Myntra Data Query Agent Documentation

Date: September 28, 2025
Author: Adityanarayana Rao Yerabati
Description: This document provides an overview of the Myntra Data Query Agent (MDQA), a custom mini agent for analyzing the Myntra e-commerce product dataset. It includes definitions of key concepts, code details, a flow diagram, and placeholders for screenshots of example queries.

Introduction to Key Concepts

This section provides short, concise explanations of core concepts used in the agent.

- **What is RAG (Retrieval-Augmented Generation)?**
  RAG is a technique that combines information retrieval with generative AI (like LLMs). It "retrieves" relevant data from a source (e.g., a database or documents) to augment the prompt given to a language model, ensuring generated responses are accurate, grounded in real data, and less prone to hallucinations.
- **What is an Agent?**
  An agent is an AI system that can autonomously perform tasks, make decisions, and interact with environments or users. It often uses tools (e.g., APIs, databases) in a loop: observe input, plan actions, execute, and refine based on results. Agents can be simple (rule-based) or complex (LLM-powered with reasoning).
- **What is a Custom Mini Agent?**
  A lightweight, purpose-built agent without heavy frameworks (e.g., no LangChain). It focuses on specific tasks like data analysis, using a simple loop for interaction. In this case, it's tailored for SQL-based queries on structured data.
- **What is DuckDB?**
  DuckDB is an in-memory analytical database engine optimized for fast querying of large datasets (e.g., CSVs) without loading everything into RAM. It's used here for efficient SQL operations on the ~1.4 GB Myntra dataset.
- **What is OpenAI API in This Context?**
  The OpenAI API (e.g., GPT-3.5-turbo) powers the LLM for translating natural language to SQL and generating insights. It requires an API key for authentication.

Overview of the Myntra Data Query Agent (MDQA)

The MDQA is a custom mini agent built in Google Colab for analyzing the Myntra e-commerce product dataset (from Kaggle: https://www.kaggle.com/datasets/promptcloud/myntra-e-commerce-product-data-november-2023). It uses RAG principles:

- **Retrieval**: Generates and executes SQL queries on DuckDB to fetch data subsets.
- **Augmentation & Generation**: Feeds retrieved data to an LLM for insightful natural language responses.

Key Features:

- Interactive loop for user queries (e.g., "What are the top 5 brands with the most products listed?").
- Handles large CSVs efficiently.
- Error handling for SQL issues (e.g., column mismatches).
- Based on dataset schema (e.g., columns like `Brand`, `Product_Id`, `Sale_Price`).

Assumptions:

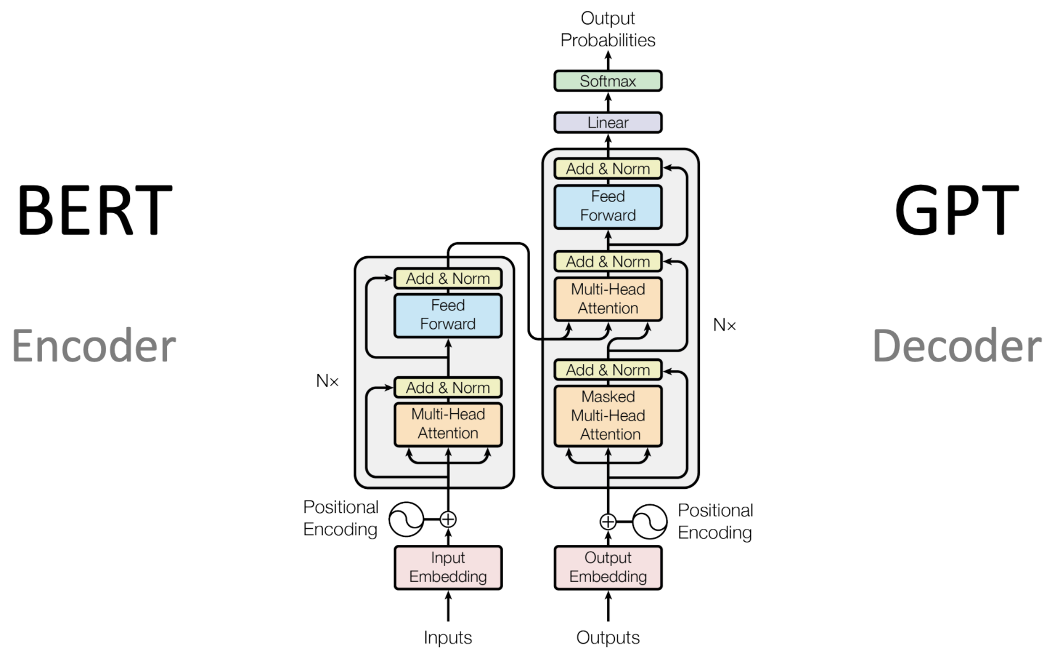
- Dataset uploaded as CSV.
- OpenAI API key provided.
- Runs in Google Colab with dependencies: duckdb, pandas, openai.

