Retail Insights Assistant

From Prototype to Production (100GB+ Scale)

Current Implementation (The Prototype)

- Goal: Enable natural language querying on ad-hoc CSV datasets.
- Architecture:
  - User Interface: Streamlit (Chat & Visualization).
  - Orchestration: LangGraph (Stateful Multi-Agent Workflow).
  - Reasoning Engine: Groq (Llama 3) for fast inference and code generation.
  - Execution Environment: Local Python/Pandas Sandbox.
- Workflow:
  - Router: Analyzes user intent (Query vs. Chat).
  - Coder: Generates Pandas syntax based on dynamic schema injection.
  - Executor: Runs code to produce DataFrames or Matplotlib figures.
  - Synthesizer: Translates raw data into business insights.

Scaling to 100GB+ (The Challenge)

- Problem: 100GB CSVs cannot fit in memory (RAM); Pandas is insufficient.
- Solution: The "Lakehouse" Architecture.
- Storage Layer: Move data to Snowflake or Google BigQuery (Columnar storage is essential for retail aggregates).
- Compute Layer: Separate compute from storage. Use PySpark (Databricks) for ETL/Cleaning of raw logs.
- Retrieval Layer (RAG):
    - Structured: Shift LLM from "Text-to-Pandas" to "Text-to-SQL". The database handles the heavy lifting (Sum, Avg, Group By).
    - Unstructured: Use Vector Search (Pinecone) to find relevant Product Descriptions or Customer Reviews, reducing context window usage.

Handling Ambiguity & Accuracy

- Schema Injection: We only pass metadata (Column Names, Types, Sample Rows) to the LLM, not the full data.
- Validation Loop:
  - Step 1: Agent generates SQL/Code.
  - Step 2: "Dry Run" or Syntax Check.
  - Step 3: If empty result, the Validation Agent rewrites the query automatically (Self-Correction).
- Security: Read-only database credentials to prevent SQL injection or accidental deletion.

Performance & Cost Optimization

- Caching: Redis cache for frequent queries (e.g., "Total Sales Q4") prevents redundant LLM/DB calls.
- Latency:
  - Current (Pandas): ~2-5s processing time.
  - Future (Snowflake): ~1-3s for SQL execution on TB-scale data.
- Cost Control: Use smaller models (Llama-3-8b) for query routing and larger models (Llama-3-70b) only for complex code generation.