Receipt Evaluation Report

Github source: https://github.com/SaKamiAn/FTEC5660

Colab source: https://colab.research.google.com/drive/1FTJ8mvW4c3jTVPXHIunY3nVK-5uYVZHe#scrollTo=2D4vHPk2joH_

1. Problem Background

In this task, **7 receipt images** are given and we are required to answer two evaluation queries with agentic AI method.

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?

The core challenge is that the information (total paid, discounts) must first be **extracted from images using a multimodal LLM (Gemini)**, and then **calculated deterministically by code** to ensure evaluation stability.

2. Key Difficulties

a) **Multimodal Output Is Not Guaranteed to Be Pure JSON**
Even when explicitly asking for JSON, LLMs may prepend explanations, markdown fences, or natural language text, which causes json.loads() to fail.

b) **Separation of Responsibilities**
LLMs are good at *information extraction*, but numerical computation must be handled by deterministic code to avoid hallucination and ensure reproducibility.

c) **Multiple Receipts Aggregation**
We must first extract per-receipt values, then aggregate across all 7 receipts to answer the queries.

3. Solution Strategy

The solution is divided into three clear stages:

Stage 1: Multimodal Information Extraction

Use the **Google Generative AI SDK (Gemini)** to process 7 receipt images.
Prompt the model to extract, for each receipt:
total_paid
total_discount
The model is instructed to return results in a strict JSON schema.

Stage 2: Robust JSON Parsing

Because the model output may still contain extra text, we apply **defensive parsing**:

Extract the first valid JSON object from the model output using a regular expression.
Parse it safely using json.loads().
This step ensures robustness and prevents JSONDecodeError.

Stage 3: Deterministic Computation
Once the structured data is available:
Query 1 is answered by summing all total_paid values.
Query 2 is answered by summing (total_paid + total_discount) for each receipt.
All calculations are done purely in Python to ensure correctness and reproducibility.

4. Final Output Format
The final output follows a standardized evaluation-friendly structure:
a) Per-receipt extracted values
b) Final numeric answers for Query 1 and Query 2
This design aligns with common benchmark and homework evaluation pipelines.

5. Key Takeaways
LLMs should be treated as **probabilistic extractors**, not trusted calculators.
Always assume LLM output may violate formatting constraints.
Deterministic code is essential for numerical evaluation tasks

Appendix with some information from Google

I failed with Langchain because of some reports of the error. So, I turned to Google GenAI SDK to meet the requirement of the Homework1. And there are some following reasons.

1. Dependency Stability and Environment Compatibility

LangChain introduces multiple transitive dependencies, including optional integrations with deep learning frameworks such as **PyTorch**.
In managed notebook environments (e.g., Google Colab), this may trigger **unexpected runtime conflicts**, such as namespace registration errors in torch.library, even when the user does not explicitly invoke any torch-related functionality.
By contrast, the GenAI SDK:

- Has a **minimal dependency surface**
- Does not implicitly load PyTorch or Triton
- Is significantly less prone to environment-level runtime errors

This makes it more suitable for constrained or preconfigured execution environments used in assignments and benchmarks.

2. Explicit Control Over Multimodal Inputs

When processing multiple receipt images, precise control over:

- Image encoding (base64)
- Input ordering
- Prompt structure

is critical.
The GenAI SDK allows direct construction of multimodal inputs without additional abstraction layers.
This reduces hidden transformations and makes the data flow **fully explicit and auditable**, which is important for debugging and reproducibility.

3. Predictable Output Handling for Evaluation

This task requires **strictly structured outputs** to support deterministic downstream computation and evaluation.
Using the GenAI SDK:

- The raw model output is directly accessible
- Output post-processing (e.g., JSON extraction) is fully controlled by user code
- There is no hidden prompt templating or output wrapping

This aligns well with benchmark-style evaluation pipelines, where **format stability is more important than developer convenience**.

4. Clear Separation of Responsibilities

The chosen design enforces a clean separation:

- **LLM** → probabilistic information extraction from images
- **Program code** → deterministic numerical computation and aggregation

Using the GenAI SDK keeps this boundary explicit and avoids over-abstracting LLM calls into chains that may blur responsibility between extraction and reasoning.

Summary

LangChain is a powerful framework for complex agentic workflows.
However, for a **multimodal extraction + deterministic evaluation task**, the GenAI SDK provides:

- Higher runtime stability
- Lower abstraction overhead
- Greater control over inputs and outputs

Therefore, it is the more appropriate choice for this assignment.