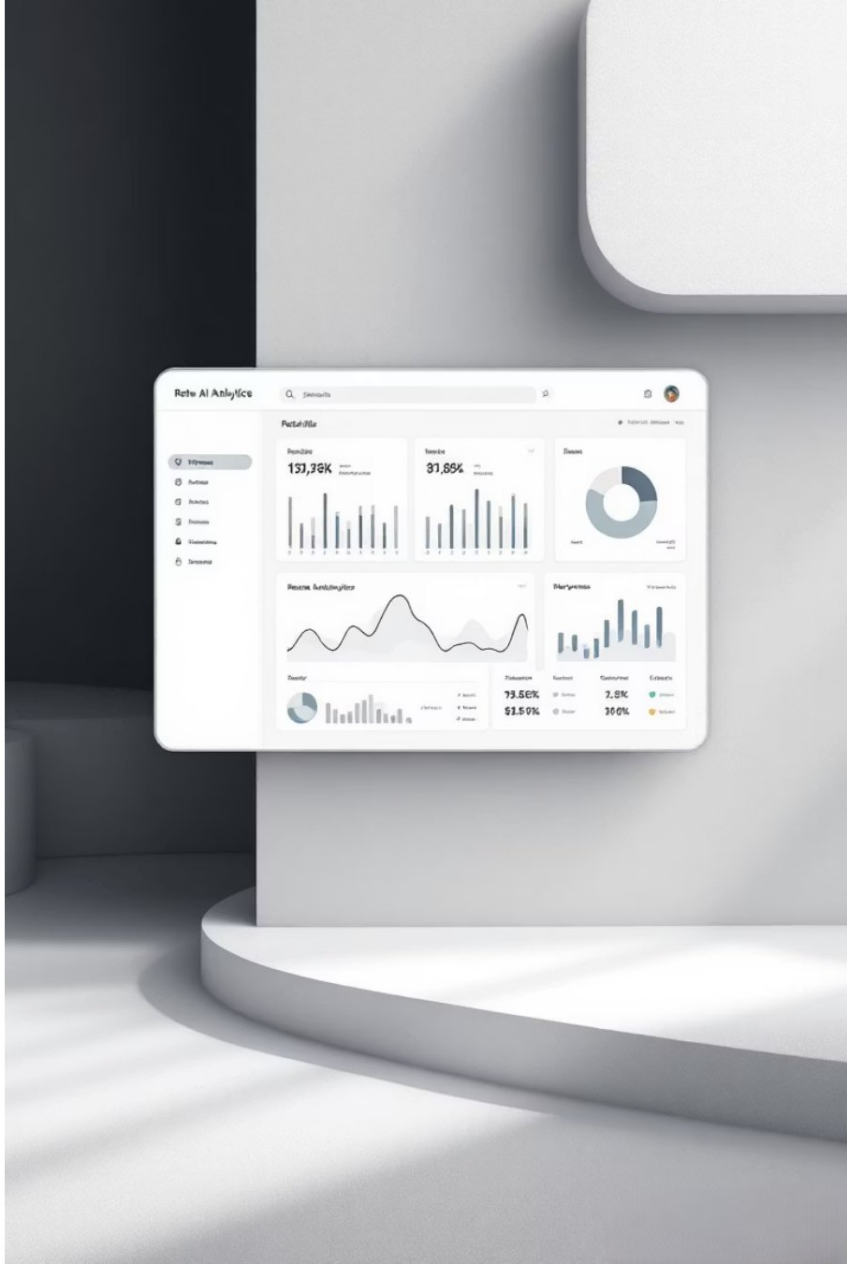
Retail Insights Assistant

Intelligent Retail Analytics • Natural-Language Q&A • 100GB+ Scale

Manoj Kumar Kamble



The Challenge: Fragmented Retail Data

The Problem

Retail data is scattered across multiple platforms—Amazon sales, international markets, inventory systems, pricing databases, and expense tracking. Executives need instant answers to complex questions like "What's our YoY growth?" or "Which SKU underperformed this quarter?" but accessing insights requires manual data compilation and analysis across disparate sources.

Our Objective

Build a GenAI-powered Retail Insights Assistant that automatically summarizes performance, answers natural-language queries, merges multi-source datasets, and scales to handle 100GB+ of historical retail data. This system transforms how stakeholders interact with their data—from days of analysis to seconds of conversation.

