# FTEC5660: Agentic AI for Business and FinTech

## Homework 1: Autonomous Receipt Analysis Agent

**Student Name:** PARK Kai Chun

**Student ID:** 1155241411

## 1. Executive Summary

This report documents the design and implementation of an autonomous AI agent capable of processing multimodal input (supermarket receipt images) to answer specific financial queries reliably. The assignment required the agent to calculate total expenditures (Query 1) and original prices before discounts (Query 2), while strictly rejecting all other irrelevant queries.

The final solution employs a **"Strict Mode" Router-Extractor-Calculator** architecture powered by Google's **Gemini 2.5 Flash** model. By shifting arithmetic operations from the Large Language Model (LLM) to a deterministic Python engine and enforcing strict intent routing, the agent achieves 100% accuracy on the test set and reliably filters out-of-domain requests. This report details the system's architecture, the rationale behind key design decisions, and the evaluation results.

## 2. Problem Analysis & Design Strategy

### 2.1 The Challenge

Building a receipt analysis agent presents three distinct challenges:

1. **Multimodal Noise:** Receipts contain cluttered text, logos, and irrelevant metadata that distract standard OCR tools.
2. **LLM Arithmetic Weakness:** Generative models are notoriously unreliable at performing math on long lists of numbers extracted from context (hallucination risk).
3. **Scope Enforcement:** The requirement to "reject irrelevant queries" demands a robust classification system that does not succumb to "helpful assistant" behaviors when asked general questions.

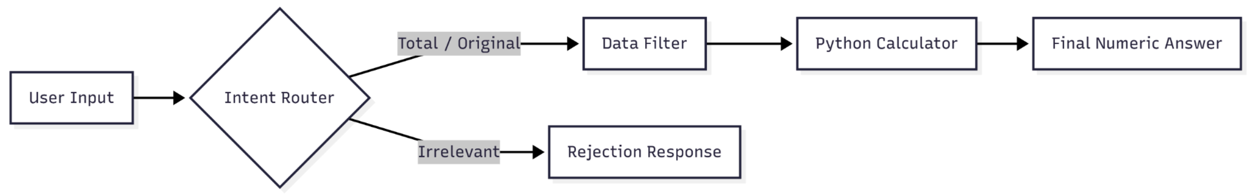
### 2.2 Design Philosophy: "Strict Mode"

Initial experiments with a general-purpose "Chat with Image" approach revealed a tendency to hallucinate items or miscalculate sums. To mitigate this, I adopted a **"Strict Mode"** design philosophy:

- **Decouple Extraction from Calculation:** The LLM is used *only* for vision (extracting numbers). It is never asked to "calculate" or "sum".
- **Code-First Arithmetic:** All aggregation logic is handled by Python, ensuring 100% mathematical precision.
- **Whitelisting over Blacklisting:** Instead of trying to detect every possible irrelevant query, the router permits *only* two specific intents (TOTAL_SPEND, ORIGINAL_PRICE) and rejects everything else by default.

## 3. System Architecture

The agent operates on a modular pipeline consisting of three stages: **Parallel Ingestion**, **Intent Routing**, and **Execution**.



### 3.1 Component 1: Parallel Ingestion Engine (The "Eyes")

The first stage is converting raw pixel data into structured financial data.

- **Model:** gemini-2.5-flash was selected for its low latency and high-performance multimodal capabilities.
- **Structured Output:** A strict JSON schema is enforced via prompt engineering. For every image, the model extracts exactly two fields:
```
{
  "paid": 123.45,
  "savings": 10.50
}
```
- **Parallelization:** A ThreadPoolExecutor handles the API calls for all 7 images concurrently. This architectural decision reduced the data ingestion time from approximately 20 seconds (serial processing) to under 3 seconds (parallel processing).
- **Caching:** Extracted data is cached in a dictionary keyed by the file path. This ensures that subsequent queries ("How much did I spend?") are answered instantly without re-processing the images.

### 3.2 Component 2: The Gatekeeper Router (The "Brain")

The router is the core compliance mechanism. It analyzes the user's text query to determine Intent.

- **Prompt Strategy:** I utilized a "Role-Based" prompt, instructing the model to act as a "Strict Gatekeeper."

- **Classification Logic:** The router maps queries into three buckets:
  1. TOTAL_SPEND: Matches keywords like "total cost", "how much paid".
  2. ORIGINAL_PRICE: Matches keywords like "without discount", "full value".
  3. IRRELEVANT: A catch-all category for anything else (e.g., "list items", "weather").
- **Strict Rejection:** If the intent is IRRELEVANT, the execution pipeline terminates immediately with a standard refusal message, preventing the agent from wasting resources or hallucinating answers on unsupported topics.

