancing usability, while keeping /ask for custom queries.

Challenges

LLM Resource Demands:

Issue: Loading gpt-neo-2.7B (even with 4-bit quantization) requires significant memory (~2-3 GB GPU), causing potential crashes (e.g., Recv failure).

Mitigation: Used 4-bit quantization and suggested smaller models (e.g., distilgpt2) for testing. Could optimize further with model pruning or batch processing.

Vector Store Precision:

Issue: VectorStore summaries might lack detailed stats (e.g., cancellation by country), leading to "insufficient data" responses for some questions.

Mitigation: Current summaries include totals and averages; could enhance with per-country or per-hotel stats, though this increases storage and retrieval time.

Response Time:

Issue: LLM inference can take 2-5 seconds per query, risking timeouts or Recv failure in curl.

Mitigation: Set max_new_tokens=150 and temperature=0.3 for concise, fast responses. Could add async caching or precomputed embeddings for frequent queries.

JSON File Concurrency:

Issue: Appending to responses.json in a single file risks race conditions if multiple users query simultaneously.

Mitigation: Current script rewrites the file each time; could use a lock (e.g., threading.Lock).

Error Handling:

Issue: Network or parsing errors (e.g., curl JSON issue) can disrupt service.

Mitigation: Added robust logging and HTTP exceptions; could improve with retry logic or more detailed client feedback.

Future Improvements

Scalability: Move to a distributed setup (e.g., Docker with a load balancer) and use a database for responses.

Performance: Cache frequent queries or offload LLM inference to a dedicated server.

Accuracy: Enhance VectorStore with richer summaries (e.g., cancellation trends) for better LLM answers.