Individual Homework 01 Report

Student Name: Liao Ruiqi   Student ID: 1155245575

1. Problem Description

The objective of this assignment is to develop an AI agent capable of analyzing multiple supermarket receipts (images) and answering specific financial queries. The system must handle multimodal input (images + text) and address the following tasks:

- Query 1 (Total Spend): Calculate the total actual payment across all bills.
- Query 2 (Original Price): Calculate the total cost if no discounts were applied.

Guardrails: Detect and reject irrelevant queries (e.g., questions about weather).

2. Solution Architecture -- Pipeline

Instead of feeding all images into the LLM simultaneously (which risks context overflow and calculation hallucinations), I adopted a pipeline. This decomposes the complex task into manageable sub-tasks.

2.1 Individual Extraction

A specialized function **extract_amount_from_single_image** processes each receipt independently.

- Input: One receipt image + User Query.
- Model: Vertex AI
- Task: The LLM extracts a single, precise float value based on the specific query logic.

```python
# 2. Single image extraction
def extract_amount_from_single_image(image_path, query_text):
    """
    process only one, return float
    """
    try:

        data_url = get_image_data_url(image_path)

        # construct only one picture
        message_content = [
            {"type": "text", "text": query_text},
            {"type": "image_url", "image_url": {"url": data_url}}
        ]

        messages = [
            SystemMessage(content=single_image_system_instruction),
            HumanMessage(content=message_content)
        ]

        # use model
        response = llm.invoke(messages)
        content = response.content.strip()

        # clean
        clean_val = content.replace('$', '').replace('HKD', '').replace(',', '').strip()

        print(f"Processing {os.path.basename(image_path)} -> Raw: {content} -> Parsed: {clean_val}")

        return float(clean_val)

    except Exception as e:
        print(f"Error processing {image_path}: {e}")
        return 0.0
```

2.2 Aggregation

A Python-based pipeline iterates through the extracted results:

- Logic: Total_Sum = sum(individual_results)
- Reasoning: LLMs are prone to arithmetic errors when summing multiple decimal numbers.

  So Offload the summation.

```python
# Aggregate all results
def process_bill_query_pipeline(query_text):
    """
    Iterate through all images in the folder, compute each individually, then sum them up.
    """
    # Get all
    image_paths = sorted(glob.glob("*.jpg"))
    print(f"Starting pipeline analysis for {len(image_paths)} images...")

    total_sum = 0.0
    detailed_logs = []

    for path in image_paths:
        amount = extract_amount_from_single_image(path, query_text)

        total_sum += amount
        detailed_logs.append(f"{os.path.basename(path)}: {amount}")


    print("Individual Results:", detailed_logs)
    return total_sum
```

3. Prompt Engineering Strategy

The core challenge was accurately interpreting complex Hong Kong supermarket receipts, which contain mixed English/Chinese text and complex discount structures.

3.1 Strategy for Query 1: Total Spend (Actual Payment)

- Distinguishing between "Subtotal" and the final payment after "Rounding" (路整).

Solution: The system prompt explicitly instructs the model to anchor on the bottom-most payment keywords: "Octopus", "Amount Deducted", "Total Pay", or "VISA".

Key Instruction: "If 'Rounding' exists, look for the value *after* or *below* it."

3.2 Strategy for Query 2: Original Price (Without Discount)

Solution: implement a reconstruction formula based on the "Subtotal" (小計)anchor, which is statistically the most stable figure on these receipts.

