1. Introduction

This report is about what I did for FTEC5660 Homework1. Basically, we had to make something that can look at supermarket receipt photos and answer questions about them. Honestly when I first read the requirements, I thought it would be quick to finish since Gemini and similar models are supposed to be good with images. Turns out I was wrong and ran into more problems than I expected.

The system needs to handle three scenarios. First, telling the user how much they spent in total. Second, calculating what the user would have paid without any discounts. Third, recognizing when a question is unrelated to receipts and refusing to answer it.

This report covers my thought process, the different methods I experimented with, the issues I encountered, and my final working solution.

2. Understanding the Problem

I started by examining the receipt images from the assignment before writing any code. The receipts are from various Hong Kong supermarkets with different formats. Some have English text, others include Chinese characters, and the layouts are not consistent across receipts.

I quickly realized that receipts contain many types of numbers. There are item prices, discount amounts shown as negative values, subtotals, payment amounts from OCTOPUS or VISA and sometimes balance figures. Just asking the model to add everything up would obviously fail. The system needs to distinguish which numbers matter for each query type.

For Query 1, we need the final payment amount, typically displayed next to OCTOPUS, VISA, or as Net Total. For Query 2, we need to figure out the original price by identifying discount amounts and adding them back to the final payment.

3. My First Attempt (And Why It Failed)

I started with what I thought was a straightforward approach. I wanted to write a single prompt that sends all receipt images to Gemini and gets the calculated answer directly. For Query 1, my prompt was something like "Sum all final payment amounts from all receipts and return only the number." For Query 2, I asked it to add back negative amounts to get original prices.

I expected this to work since Gemini 2.5 Flash is supposed to be capable of both vision tasks and basic arithmetic. The actual results did not match my expectations.

The OCR functionality worked reasonably well. I examined what the model was reading from the images, and it was picking up the numbers accurately. The strange part was the math. The model would extract correct values but produce incorrect sums. I saw cases where it read five numbers correctly, but the total was wrong by ten or twenty HKD.

I tried adjusting my prompts multiple times, wondering if my instructions were confusing. Eventually I accepted that this is simply a weakness of language models. They process text tokens, not numerical operations the way a calculator does. This was a useful realization because I learned not to rely on LLMs for tasks they are not designed for.

4. My Final Approach

Based on what I learned from the failed attempt, I changed my strategy. The new idea was simple: use the model for what it does well and use Python for what Python does well.

4.1 Two-Stage Pipeline

My final solution has two separate stages. Stage one uses Gemini purely as an OCR tool to pull structured information from each receipt. Stage two uses Python to do all arithmetic. This division of work makes the system much more dependable since Python arithmetic is always correct.

4.2 Designing the OCR Prompt

Getting the extraction prompt right took multiple tries. The final version asks Gemini to output three categories from each receipt: original item prices as positive numbers, discount amounts as negative numbers, and the final payment figure.

Figuring out what qualifies as a discount was harder than I anticipated. I discovered through trial and error that receipts show various negative numbers that are not actually discounts. ROUNDING is a small adjustment for rounding to the nearest ten cents. Balance figures show remaining card value. OCTOPUS or VISA lines show payment amounts, not savings.

My early prompts did not mention these distinctions. The model kept including balance amounts as discounts, which completely broke Query 2 results. After I added specific rules telling the model what to ignore, the extraction became accurate.

4.3 Routing Queries

I handle query classification using keyword detection in Python. When the query includes terms like "total", "spend", "spent", "pay", "paid", or "cost" without "without" or "discount" appearing, I treat it as Query 1 and sum up the final prices.

When the query has words like "without", "before", or "original" along with "discount", I classify it as Query 2 and return final prices plus discount amounts combined.

Any other query gets sent to the model along with the receipt images. I instruct the model to respond with "IRRELEVANT_QUERY" if the question does not relate to the receipts.

5. Implementation Details

I wrote the code in Python with LangChain for interacting with the Gemini API. Below are some specific points about the implementation.

5.1 Processing Images

Receipt images get converted to base64 encoded strings and wrapped as data URLs. The Gemini API requires this format for image inputs. Two helper functions, image_to_base64 and get_image_data_url, perform this conversion.

5.2 Setting Up the API

I ran into an authentication problem at first. My API key came from Google AI Studio but I had mistakenly set vertexai=True in the code. This caused errors because Google AI Studio and Vertex AI use separate authentication mechanisms. Switching to vertexai=False fixed the issue right away.

5.3 Parsing Model Output

The model returns structured text with labeled sections for prices, discounts, and final amount. My code goes through this text line by line, keeps track of which section it is reading, and pulls out numbers with regular expressions.

6. Testing and Results

I evaluated my system with the test code from the assignment materials.

Query 1 has a ground truth of HKD 1974.3, which is the sum of 394.7, 316.1, 140.8, 514.0, 102.3, 190.8, and 315.6. My system produced an answer within the allowed two-dollar margin.

Query 2 has a ground truth of HKD 2348.2, summing 480.20, 392.20, 160.10, 590.80, 107.70, 221.20, and 396.00. My answer also fell within the tolerance range.

I tested irrelevant queries with questions like "How old are you?" and "What is your name?" that obviously have nothing to do with shopping receipts. The system correctly returned "IRRELEVANT_QUERY" for these cases.

7. Lessons Learned

Working on this homework gave me some practical insights about building applications with large language models.

One thing I learned is that these models are not equally good at everything. Vision and language understanding are their strong points, but arithmetic is surprisingly unreliable. Knowing this, I designed my system to keep calculation tasks separate from what the model handles.

Something else I picked up is that prompts rarely work on the first try. My early attempts left out important details, and the model did weird things as a result. I remember one version where it kept treating payment amounts as discounts, which made no sense to me at first. I had to go back, add more specific instructions about what to ignore, run it again, and see if anything improved. This back and forth happened maybe four or five times before I got output that made sense.

Debugging this kind of application felt quite different from debugging regular programs. With normal code, if something is broken, you usually get an error message pointing to the problem. But the model does not complain when it misunderstands your prompt. It just quietly returns a wrong answer and moves on. I spent a lot of time printing out what the model was returning at each step so I could trace back where things were going off track.

8. Conclusion

For this assignment, I created a receipt analysis tool that answers spending questions and filters out unrelated queries. The main lesson was recognizing that LLMs should handle perception tasks while traditional code handles computation. Going through this project helped me understand prompt design better and gave me hands-on experience with multimodal AI systems. I expect these skills to be relevant for other work in this course.