Supermarket Receipt AI Agent - Solution Report

2026/01/24

Problem Statement

The task is to build an AI agent that can:

1. **Input:** Take multiple images of supermarket bills/receipts and a user query (text)

2. **Process:** Answer two specific types of queries:

   - **Query 1:** "How much money did I spend in total for these bills?"

   - **Query 2:** "How much would I have had to pay without the discount?"

3. **Reject:** Have the capacity to reject irrelevant queries that are not related to the above two

Solution Format and Structure

Notebook Framework: The solution is implemented within the course-provided notebook (https://colab.research.google.com/drive/1EMg1-uQwEi8Slnyc3XHz-2UvH7m_KUAo?usp=sharing), which contains essential starter codes.

Agent Implementation Location: The core agent code is written in **Section 2: AI Agent** of the notebook. This section contains all the agentic AI functions:

- Input validation

- Query routing

- Parallel image processing

- Answer calculation

- Answer formatting

API Key Configuration: The course-provided notebook template uses `VERTEX_API_KEY` for API key configuration. However, my implementation uses `GEMINI_API_KEY` instead.

Solution Overview

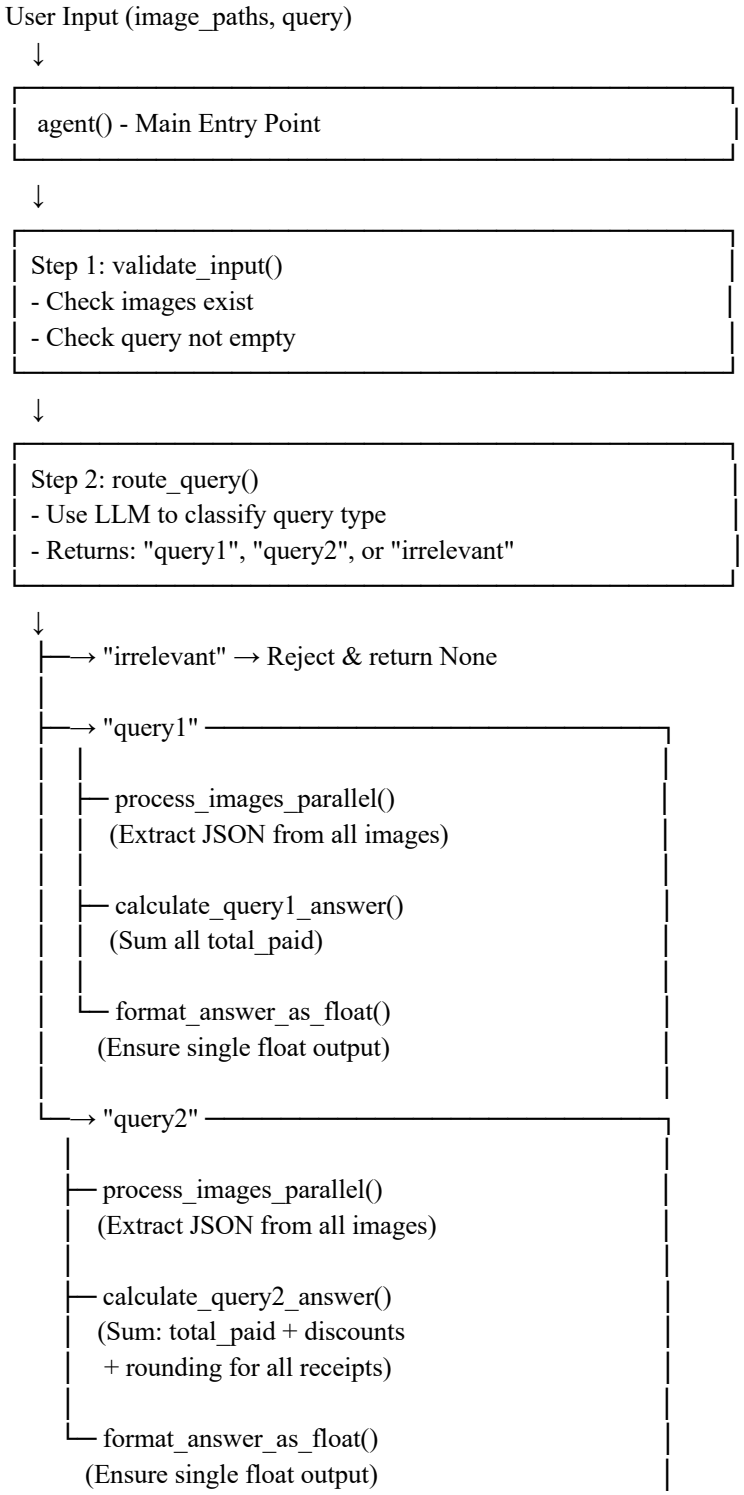
I designed an **agentic AI system** using LangChain and Google's Gemini 2.5 Flash model. The solution follows a modular architecture with three main stages:

1. **Input Validation:** Verify that images exist and query is not empty

2. **Query Routing:** Use LLM to classify the query type (Query 1, Query 2, or irrelevant)

3. **Parallel Processing:** Extract receipt data from all images simultaneously, then calculate the answer

Architecture



Core Components

1. Input Validation (validate_input)

- Ensure the agent receives valid inputs before processing.
- At least one image is provided
- All image file paths exist
- Query is not empty
- Returns: (is_valid: bool, error_message: str)

2. Query Router (route_query)

- Classify user queries into one of three categories using LLM.
- Returns: query1, query2, or irrelevant

3. Parallel Image Processing (process_images_parallel)

- Extract structured JSON data from multiple receipt images simultaneously.
- Image Encoding: Convert each image to Base64 Data URL format
- Parallel Extraction: Use RunnableParallel to call LLM for all images at once
- JSON Parsing: Parse LLM responses into structured data
- Data Validation: Verify required fields (total_paid, items) exist

4. Query 1 Calculator (calculate_query1_answer)

- Calculate the total amount spent across all receipts.

5. Query 2 Calculator (calculate_query2_answer)

- Calculate the total amount that would have been paid without any discounts.

6. Answer Formatter (format_answer_as_float)

- Ensure the final answer is returned as a single float value.
- Uses LLM to format the calculated answer as a pure number

Code Organization

Each function is well-documented with docstrings explaining purpose, parameters, and return values.

Solution Validation

In my local testing, the solution successfully: Handles multiple receipt images / Answers Query 1 (total spent) / Answers Query 2 (total without discounts) / Rejects irrelevant queries / Provides clear error messages / Processes images in parallel for efficiency / Handles edge cases gracefully.

Technical Stack

- Language: Python 3
- LLM Framework: LangChain
- Model: Google Gemini 2.5 Flash
- Key Libraries:
  - langchain_google_genai: LLM integration
  - langchain_core: Chain composition and parallel processing
  - base64 / mimetypes: Image encoding
  - json: Data parsing