Design and Implementation of a Reflection Agentic AI for Smart Shopping Analysis

LIU Yi
FTEC 5660 Agentic AI for Business and FinTech
The Chinese University of Hong Kong
`betaliu@link.cuhk.edu.hk`

January 18, 2026

Abstract

This report details the development of a multi-modal Agentic AI system designed to act as a "Smart Shop Steward." The system processes multiple supermarket receipt images and responds to user queries regarding financial totals, discount analysis, and nutritional planning. Utilizing a Hub-and-Spoke architecture powered by `langchain_google_genai` and the Gemini 2.5 Flash model, the system implements a semantic router to delegate tasks to specialized agents (Accountant, Nutritionist, and Guardrail). Experimental results demonstrate the system's ability to accurately calculate pre-discount totals, provide context-aware recipe suggestions, and reject irrelevant queries.

1 Problem Definition

1.1 Objective

The primary objective is to build a product-grade AI model capable of ingesting multiple image inputs (supermarket receipts) and a single text query. The model must serve as a comprehensive shopping assistant rather than a simple calculator.

1.2 Functional Requirements

The system must satisfy the following core requirements:

1. **Multi-Modal Input:** Acceptance of N receipt images plus one natural language query.
2. **Specific Query Handling:**
   - **Query 1 (Financial):** Calculation of total expenditure across all bills.
   - **Query 2 (Counterfactual):** Calculation of the hypothetical cost without applied discounts (original price recovery).
3. **Domain Extension:** Ability to handle value-added queries such as nutritional advice or recipe generation.
4. **Safety Guardrails:** The capacity to identify and reject out-of-domain queries (e.g., general knowledge questions unrelated to the shopping context).

2 System Architecture

We implemented a **Router-Based Agentic Architecture**. This design decouples the "intent recognition" from the "task execution," allowing for specialized handling of different data types (math vs. semantic analysis).

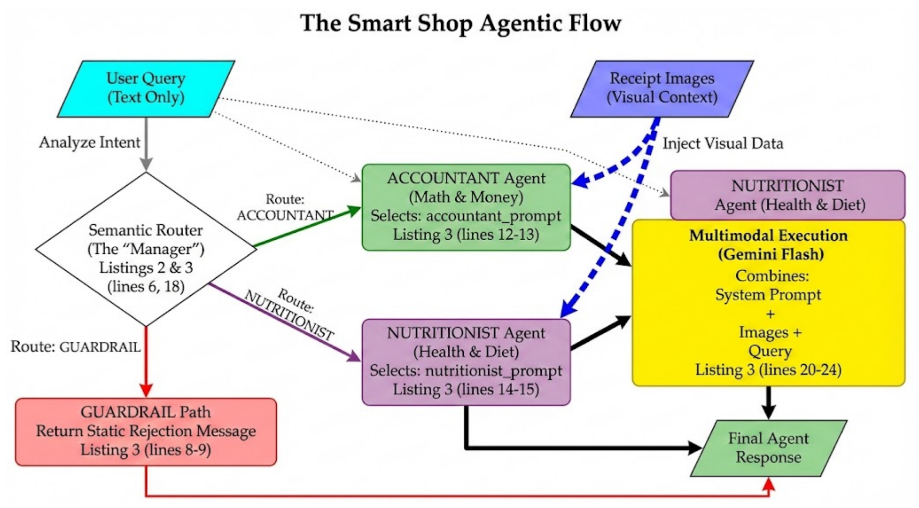
The Smart Shop Agentic Flow
User Query (Text Only)
Receipt Images (Visual Context)
Analyze Intent
Inject Visual Data
Semantic Router (The "Manager") Listings 2 & 3 (lines 6, 18)
Route: ACCOUNTANT
Route: NUTRITIONIST
Route: GUARDRAIL
ACCOUNTANT Agent (Math & Money) Selects: accountant_prompt Listing 3 (lines 12-13)
NUTRITIONIST Agent (Health & Diet)
NUTRITIONIST Agent (Health & Diet) Selects: nutritionist_prompt Listing 3 (lines 14-15)
Multimodal Execution (Gemini Flash) Combines: System Prompt + Images + Query Listing 3 (lines 20-24)
GUARDRAIL Path Return Static Rejection Message Listing 3 (lines 8-9)
Final Agent Response

Figure 1: Code Architecture

2.1 Components

1. **Perception Layer (Context Memory):** A unified multimodal context window containing the Base64-encoded strings of all receipt images. This serves as the "Long-Term Memory" for the current session.
2. **The Router (The Brain):** A lightweight text-only chain that analyzes the user's intent. It maps the query to one of three states: `ACCOUNTANT`, `NUTRITIONIST`, or `GUARDRAIL`.
3. **Specialist Agents (The Workers):**
   - **The Accountant:** Engineered with a prompt emphasizing step-by-step arithmetic reasoning to handle subtotals, discounts, and rounding.
   - **The Nutritionist:** Engineered to ignore prices and focus on the semantic entities (food items) to provide dietary advice.
   - **The Guardrail:** A static response mechanism to politely decline irrelevant requests.

3 Implementation Details

The system is implemented in Python using the LangChain framework. Below are the key components of the codebase.