Comprehensive Feature Set

Multi-Format Ingestion

Supports CSV, Excel, and JSON formats across all retail data sources: Amazon Sales, International Sales, Inventory, Pricing, and Expenses. Flexible input handling ensures seamless integration regardless of source system.

Automatic Summarization

Generates detailed business summaries with metrics, charts, and tables. Provides insights across categories, SKUs, sizes, and colors without manual intervention. Executive-ready reports in seconds.

Natural Language Q&A

Conversational interface translates questions into query plans, executes SQL, and returns answers with full provenance. Includes confidence scoring and a dropdown menu of common retail questions to get started quickly.

System Architecture Overview

01

UI Layer

Streamlit-based interface for file upload, summary generation, and chat interactions

02

Multi-Agent Layer

Language-to-Query Agent, Data Extraction Agent, and Validation Agent working in concert

03

Data Layer

Multi-file ingestion with schema unification, DuckDB SQL execution, and Parquet-ready canonical data model

04

LLM Layer

Configurable OpenAI or Gemini integration with prompt templates and structured JSON output

05

Optional Vector Store

Chroma or FAISS for conversation memory and contextual retrieval

The complete flow: User → UI → LLM Query Planner → SQL → DuckDB → Results → LLM Validator → Final Answer

Canonical Data Model

Multi-Source Ingestion

We consolidate data from five critical sources:

- **Amazon Sales** – revenue, quantity, dates
- **International Sales** – revenue, quantity by region
- **Inventory** – stock levels, category, size, color
- **Product Master** – pricing (MRP, TP)
- **Expenses** – cost allocation by type

Standardization Process

Column normalization, date parsing, numeric cleanup (removing currency symbols and commas), and SKU standardization ensure data quality.

Unified Schema Overview

The consolidated data is transformed into a unified schema, designed for efficient querying and analysis. Here are the core tables, their columns, data types, and relationships:

1. `sales_transactions` Table

- **Purpose:** Stores all sales event data.
- **Columns:**
  - `date` (DATE): Date of the transaction.
  - `sku` (TEXT): Stock Keeping Unit, linking to `product_master` and `inventory`.
  - `qty` (INTEGER): Quantity of units sold.
  - `revenue` (DECIMAL): Total revenue generated from the transaction.
  - `source` (TEXT): Origin of the sale (e.g., 'Amazon', 'International').

2. `inventory` Table

- **Purpose:** Maintains current stock levels and product attributes.
- **Columns:**
  - `sku` (TEXT): Primary Key, also a Foreign Key to `product_master`.
  - `stock` (INTEGER): Current quantity in stock.
  - `category` (TEXT): Product category.
  - `size` (TEXT): Product size.
  - `color` (TEXT): Product color.

3. `product_master` Table

- **Purpose:** Contains master data for all products.
- **Columns:**
  - `sku` (TEXT): Primary Key for product identification.
  - `mrp` (DECIMAL): Maximum Retail Price.
  - `tp` (DECIMAL): Trade Price (cost to company).
  - `category` (TEXT): Product category, consistent with `inventory`.

4. `expenses` Table

- **Purpose:** Records operational expenses.
- **Columns:**
  - `date` (DATE): Date when the expense was incurred.
  - `amount` (DECIMAL): Monetary value of the expense.
  - `type` (TEXT): Classification of the expense (e.g., 'Marketing', 'Logistics').

Relationships & Implementation

The primary relationship between `sales_transactions`, `inventory`, and `product_master` is established through the `sku` column. This allows for seamless joins to enrich sales data with product details and inventory status.

Technical implementation leverages **DuckDB** for in-memory SQL execution, enabling rapid querying of large datasets. The canonical data model is stored in **Parquet** format, ensuring columnar storage efficiency and compatibility with various data processing tools. Data ingestion includes robust schema validation to enforce these data types and relationships.

LLM Integration Strategy

Intent Understanding

The LLM extracts key elements from natural language: metrics (revenue, quantity), dimensions (category, SKU), filters (date ranges, product attributes), and time windows. This structured understanding drives accurate query generation.

Query Planning

Using our schema map, the LLM generates SQL templates in structured JSON format. This ensures consistent, executable queries that align with our canonical data model while maintaining semantic accuracy.

Answer Validation

Results undergo automated validation checking for anomalies like negative values or missing fields. The system generates readable insights with confidence scores, ensuring stakeholders trust the answers they receive.

Optional Memory

Vector database integration enables long conversation threads and report retrieval. This contextual memory makes multi-turn conversations feel natural and maintains state across complex analytical workflows.

