HW1 Report: Multimodal Bill Processing Agent

Student Name: ZHANG Jiancong

Student ID: 1155241158

1. Introduction

The objective of this assignment is to develop a Multimodal AI Agent capable of processing multiple supermarket receipt images and responding to specific financial queries. As outlined in the requirements, the system must handle three distinct scenarios:

1. **Query 1:** Calculate the total money spent.
2. **Query 2:** Calculate the original price without discounts.
3. **Irrelevant Queries:** Reject questions unrelated to the bills.

My solution implements a **Router-based Parallel Architecture** using LangChain and Google Gemini. The design prioritizes **deterministic calculation** over probabilistic generation to ensure financial accuracy.

2. System Architecture

To meet the requirement of processing "several images" efficiently, I moved away from a sequential loop approach and adopted a **Parallel Execution Model**. The system consists of three logical layers:

2.1 Layer 1: The Semantic Router (Intent Classification)

Before processing images, the system analyzes the user text query.

- **Function:** Classifies intent into TOTAL, ORIGINAL, or IRRELEVANT.
- **Benefit:** Acts as a defensive layer. If the query is IRRELEVANT, the system halts immediately, saving computational costs and latency.

2.2 Layer 2: Parallel Information Extraction

If the query is financial, the system triggers a **Parallel Chain** (RunnableParallel).

- **Process:** All receipt images are processed concurrently (simultaneously) by the LLM.
- **Output:** Instead of text, the LLM extracts structured data in JSON format: {"total_paid": float, "savings": float}.

2.3 Layer 3: Deterministic Logic Aggregator

The final answer is derived using Python logic based on the Router's decision, rather than relying on the LLM's arithmetic capabilities.

- **For Query 1:** Sum(total_paid)
- **For Query 2:** Sum(total_paid) + Sum(savings)

3. Prompt Engineering & Refinement Process

To achieve high accuracy, I utilized an **iterative debugging process** for my prompts. Below documents how the prompts evolved from naive baselines to the final robust versions.

3.1 Handling Query 2 (Original Price)

The requirement to calculate "price without discount" posed a significant challenge regarding hallucination.

Iteration	Approach & Prompt Strategy	Outcome / Analysis
v1 (Baseline)	**End-to-End Reasoning:**"Look at the images and tell me how much I would pay without discount."	**Failure:** The model often attempted to do math internally and failed. It frequently missed "hidden" savings that were listed at the bottom of the receipts.
v2 (Refined)	**Structured Extraction + External Math:**"Extract the total_paid and savings amounts. Output strictly in JSON. Do not calculate."	**Success:** By shifting the burden from *calculation* to *extraction*, the model's accuracy improved significantly. The "original price" is then calculated deterministically in Python (Total + Savings), ensuring 100% arithmetic correctness.

3.2 Handling Irrelevant Queries (Rejection)

The model required a mechanism to reject out-of-domain queries (e.g., weather, food recipes).

- **Initial Approach:** I used a generic system prompt: *"You are a financial assistant."*
    - *Issue:* The model tried to be too helpful. When asked about "apples" (irrelevant context), it tried to find apples in the bill items, which was not the intended behavior for a general conversation.
- **Final Approach:** I implemented a **Strict Classification Router**.
    - *Prompt: "Classify the User Query into exactly one of the following three categories: TOTAL, ORIGINAL, IRRELEVANT. Do not output anything else."*
    - *Result:* This enforced a hard boundary. Any query not mapping to the financial logic is immediately blocked with a standard rejection message.

4. Implementation Details

The solution is implemented in Python within Google Colab, leveraging the **LangChain** framework for orchestration.

- **Model:** gemini-1.5-pro (via ChatGoogleGenerativeAI) is used for both text routing and image analysis.
- **Parallelism:** RunnableParallel is used to map the extraction chain across the list of image inputs. This ensures that the processing time remains manageable even as the number of receipts increases.
- **Data Structure:** JsonOutputParser is used to force the LLM to return valid dictionaries, preventing parsing errors during the aggregation phase.

5. Experimental Results

The system was tested against the provided sample images and specific test queries.

Scenario 1: Query 1 (Total Spend)

- **Input:** "How much money did I spend in total?"
- **Mechanism:** Router detected TOTAL. Aggregator summed total_paid fields.
- **Output:** The total money spent for these bills is 851.60

```
[Agent] 收到查询: 'How much money did I spend in total?'
[Agent] 识别意图: TOTAL
[Agent] 正在并行处理 3 张图片...
[Agent] 提取数据成功: [{'total_paid': 394.7, 'savings': 85.48}, {'total_paid': 316.1, 'savings': 76.09}, {'total_paid': 140.8, 'savings': 19.22}]
Final Answer: The total money spent for these bills is 851.60.
```

Scenario 2: Query 2 (Without Discount)

- **Input**: "How much would I have had to pay without the discount?"
- **Mechanism:** Router detected ORIGINAL. Aggregator calculated total_paid + savings.
- **Output:** Without the discount, you would have had to pay 1032.39.

```
[Agent] 收到查询: 'How much would I have had to pay without the discount?'
[Agent] 识别意图: ORIGINAL
[Agent] 正在并行处理 3 张图片...
[Agent] 提取数据成功: [{'total_paid': 394.7, 'savings': 85.48}, {'total_paid': 316.1, 'savings': 76.09}, {'total_paid': 140.8, 'savings': 19.22}]
Final Answer: Without the discount, you would have had to pay 1032.39.
```

Scenario 3: Irrelevant Query

- **Input**: "What is the weather like today?"
- **Mechanism:** Router detected IRRELEVANT. Execution halted.
- **Output:** I am a financial assistant for supermarket bills. I cannot answer queries unrelated to the bills

```
[Agent] 收到查询: 'What is the weather like today?'
[Agent] 识别意图: IRRELEVANT
Final Answer: I am a financial assistant for supermarket bills. I cannot answer queries unrelated to the bills.
```

6. Conclusion

By separating **perception** (extracting numbers from images) from **reasoning** (routing intents) and **calculation** (Python math), the designed agent achieves high reliability and scalability. The **Prompt Refinement** process played a crucial role in eliminating arithmetic hallucinations and ensuring strict adherence to the query rejection requirements.