Basic Transformer and RAG Diagrams

This section provides basic diagrams for the Transformer architecture (the foundation for many LLMs used in RAG) and Retrieval-Augmented Generation (RAG) itself. These concepts are integral to how the agent uses LLMs for SQL generation and insights.

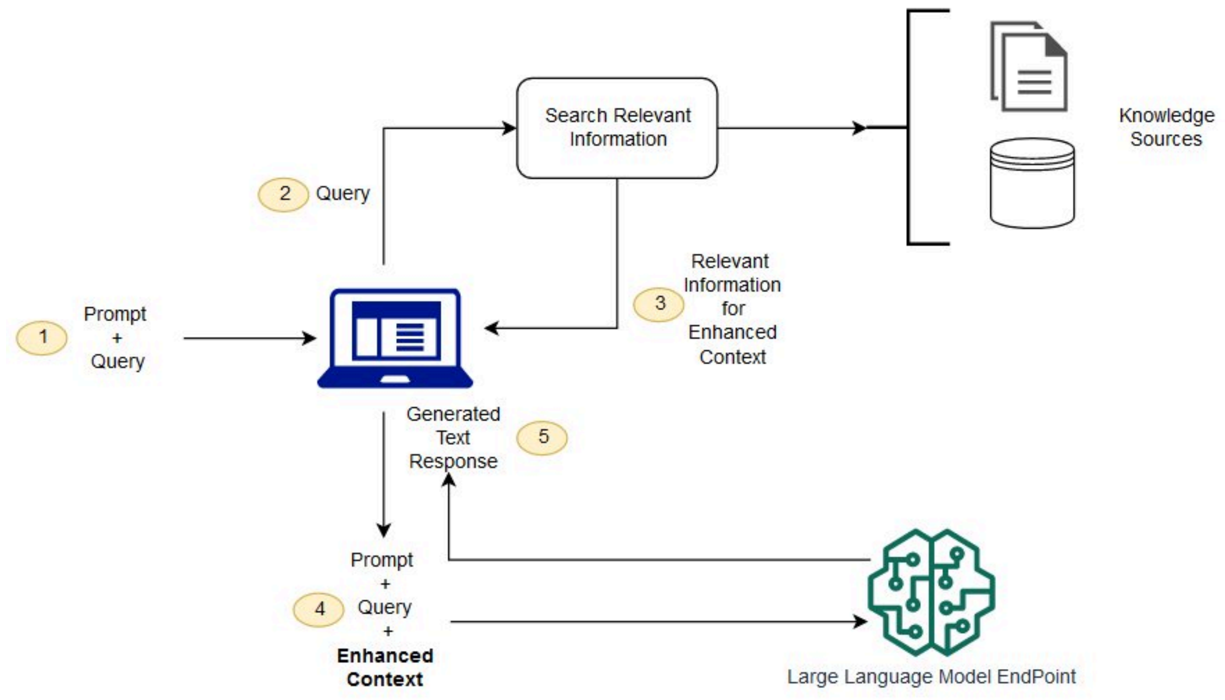
Basic Transformer Architecture

The Transformer is a neural network architecture introduced in the paper "Attention Is All You Need" (2017). It relies on self-attention mechanisms to process sequential data in parallel, making it efficient for tasks like translation and text generation. Key components include encoders (for input processing) and decoders (for output generation), with multi-head attention, feed-forward networks, and positional encodings.



Basic RAG (Retrieval-Augmented Generation) Diagram

RAG enhances LLMs by retrieving external data to augment prompts, improving accuracy and reducing hallucinations. It involves indexing data, retrieving relevant chunks via embeddings and vector search, and generating responses with the augmented context.


Search Relevant Information
Knowledge Sources
2 Query
1 Prompt + Query
3 Relevant Information for Enhanced Context
Generated Text Response 5
4 Prompt + Query + Enhanced Context
Large Language Model EndPoint

Description: The diagram illustrates the RAG pipeline: document ingestion, query retrieval, and response generation using an LLM.

Code Details

The agent is implemented across 5 Colab cells. Below is a breakdown of each cell, including purpose and key code snippets.

Cell 1: Install Dependencies

Installs required Python packages.

```