Our prompt engineering approach includes strict JSON schemas, guardrails against hallucination, and instruction tuning specifically for the retail analytics domain.

Multi-Agent Orchestration

Language-to-Query Agent

Transforms natural language questions into structured query plans. Identifies the appropriate dataset (sales, inventory, or expenses) and builds SQL templates aligned with our schema.

Data Extraction Agent

Refines SQL based on actual column names and executes queries against DuckDB. Produces result tables with previews and metadata about rows scanned and execution time.

Validation Agent

Inspects results and computes confidence scores. Generates key insights, formatted tables, confidence labels, and optional warnings about data quality or query ambiguity.

This three-agent architecture ensures accuracy, reliability, and transparency in every query response. Each agent has a specialized role, creating a robust pipeline from question to insight.

Query-Response Pipeline in Action

Example Query: "Which category had the highest revenue and stock availability last quarter?"

1 User Input

Question entered through Streamlit UI

2 Query Planning

Language Agent generates structured JSON plan

3 SQL Execution

Data Agent runs query against DuckDB

4 Validation

Validation Agent generates insights

5 Response

UI displays answer with full provenance

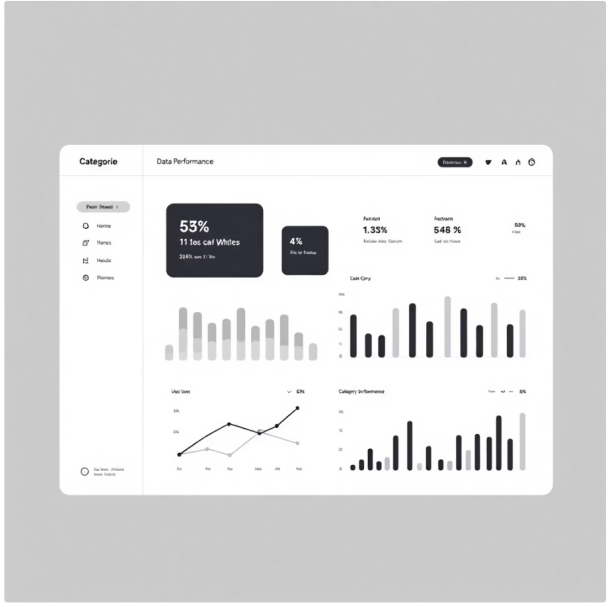
Sample Output

Insight: "Kurta leads with 114,339 stock units and top revenue of ₹2.4M for Q4."

Confidence: High (98%)

Rows Scanned: 45,231 transactions

The system includes the actual SQL query in the response, enabling technical users to verify logic and non-technical users to understand the data source. Full transparency builds trust in AI-generated insights.



Scaling to 100GB+ Data Volumes

Distributed Data Engineering

PySpark and Databricks handle heavy ETL workloads. Batch ingestion converts raw CSV files into partitioned Parquet format. Streaming ingestion via Kafka and Spark enables real-time updates. Periodic pre-aggregations (daily, monthly, quarterly) accelerate common queries.

Multi-Tier Storage Architecture

Data Lake (S3, ADLS, or GCS) organized into raw, clean, and curated zones. Cloud data warehouse (BigQuery or Snowflake) for enterprise analytics. Parquet with Delta Lake provides ACID transactions and time travel capabilities. Strategic indexing includes min/max statistics and vector embeddings.

Optimized Retrieval Patterns

DuckDB and Trino enable fast subset queries on local data. Precomputed aggregates serve common questions instantly. RAG (Retrieval-Augmented Generation) provides contextual filtering for complex analytical scenarios requiring historical context.

Performance, Cost & Monitoring

Cost Optimization

- Small LLM for query parsing, large LLM only for complex summarization
- Three-tier caching: query plans, SQL results, and LLM responses
- Pre-aggregated tables reduce compute costs by 60-80%
- Smart model routing based on query complexity

Performance Tuning

- DuckDB delivers sub-second queries on multi-GB datasets
- Pushdown filters and partition pruning minimize data scanned
- Vector DB invoked only when semantic search adds value
- Parallel query execution for batch operations

Monitoring & Quality

- Track SQL execution latency and LLM token usage
- Measure accuracy against ground-truth queries
- Alert on missing columns or malformed data
- Confidence-based fallback for low-certainty answers

<1s

Query Response Time

Average latency for standard queries

80%

Cost Reduction

Through caching and pre-aggregation

95%

Query Accuracy

Against validated test cases

100GB+

Data Scale

Supported with sub-second performance