5660 HW1 Report
LangChain Receipt QA Agent

Name: Liu Sixiao
Student ID:1155247019

1. Overview

This project builds a LangChain-based system that takes multiple supermarket receipt inputs (extracted text through OCR) with user querys, and returns answers for two supported queries:

- **Query 1**: How much money did I spend in total for these bills?
- **Query 2**: How much would I have had to pay without the discount?
- **Irrelevant query handling**: refuse to answer if the input question is unrelated to the receipts

2. System Design

Input representation

All receipts are concatenated into a single string *receipt_content*. To avoid cross-receipt contamination, I enforce explicit session boundaries:

```
----begin---- ... receipt text ... ----end----
```

This boundary design makes it easier for the model to treat each receipt as an independent session and reliably extract totals/discounts per session.

LLM and LangChain stack

- LLM: ChatGoogleGenerativeAI (Gemini), with **temperature=0** for stable outputs.
- Prompting: *ChatPromptTemplate.from_messages()* using a system role + user instruction.
- Composition: LCEL pipeline pattern *prompt* | *llm* for clean modular chaining.
- Output handling: an extra normalization step to guarantee **float-parseable** numeric outputs.

3. Chains and Prompt Engineering

I implemented one dedicated pipeline for each query, plus a shared idea of output normalization to ensure the final answer can be directly cast to *float*.

3.1 Query 1: Total Amount Spent (Summation)

Chain definition: **full_query1_pipeline = prompt1 | llm**.

Prompt1 instructs the model to scan each receipt session (bounded by begin/end), extract the session total (e.g., TOTAL/AMOUNT PAID), and sum across sessions.

The output must contain **only the final numeric result** with no reasoning text.

Output normalization:

To prevent formats like “Total: $123.45”, I apply a second prompt (result-transformation) that outputs only a numeric string.

3.2 Query 2: Payment Without Discount (Avoid Double Counting)

Query 2 is harder because discounts can appear in multiple places (item lines, discount sections, or subtotals). I use a two-step approach: (A) discount-focused extraction to reduce noise and then (B) calculation prompt that adds back discounts carefully without double-counting.

- **Step A (filter)**: Extract only discount-related lines/sections plus each session total.
- **Step B (compute)**: Use prompt2 to compute “pay without discount” while checking for discount duplication.
- **Normalization**: Apply a final transformer prompt to output a float-compatible numeric string.

4. Query Handling and Rejection

To satisfy the requirement of rejecting irrelevant queries, I implemented a **prompt-level rejection policy** inside the system messages of both Query 1 and Query 2 pipelines. Conceptually, the model first checks whether the input is receipt-related; if not, it should return a refusal response instead of fabricating an answer.

In the notebook, I also add a simple sanity test by sending an unrelated question to the pipeline (e.g.,“What’s the weather today in HK?”)

```
Irrelevant_response = full_query2_pipeline.invoke("What's the weather today in HK?")
```

5. Evaluation

I validate outputs using a lightweight test function that compares the model answer against ground-truth totals with a tolerance of ±2 to account for OCR noise and rounding differences.

This ensures: (1) outputs are numeric and parseable, and (2) totals are close to the expected sums.

6. Discussion and Future Improvements

- **Strengths**: receipt session boundaries, task-specific prompts, and strict numeric normalization improve stability.
- **Discount realism**: in real receipts, discounts often appear as negative lines (e.g., -1.50). A future improvement is to classify negative amounts using keywords (discount,refund,return).
- **Consistency**: unify delimiter strings across all steps to avoid ambiguity.
- **Agent completion**: wrap Query1/Query2 as LangChain tools and add a router to fully satisfy query rejection.

7. Conclusion

Overall, the solution demonstrates a clean post-OCR LangChain design: multi-receipt text structuring, two task pipelines for Query1 and Query2, output normalization for float parsing, and automatically rejecting irrelevant queries.