!pip install -q duckdb pandas openai
```

Cell 2: Upload Dataset CSV

Allows direct upload of the CSV file.

```
from google.colab import files
import os

print("Please upload the Myntra dataset CSV file (e.g.,
myntra_ecommerce_product_data_november_2023.csv):")
uploaded = files.upload()
CSV_FILE = list(uploaded.keys())[0]
print(f"Uploaded file: {CSV_FILE}")
```

Cell 3: Load Dataset into DuckDB and Get Schema

Loads the CSV into DuckDB, prints schema and sample data, and initializes OpenAI client.

```
import duckdb
import pandas as pd
from openai import OpenAI

con = duckdb.connect(':memory:')
con.execute(f"CREATE OR REPLACE TABLE products AS SELECT * FROM
read_csv_auto('{CSV_FILE}', header=true, ignore_errors=true)")
schema = con.execute("DESCRIBE products").fetchdf().to_string(index=False)
print("Table Schema:\n", schema)
sample_data = con.execute("SELECT * FROM products LIMIT
5").fetchdf().to_string(index=False)
print("\nSample Data:\n", sample_data)
client = OpenAI(api_key="MY_API_KEY")  # Replace with your key
```

Cell 4: Define Custom Agent Functions

Defines core functions for SQL generation, execution, and insights.

```
def generate_sql(query: str, schema: str, sample_data: str) -> str:
    # LLM prompt to generate SQL (details omitted for brevity)
    # Uses OpenAI to create SQL based on query, schema, and sample.

def execute_sql(sql: str) -> tuple[pd.DataFrame, str]:
    # Executes SQL on DuckDB, returns DataFrame or error.

def generate_insights(query: str, data: str, error: str) -> str:
    # Uses LLM to analyze data or explain errors.
```

Cell 5: Run the Custom Mini Agent

Interactive loop to process queries.

