Agentic AI for Recognizing Receipt Amounts and Discounts - Implementation Logic Explanation

Student ID: 1155241476

1. Overall Execution Flow

This agentic AI system is designed to answer user queries related to supermarket receipts.

The system follows a sequential execution pipeline that mirrors the agentic architecture introduced in the course tutorials.

First we give a user query and a set of receipt images, the system will determine the user's intent, then activate downstream agents and tools to generate the final answer.

This design ensures that computationally expensive multimodal processing is only performed when the user query is relevant, and that all numerical reasoning is handled deterministically outside the language model to prevent hallucination.

---

2. Query Routing Logic

Upon receiving a user query, the Query Router Agent(`query_router_agent = query_router_prompt | llm`) is invoked to classify the intent of the request.

The router uses a constrained language model prompt to map natural language input to one of three predefined labels: **Query_1**, **Query_2**, or **REJECT**.

The raw language model output is subsequently post-processed using a lightweight normalization function(`normalize_route`), which converts the model response into deterministic control signals (`cost` , `original` , or `irrelevant`).

`router_chain = query_router_agent | RunnableLambda(normalize_route)`

This additional processing step ensures that downstream agent operates on stable and predictable values rather than free-form text generated by the model.

---

3. Conditional Execution via Branching

Based on the normalized routing decision, the system uses conditional branching to determine the subsequent execution path.

```
branch = RunnableBranch(
    (lambda x: x["route"] == "cost", RunnableLambda(handle_extraction_and_calc)),
    (lambda x: x["route"] == "original", RunnableLambda(handle_extraction_and_calc)),
    RunnableLambda(handle_irrelevant)
)

full_chain = ( {"route": router_chain} | branch)
```

- If the query is relevant, the system proceeds to extract information from receipt images and perform numerical calculations.
- If the query is irrelevant, the system immediately returns a rejection message without invoking the receipt extraction agent.

This branching mechanism improves efficiency and robustness by preventing unnecessary multimodal inference for unsupported queries.

---

4. Receipt Extraction Agent Logic

For relevant queries, the Receipt Extraction Agent(`extraction_chain = extraction_prompt | llm_structured`) is activated to process receipt images.
Each image is handled independently, and the agent extracts structured financial information using a predefined schema.

For each receipt, the agent outputs a structured data object containing:

- The final amount actually paid by the customer
- A list of individual discount amounts

Structured output constraints are applied to enforce schema compliance, ensuring that extracted values can be safely consumed by downstream components.

---

5. Batch Processing of Multiple Receipts

To support multiple receipt images, the extraction agent is executed in batch mode.
This design allows the system to scale naturally with the number of receipts provided by the user, while maintaining a consistent extraction interface for each image.
`image_paths = [...]`
Batch processing also improves efficiency by enabling concurrent execution where supported by the underlying runtime.

---

6. Numerical Reasoning via External Tools

After structured data extraction, all numerical reasoning is performed using deterministic Python functions(`def calculate_original_price, def handle_extraction_and_calc`) rather than the language model itself.

- For **Query 1**, the system computes the total amount paid by summing the `subtotal` values across all extracted receipts.
- For **Query 2**, the system reconstructs the original total by adding the extracted discount amounts back to the paid subtotals.

This separation ensures that the language model is not responsible for arithmetic operations, significantly reducing the risk of hallucinated or inconsistent numerical results.

---

7. Irrelevant Query Handling

When the router identifies an irrelevant query, the system bypasses all extraction and calculation steps and returns a predefined rejection response. (`def handle_irrelevant`)

This behavior satisfies the assignment requirement that the model must be able to reject unsupported queries, while maintaining clear and predictable system behavior.

---

8. Summary

Overall, the implementation demonstrates a clear separation of concerns between intent recognition, perceptual extraction, and numerical reasoning.
This modular execution logic aligns closely with the agentic AI principles discussed in the course and contributes to the system's robustness, interpretability, and reliability.