Name: Wang Xiaofan Student ID: 1155247021

Agentic AI for Business and FinTech (SEEM5660)

Individual Homework 01, Due date: 30 January, midnight.

Introduction

The objective of this assignment was to develop a multimodal AI agent capable of processing multiple images of supermarket receipts. The agent is required to accurately answer two specific types of user queries: calculating the total money spent and calculating the original price without discounts. Additionally, the system must possess the capability to identify and reject irrelevant queries.

This report documents the solution architecture, implementation strategies, and the logic used to ensure calculation accuracy and robust query routing.

1 System Architecture

The solution follows a Map-Reduce pattern combined with a Logic-Based Router. This architecture was chosen to handle the multi-image input efficiently while mitigating the common "hallucination" issues associated with Large Language Models (LLMs) performing arithmetic operations.

1.Input: The user uploads multiple image receipts. To accommodate the Multimodal LLM input requirements, the system uses the mimetypes library to automatically identify file types and converts them into Base64-encoded Data URLs.

2.Map Phase: The system iterates through each image individually. It uses the gemini-2.5-flash model to extract specific financial data points (Total Spend and Discount) into a structured JSON format.

3.Reduce Phase: Python native code aggregates the extracted data. It calculates the Grand Total and Grand Discount using standard programming arithmetic rather than relying on the LLM to sum numbers.

4.Routing Phase: The system analyzes the user's text query using keyword matching to determine the intent.

5.Output: Based on the routed intent, the system constructs the final response using the pre-calculated values or issues a refusal message.

2 Implementation Details

2.1 Tool Stack

LLM: Google Gemini 2.5 Flash. This model was selected for its cost-efficiency and strong multimodal capabilities.

Orchestration: LangChain Core (for prompt templating and output parsing).

Environment: Google Colab.

2.2 Parallel Information Extraction (Map Phase)

This is the core processing component, corresponding to the map_extract_data function.
The prompt explicitly instructs the model to "Look for any numeric value with a NEGATIVE SIGN" and items labeled "Savings" or "Adjustment". The model is forced to return a JSON object. This ensures the Python script can reliably parse the numbers later.

```
prompt_text = (
    "Analyze this receipt image to extract financial data.\n"
    "1. Identify the 'Total Paid' (Final Charge).\n"
    "2. Identify 'Discount': Look for any numeric value with a NEGATIVE SIGN (e.g., -1.00, -5.20).
    "Sum the ABSOLUTE VALUES of these negative numbers. "
    "Also include items labeled 'Savings' or 'Adjustment' if they reduce the total.\n"
    "If no discount/negative numbers found, set discount to 0.\n"
    "\n"
    "Return ONLY JSON format: {\"total\": float, \"discount\": float}"
)
```

2.3 Data Aggregation & Calculation (Reduce Phase)

A critical design decision was to not ask the LLM to sum the totals across images. LLMs often make arithmetic errors with floating-point numbers. Instead, the extracted values are summed using Python:

```
grand_total = sum(item['total'] for item in data_list)
grand_discount = sum(item['discount'] for item in data_list)
grand_original = grand_total + grand_discount
```

This ensures mathematical precision. The original price is derived logically:

$$\text{Original Price} = \text{Total Paid} + \text{Total Discount}$$

2.4 Intent Routing & Response Generation

The router uses Python conditional logic (if-elif-else) to classify user intent based on keywords.
1.Spending Inquiries: Triggered by keywords like "total", "spend", "cost", "pay". Returns grand_total.
2.Original Price/Discount Inquiries: Triggered by keywords like "without discount", "original", "gross". Returns grand_total + grand_discount.
3.Irrelevant Queries: If no keywords match, it returns a default "I'm sorry, I can't answer this question" response.

```
if "total" in query and ("spend" in query or "pay" in query or "cost" in query or "money" in query)
    return f"Your total spending is {grand_total:.2f}"
elif ("without" in query and "discount" in query) or "original" in query or "gross" in query:
    return (f"You should pay {grand_original:.2f} without discount")
else:
    return "I am sorry, I can't answer this question."
```

2.5 Output Layer

This layer is responsible for converting calculation results into user-friendly feedback and managing the interaction lifecycle.

The system generates the final natural language response based on the variables and templates determined during the routing phase.
The system remains active after generating output, allowing users to pose new questions regarding the same batch of uploaded images, or enter the command new to restart and upload a new batch of images.

3 System Demonstration & Results

The system was tested with a batch of real-world data to validate its extraction capabilities and logic routing. The execution yielded the following results:

3.1 Batch Ingestion

The system successfully uploaded and processed a batch of 7 receipt images (receipt1.jpg through receipt7.jpg). The file type detection logic correctly handled the .jpg format for all inputs.

3.2 Financial Analysis & Aggregation

The system demonstrated accurate financial calculations based on the user's natural language queries:
Total Spending: In response to the query "How much money did I spend in total for these bills?", the system aggregated the extracted totals from all 7 receipts and returned a calculated sum of 1974.30.
Gross Value (Pre-Discount): When the user asked "How much would I have had to pay without the discount?", the system correctly summed the total paid and the discount amounts to derive a gross original value of 2348.20.

3.3 Boundary Handling

The system's routing logic successfully filtered out-of-scope queries. When presented with an irrelevant question ("What's the weather today?"), the system correctly triggered the fallback mechanism and returned the default response: "I am sorry, I can't answer this question."