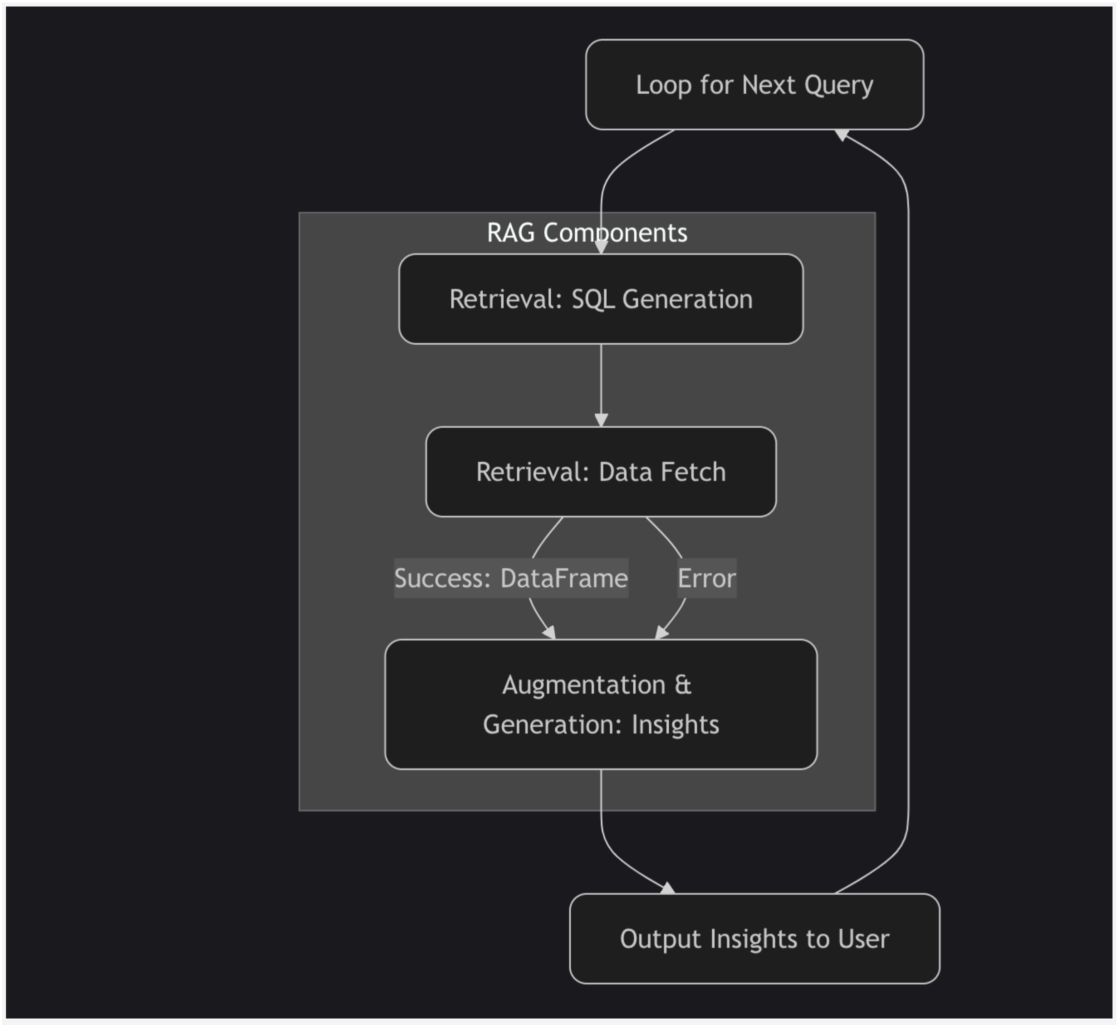
```
def run_mini_agent():
    print("Mini Agent ready! Ask questions about the Myntra dataset (e.g., 'Top 5 brands by
average price'). Type 'exit' to stop.")
    while True:
        user_query = input("\nYour query: ").strip()
        if user_query.lower() == 'exit':
            break
        # Generate SQL, execute, generate insights, and print.

run_mini_agent()
```

How It Works (High-Level):

- User inputs a natural language query.
- LLM generates SQL using schema/sample.
- DuckDB executes SQL to retrieve data.
- LLM generates insights from retrieved data.

