# Receipt QA Pipeline Report

Name: PENG ZIYYU  Student ID:1155246323

## 1. Problem Statement

This homework builds a robust pipeline that can take any number of receipt images and a user question, then return a correct answer for two target queries:

- **Query 1 (example):** How much money did I spend in total for these bills? (sum of final paid amounts)
- **Query 2 (example):** How much would I have had to pay without the discount? (sum of original amounts before discounts)

In the evaluation stage, the grader may provide different receipt images and may ask unrelated questions. The system must answer Q1/Q2 reliably and respond with "Sorry,your question is not relevant to my work." for unrelated questions.

## 2. High-level Design (Chains)

Following the course concepts (prompting and chaining), the solution is designed as two LLM-based chains plus deterministic calculators:

Pipeline:

```
[Receipt Images]
   (Chain A: Multimodal Information Extraction -> strict JSON)
   (Deterministic Functions: compute Q1 / compute Q2)
   (Chain B: Question Classification: Q1 / Q2 / OTHER)
```

Key idea:

Use an LLM only for perception + language tasks: reading receipts and interpreting the user question.

Use deterministic code for math: once structured numbers are extracted, calculations are exact and reproducible.

## 3. Chain A - Multimodal Receipt Extraction

Chain A converts N receipt images into one strict JSON object. To make the output machine-parsable, the system prompt forces: JSON only, no markdown, no extra commentary.

### 3.1 JSON Schema

| Field | Type | Meaning / Notes |
|---|---|---|
| receipts | array | One element per image, |

| | | |
|---|---|---|
| | | in the same order. |
| receipt_index | int | 1-based index matching image order. |
| paid_amount | number | Final paid/deducted amount (after discounts and rounding). |
| paid_method | string | Payment method (e.g., OCTOPUS, VISA, CASH, UNKNOWN). |
| subtotal_amount | number\|null | Subtotal shown on receipt if visible, else null. |
| rounding_amount | number | Rounding line if visible (often a small negative), else 0. |
| discounts | array | All visible discount lines; each contains label and amount (negative). |

### 3.2 Prompt Design

The extraction prompt explicitly defines the schema and rules. Important safeguards:

- Paid amount must be the final payment line (e.g., OCTOPUS/VISA amount).
- Include all discount lines (Buy X Save, % OFF, coupons, app promotions).
- Never invent missing fields; use null / UNKNOWN when not visible.
- Return strict JSON that can be parsed by json.loads.

### 3.3 Implementation Notes

Images are converted to data URLs (base64) and passed as multimodal inputs. The system retries on transient errors (e.g., rate limits) with exponential backoff.

```
def get_image_data_url(path):
b64 = base64.b64encode(open(path,"rb").read()).decode()
return f"data:image/jpeg;base64,{b64}"
```

## 4. Chain B - Question Classification

To handle paraphrases (e.g., “how much did I pay in total” == Q1), Chain B uses an LLM classifier that outputs exactly one label: Q1, Q2, or OTHER.

Classification rules:

- Q1: asks total amount actually paid/spent across receipts.
- Q2: asks total amount before discounts / without discounts.
- OTHER: anything else (e.g., weather).

## 5. Deterministic Computation (No LLM)

After extraction, Q1 and Q2 are computed deterministically to avoid arithmetic mistakes.

### 5.1 Q1 Formula

Sum of final paid amounts across all receipts:

**Q1 = sum(receipt.paid_amount for receipt in receipts)**

### 5.2 Q2 Formula

To reconstruct the original cost without discounts, add back the absolute value of each discount line:

**original_per_receipt = paid_amount + sum(abs(d.amount) for d in discounts)**
**Q2 = sum(original_per_receipt for each receipt)**

Note: rounding is not treated as a "discount" here; it is already reflected in the final paid amount.

## 6. Robustness and Error Handling

Several practical issues were addressed during development:

- PromptTemplate curly-brace collisions: avoid embedding raw JSON skeleton with { } inside ChatPromptTemplate variables; otherwise LangChain treats them as missing variables.
- JSON parsing stability: extract the first JSON object from the model output (support fenced ```json blocks) before json.loads.
- Rate-limit handling: retry and backoff; if exhausted, surface a clear error message in debug mode.
- Unrelated questions: classifier returns OTHER and the system replies "I don't know."

## 7. Testing and Results

Testing was done in Google Colab with an interactive "upload images + ask questions" loop. The same pipeline supports any number of images.

### 7.1 Example (7 receipts used for development)

On the provided 7 receipt images (development set), the system produced values close to the expected totals.

- Q1 total paid (HKD): 1974.40
- Q2 without discounts (HKD): 2237.47

Small variations can occur if the extraction model misreads a line item, but deterministic calculation ensures consistency once JSON is correct.

### 7.2 Interactive Test Interface

The submitted notebook includes a Colab upload step and a question loop. This matches the evaluation scenario where the grader may provide arbitrary receipt images and ask arbitrary questions.

```
uploaded = files.upload()
image_paths = list(uploaded.keys())

while True:
q = input("Question (or 'exit'): ").strip()
if q.lower() == "exit": break
print(answer_from_images_and_question(llm, image_paths, q)
```

## 8. Conclusion

This solution applies the course idea of chaining: a multimodal extraction chain to convert unstructured receipts into structured JSON, a classification chain to route user questions, and deterministic functions to compute accurate financial totals. The design is modular, testable, and robust to irrelevant questions.