Formula: Original Price = SUBTOTAL+｜Sum of Negative Discount Values｜

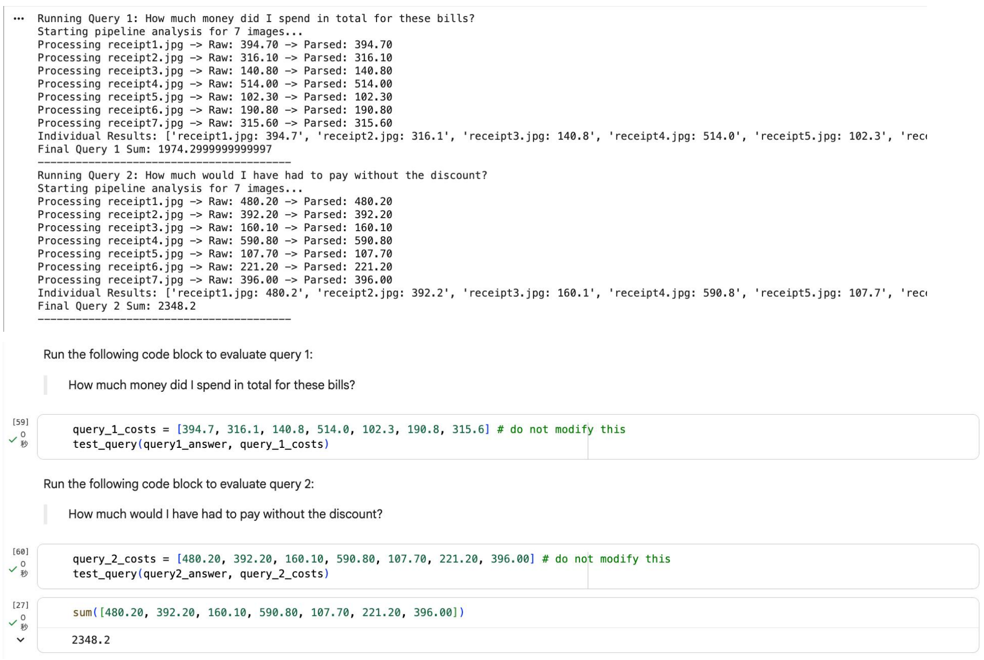
The prompt forces the model to ignore "Rounding" and strictly sum the absolute values of explicitly printed discounts (e.g., -$4.00, Buy 2 Save $3.9) and add them to the Subtotal.

3.3 Guardrails (Irrelevant Queries)

A specific instruction block was added to the System Prompt. If the user query is unrelated to financial analysis (e.g., "How is the weather?"), the model is hard-coded to return the string "Irrelevant query", which the Python pipeline intercepts.

4. Test Results

The model was evaluated on a dataset of 7 receipt images and get correct results.



```
Running Query 1: How much money did I spend in total for these bills?
Starting pipeline analysis for 7 images...
Processing receipt1.jpg -> Raw: 394.70 -> Parsed: 394.70
Processing receipt2.jpg -> Raw: 316.10 -> Parsed: 316.10
Processing receipt3.jpg -> Raw: 140.80 -> Parsed: 140.80
Processing receipt4.jpg -> Raw: 514.00 -> Parsed: 514.00
Processing receipt5.jpg -> Raw: 102.30 -> Parsed: 102.30
Processing receipt6.jpg -> Raw: 190.80 -> Parsed: 190.80
Processing receipt7.jpg -> Raw: 315.60 -> Parsed: 315.60
Individual Results: ['receipt1.jpg: 394.7', 'receipt2.jpg: 316.1', 'receipt3.jpg: 140.8', 'receipt4.jpg: 514.0', 'receipt5.jpg: 102.3', 'rec
Final Query 1 Sum: 1974.2999999999997
----------------------------------------
Running Query 2: How much would I have had to pay without the discount?
Starting pipeline analysis for 7 images...
Processing receipt1.jpg -> Raw: 480.20 -> Parsed: 480.20
Processing receipt2.jpg -> Raw: 392.20 -> Parsed: 392.20
Processing receipt3.jpg -> Raw: 160.10 -> Parsed: 160.10
Processing receipt4.jpg -> Raw: 590.80 -> Parsed: 590.80
Processing receipt5.jpg -> Raw: 107.70 -> Parsed: 107.70
Processing receipt6.jpg -> Raw: 221.20 -> Parsed: 221.20
Processing receipt7.jpg -> Raw: 396.00 -> Parsed: 396.00
Individual Results: ['receipt1.jpg: 480.2', 'receipt2.jpg: 392.2', 'receipt3.jpg: 160.1', 'receipt4.jpg: 590.8', 'receipt5.jpg: 107.7', 'rec
Final Query 2 Sum: 2348.2
----------------------------------------
```

Run the following code block to evaluate query 1:

> How much money did I spend in total for these bills?

[59]
```
query_1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this
test_query(query1_answer, query_1_costs)
```

Run the following code block to evaluate query 2:

> How much would I have had to pay without the discount?

[60]
```
query_2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this
test_query(query2_answer, query_2_costs)
```

[27]
```
sum([480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00])
```
```
2348.2
```