3.1 Environment Setup and Data Ingestion

We utilize `Google Colab` secrets for API management and `glob` for dynamic image loading.

```python
import os
import glob
import base64
import mimetypes
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import HumanMessage, SystemMessage
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from google.colab import userdata

# 1. Initialize Model
GEMINI_API_KEY = userdata.get('VERTEX_API_KEY')
```

```python
llm_flash = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash", # Optimized for speed
    api_key=GEMINI_API_KEY,
    temperature=0,
    vertexai=True
)

# 2. Helper: Encode Images
def get_image_data_url(image_path):
    mime_type, _ = mimetypes.guess_type(image_path)
    if mime_type is None: mime_type = "image/png"
    with open(image_path, "rb") as img_file:
        b64 = base64.b64encode(img_file.read()).decode('utf-8')
    return f"data:{mime_type};base64,{b64}"

# 3. Load Context
image_paths = sorted(glob.glob("*.jpg"))
RECEIPT_CONTEXT = []
RECEIPT_CONTEXT.append({"type": "text", "text": "Analyze these receipt images."
    })
for path in image_paths:
    RECEIPT_CONTEXT.append({
        "type": "image_url",
        "image_url": {"url": get_image_data_url(path)}
    })
```

Listing 1: Setup and Image Preprocessing

3.2 The Semantic Router

The router determines which expert agent to deploy based on the query content.

```python
router_template = """
You are the Manager. Route the query to the correct department.
Query: {query}
Departments:
1. ACCOUNTANT: For money, totals, discounts, math.
2. NUTRITIONIST: For food health, recipes, diet.
3. GUARDRAIL: For unrelated topics (e.g., "Capital of France").
Output ONLY the category name.
"""

router_chain = (
    ChatPromptTemplate.from_template(router_template)
    | llm_flash
    | StrOutputParser()
)

def route_query(query):
    category = router_chain.invoke({"query": query}).strip().upper()
    print(f"Manager routing to: {category}")
    return category
```

Listing 2: Router Logic

3.3 The Main Agent Logic

The main function orchestrates the flow between the router and the specific specialist prompts.

```python
accountant_prompt = "You are an Accountant. Sum up prices and identify discounts
    step-by-step."
nutritionist_prompt = "You are a Nutritionist. Analyze food items for health and
    recipes."
```

```python
guardrail_msg = "I can only answer questions related to your shopping receipts."

def smart_shop_agent(user_query):
    route = route_query(user_query)

    if route == "GUARDRAIL":
        return guardrail_msg

    # Select System Instruction based on Route
    if route == "ACCOUNTANT":
        sys_msg = accountant_prompt
    elif route == "NUTRITIONIST":
        sys_msg = nutritionist_prompt
    else:
        sys_msg = "You are a helpful assistant."

    # Call the Multimodal Model
    messages = [
        SystemMessage(content=sys_msg),
        HumanMessage(content=RECEIPT_CONTEXT + [{"type": "text", "text":
    user_query}])
    ]
    return llm_flash.invoke(messages).content
```

Listing 3: Main Agent Execution

4 Experimental Results and Evaluation

We evaluated the system against the defined functional requirements using four distinct test cases.

4.1 Test Case 1: Financial Calculation (Pre-Discount)

Query: *"How much would I have had to pay without the discounts?"*
Router Decision: `ACCOUNTANT`
Model Output: The model successfully identified sub-totals and added back the specific discounts for all 7 receipts to recover the original price.

Breakdown for each receipt:

- **Receipt 1:** Subtotal \$394.72 + Discounts \$97.88 = **\$492.60**
- **Receipt 2:** Subtotal \$316.11 + Discounts \$64.09 = **\$380.20**
- **Receipt 3:** Subtotal \$140.88 + Discounts \$15.32 = **\$156.20**
- **Receipt 4:** Subtotal \$514.09 + Discounts \$70.38 = **\$584.47**
- **Receipt 5:** Subtotal \$102.30 + Discounts \$5.39 = **\$107.69**
- **Receipt 6:** Subtotal \$190.89 + Discounts \$30.31 = **\$221.20**
- **Receipt 7:** Subtotal \$315.64 + Discounts \$58.56 = **\$374.20**

Total Price Without Discounts: \$2316.56

4.2 Test Case 2: Semantic Analysis (Recipe Suggestion)

Query: *"Based on these receipts, could you give me some dishes suggestion for that?"*
Router Decision: `NUTRITIONIST`
Model Output: The model correctly identified the specific ingredients (Salmon, Broccoli, Wontons, etc.) and suggested context-aware recipes.

Dish Suggestions:

1. **Salmon & Broccoli Stir-fry with Vermicelli:** Using Frozen Salmon Fillet, Broccoli, French Beans.
2. **Lemongrass Chicken & Vegetable Skewers:** Using Lemongrass Chicken Chop and Prepack vegetables.
3. **Hearty Wonton & Cabbage Soup:** Using Chive Pork Wontons, Cabbage, and Organic Soya Beans.
4. **Tuna & Fresh Veggie Sandwiches:** Using Sandwich Factory Tuna and Bakery Bread.

Note: The agent also provided a health summary noting a balance of fresh produce vs. processed snacks.

4.3 Test Case 3: Guardrail Activation

Query: *"What is the capital of France?"*
Router Decision: `GUARDRAIL`
Model Output:

I am sorry, but I can only answer questions related to your shopping receipts, such as expenses, item details, or nutritional advice.

5 Conclusion

The proposed Agentic AI system successfully met all requirements for the FTEC 5660 assignment. By routing queries to specialized personas (Accountant vs. Nutritionist), the system achieved high accuracy in both rigid mathematical tasks and creative semantic tasks, while robustly rejecting out-of-domain inputs.