SEEM5660 Individual Homework 01 Report
Multimodal Supermarket Receipt QA with Gemini

1 Overview

This homework builds a multimodal question-answering system for supermarket receipt photos.

The input is a folder of JPEG images (multiple receipts) and a user text question. The system is designed to answer only two types of queries required by the assignment: (1) the total amount paid across all receipts, and (2) the amount that would have been paid if there were no discounts. If the user asks anything unrelated to these two billing questions, the system should refuse the request instead of hallucinating an answer. The final deliverable is a runnable Python program and a short report describing the design choices and evaluation approach.

2 Method

My implementation uses the Google GenAI Python SDK to call a Gemini multimodal model and let the model read the receipt images directly. The program first scans the `./photos` directory recursively, collects all `.jpg/.jpeg` files, sorts them by filename to keep a consistent order, and loads each image as raw bytes. Each image is converted to a multimodal input part with `types.Part.from_bytes(..., mime_type="image/jpeg")`. After that, a `genai.Client` is created with a user-provided API key and an optional API base URL (a default value is used if the user presses Enter). I then create a chat session using the target model (`gemini-3-pro-preview`) and set a low temperature to make numerical outputs more stable. The key idea is to send the entire batch of receipt images once as the initial context message (e.g., "these are receipt photos" plus the image parts), and then keep the chat session alive so that the model can answer multiple questions based on the same set of receipts without re-uploading the images every time.

3 Prompt Design and Refusal Handling

To satisfy the requirement of answering only billing-related questions, I rely on a system instruction that explicitly constrains the assistant behavior: the model is told that it will receive multiple receipt images, it must answer the user query only when it is about the receipts, and it must reject irrelevant queries. In practice, this prompt acts as a policy layer that reduces the chance of the model responding to out-of-scope questions such as general knowledge, personal advice, or requests not related to totals/discounts. When a query is in-scope, the model is expected to extract the necessary monetary information from the receipts (such as the final paid totals, and the pre-discount totals or discount lines) and then compute the final answer. When a query is out-of-scope, the model should respond with a short refusal and guide the user back to the two supported question types. This approach keeps the implementation simple and does not require training a classifier or writing custom rules for every possible unrelated question; instead, the restriction is enforced at the system prompt level.

4 Experimental Setup and Evaluation

The program is evaluated interactively by asking questions after the receipt images are loaded. This matches the grading setting described in the assignment, where the evaluator will use a new set of receipt photos and ask a collection of random questions (including both required queries and irrelevant ones). For correctness, I focus on two aspects: numeric accuracy for the two supported queries, and reliable refusal for irrelevant questions. To improve robustness, the decoding temperature is set low, and the same chat context is reused so that the model has access to all receipts simultaneously when computing a global total. During manual testing, I also check edge cases such as having multiple receipts with different discount formats, receipts that display both "subtotal" and "total," and receipts where discounts are listed as separate negative items. The expected behavior is that the model consistently chooses the correct monetary field (final payable amount for Query 1, and the pre-discount amount for Query 2) and sums across all receipts.

5 Limitations and Future Improvements

A limitation of the current solution is that it does not call a tool (e.g., a calculator) for arithmetic, so adding a calculator tool for deterministic computation would improve numerical reliability.