Flow Diagram:

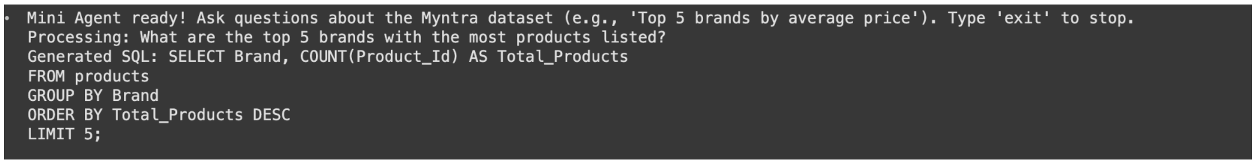


Description of Flow:

- **Start**: User query (e.g., "Top 5 brands by product count").
- **Retrieval Phase**: SQL generated and executed.
- **Generation Phase**: Insights produced.
- **Loop**: Continues until 'exit'.

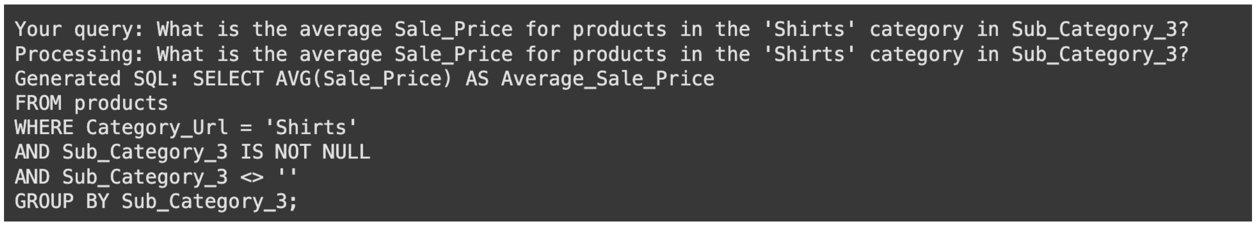
Snapshots:

1)



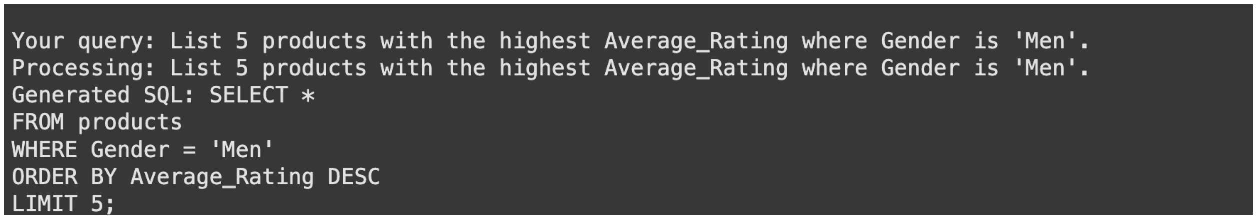
```
• Mini Agent ready! Ask questions about the Myntra dataset (e.g., 'Top 5 brands by average price'). Type 'exit' to stop.
  Processing: What are the top 5 brands with the most products listed?
  Generated SQL: SELECT Brand, COUNT(Product_Id) AS Total_Products
  FROM products
  GROUP BY Brand
  ORDER BY Total_Products DESC
  LIMIT 5;
```

2)



```
Your query: What is the average Sale_Price for products in the 'Shirts' category in Sub_Category_3?
Processing: What is the average Sale_Price for products in the 'Shirts' category in Sub_Category_3?
Generated SQL: SELECT AVG(Sale_Price) AS Average_Sale_Price
FROM products
WHERE Category_Url = 'Shirts'
AND Sub_Category_3 IS NOT NULL
AND Sub_Category_3 <> ''
GROUP BY Sub_Category_3;
```

3)



```
Your query: List 5 products with the highest Average_Rating where Gender is 'Men'.
Processing: List 5 products with the highest Average_Rating where Gender is 'Men'.
Generated SQL: SELECT *
FROM products
WHERE Gender = 'Men'
ORDER BY Average_Rating DESC
LIMIT 5;
```