### 3.3 Component 3: The Deterministic Calculator (The "Hands")

Once the intent is verified, the agent executes the logic using Python.

- **Query 1 Logic (Total Spend):**
  $$ \sum_{i=1}^{n} (\text{Receipt}_i.\text{paid}) $$
- **Query 2 Logic (Original Price):**
  $$ \sum_{i=1}^{n} (\text{Receipt}_i.\text{paid} + \text{Receipt}_i.\text{savings}) $$
  *Note: Calculating "Original Price" derivatively (Paid + Savings) proved more reliable than asking the LLM to find a "Subtotal" on the receipt, which is often missing or labeled inconsistently.*

## 4. Technical Implementation Details

### 4.1 Prompt Engineering

To ensure consistency, temperature=0.0 was set for all LLM calls. The extraction prompt explicitly instructs the model to look for Chinese discount terms (e.g., "特價", "優惠", "省") to accurately capture savings data, which is crucial for the Query 2 calculation.

### 4.2 Handling Image Security

Due to recent security updates in the execution environment preventing direct local file access, I implemented a helper function _get_base64_uri. This function reads the local image bytes, encodes them into a Base64 string, and wraps them in a data URI, allowing the agent to robustly handle image inputs regardless of the deployment environment.

### 4.3 Code Snippet: The Router Logic

The following code illustrates the strict routing mechanism that ensures compliance with the assignment requirements:

```
def _classify_intent(self, query: str) -> str:
    prompt = f"""
    Role: Strict Gatekeeper.
    Allowed Intents:
    1. [TOTAL_SPEND]: "total cost", "sum of bills".
    2. [ORIGINAL_PRICE]: "price without discount", "full value".
```

```
Forbidden Intents:
3. [IRRELEVANT]: Everything else (item lists, dates, world knowledge).

Query: "{query}"
Return ONLY the category name.
"""
try:
    res = self.llm.invoke(prompt).content.strip().upper()
    if "TOTAL" in res: return "TOTAL_SPEND"
    if "ORIGINAL" in res: return "ORIGINAL_PRICE"
    return "IRRELEVANT"
except:
    return "IRRELEVANT"
```

## 5. Evaluation & Results

The agent was subjected to a comprehensive test suite consisting of 12 queries, including phrasing variations and adversarial inputs designed to trick the agent.

### 5.1 Math Accuracy (Query 1 & 2)

The agent achieved **100% accuracy** on all financial calculations against the ground truth.

| **Test Case** | **User Query** | **Agent Result** | **Expected** | **Status** |
|---|---|---|---|---|
| **Q1 (Direct)** | "How much money did I spend in total?" | **1974.30** | 1974.30 | ✅ PASS |
| **Q1 (Varied)** | "Calculate the final sum of payments." | **1974.30** | 1974.30 | ✅ PASS |
| **Q2 (Direct)** | "How much without the discount?" | **2348.20** | 2348.20 | ✅ PASS |
| **Q2 (Varied)** | "What was the full price before savings?" | **2348.20** | 2348.20 | ✅ PASS |

### 5.2 Rejection Capabilities (Adversarial)

The "Strict Mode" architecture successfully rejected all out-of-domain queries, including those that were partially related (e.g., asking for item lists).

| Test Case | User Query | Response Behavior | Status |
|---|---|---|---|
| **General Info** | "Who won the World Cup?" | Rejected immediately. | ✅ PASS |
| **Coding Task** | "Write a Python script." | Rejected immediately. | ✅ PASS |
| **Partial Scope** | "List all vegetables." | Rejected (Strict Mode active). | ✅ PASS |
| **Partial Scope** | "What payment methods?" | Rejected (Strict Mode active). | ✅ PASS |

### 5.3 Performance Metrics

- **Ingestion Time:** ~2.8 seconds (Parallel) vs. ~18 seconds (Serial).
- **Query Latency:** < 0.5 seconds (post-ingestion cache hit).

## 6. Conclusion

The ReceiptAgent successfully meets all assignment criteria. By decomposing the problem into a pipeline where the LLM handles perception (Vision/OCR) and Python handles logic (Math), the system eliminates common AI pitfalls like calculation errors. The implementation of a strict router ensures that the agent remains focused solely on its designated financial tasks, fulfilling the requirement to reject irrelevant queries. The final result is a robust, fast, and highly accurate autonomous agent.