SEEM5660 Individual Homework 01 Report

Course: Agentic AI for Business and FinTech (SEEM5660)

Name: Shiliang Chen

Student ID: 1155245855

Date: 2026-01-30

---

1. Task Description

Supported query types:

- **Query 1**: *How much money did I spend in total for these bills?* (total amount actually paid across all bills)
- **Query 2**: *How much would I have had to pay without the discount?* (total amount before discounts/promotions across all bills)
- **Out-of-domain**: any other questions must be rejected.

At test time, new receipt images will be used and 10 random queries (Query1/Query2/out-of-domain) will be asked. The final score is the number of correct answers.

---

2. Method Overview (System Design)

The solution consists of three modules:

- **(A) Single-receipt extraction (Multimodal LLM)**
  For each receipt image, the system calls Gemini via `langchain_google_genai` to extract key monetary fields in a strict JSON format:

- `total_paid_hkd`: final amount actually paid on that receipt (prefer payment lines such as OCTOPUS/VISA/AMOUNT DEDUCTED, etc.)

- `total_discount_hkd`: total discount/promotions on that receipt (sum discount lines and return as a positive number)

- `total_without_discount_hkd`: amount without discounts (prefer a pre-discount subtotal if shown; otherwise estimate as `paid + discount`)

- **(B) Query routing (LLM classifier)**
  A dedicated router prompt is used to classify an arbitrary user query into one of:

- `q1_total_spent` / `q2_without_discount` / `reject`
  If the router output cannot be parsed or is outside the allowed labels, the system defaults to `reject` to avoid incorrect answers.

- **(C) Aggregation across multiple receipts**

- For Query1: sum `total_paid_hkd` across all receipts

- For Query2: sum `total_without_discount_hkd` across all receipts The final output is a single numeric amount (HKD).

---

3. Key Implementation Details

3.1 Structured output and robust parsing

To reduce output-format drift (e.g., extra explanations or markdown), prompts enforce **JSON-only** outputs. On the program side:

- Parse JSON directly via `json.loads`; if that fails, extract the first `{...}` block using regex and parse again
- Normalize money strings into numeric values (remove currency symbols/commas, etc.)
- Ensure `total_discount_hkd` is non-negative; if `total_without_discount_hkd` is missing, fall back to `paid + discount`

3.2 Ignoring balance/points (non-spending signals)

Receipts often contain lines such as “Remaining Value”, “Point Balance”, or similar. These are **not** discounts and **not** spending amounts. The extraction prompt explicitly instructs the model to ignore these lines to prevent miscounting.

---

4. Parallelization

Each receipt can be processed independently. Therefore, the solution uses parallelization to reduce end-to-end latency:

- Use `asyncio.gather` to process multiple receipt images concurrently
- Use `Semaphore(concurrency)` to cap concurrency and reduce the risk of rate limiting

This follows the Parallelization Pattern: **parallel independent subtasks → merge/summarize results**.

---

5. Out-of-domain Rejection

The router classifies the user query into three categories:

- Query1 / Query2: perform extraction + aggregation to answer
- reject: return a fixed refusal message indicating only Query1/Query2 are supported

A conservative fallback strategy is used: if routing fails (e.g., invalid JSON), the system rejects by default.

---

6. Evaluation and Results

The implemented pipeline passes the provided Query1 and Query2 evaluation cells in the notebook.