

LLM Initialization

Initializes Gemini 2.5 Flash model (Vertex AI integration) with zero temperature.

Query 1: Total Spent Calculation

This section implements the core logic for calculating total post-discount spending, that is sum of "total paid" across all receipts.

Prompt Design: Pure Numeric Extraction Receipt Processing Logic

```
extract_prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a receipt amount extractor. ONLY output the TOTAL PAID amount (after discounts) as a",
    ("human", [
        {"type": "text", "text": "Extract the TOTAL PAID amount from this supermarket receipt."},
        {"type": "image_url", "image_url": {"url": "{image_url}"}}
    ])
])
```

Receipt Processing Logic

Chain Construction: Combine the prompt, LLM, and StrOutputParser into a single executable chain to retrieve the model's output as a string. Add up all receipts result using `total_spent += amount`.

```
extract_chain = extract_prompt | llm | StrOutputParser()
result = extract_chain.invoke({"image_url": img_url}).strip()

# Validate and sum valid amounts
amount = float(result.replace(",", ""))
total_spent += amount
valid_receipts += 1
```

Query 2: Original Price Calculation

The core logic is **calculating total pre-discount price**: $\text{Original Price} = \text{Actual Payment} + \text{Discount}$. Use structured JSON Extraction for prompt design. Use To Instruct the Gemini model to extract **two critical numeric fields** (actual payment and total discount) from receipt images and return them in a structured JSON format.

```

extract_prompt = ChatPromptTemplate.from_messages([
    ("system", """You are a professional receipt OCR extractor.
Output ONLY valid JSON with these fields:
1. total_paid: float (actual payment amount, numbers only, exclude text/units)
2. discount_total: float (absolute sum of all consumption discounts, pre-calculated)
3. Ignore: balance/余额/餘額/points/积分
JSON Example: {"total_paid": 394.70, "discount_total": 91.4}"""),
    ("human", [
        {"type": "text", "text": "Extract payment info from this receipt (only JSON)"},
        {"type": "image_url", "image_url": {"url": "{image_url}"}}
    ])
])

```

Define a request classifier

Route user queries to the correct handler (query1/query2/irrelevant) based on the intent of the supermarket receipt-related request.

```

system_prompt = """You are a supermarket receipt query router.
Decide which handler should process the user's request based on the two valid query types:
- Output 'query1' if the request asks for the TOTAL AMOUNT SPENT on receipts.
- Output 'query2' if the request asks for the ORIGINAL TOTAL without discounts.
- Output 'irrelevant' if the request is not related to receipt totals.
ONLY output one word: 'query1', 'query2', or 'irrelevant'."""

user_prompt = "{request}"

```

System Prompt: Roles the model as a "receipt query router" with clear classification rules for three outcomes.

User Prompt: A dynamic placeholder ({request}) that injects the raw user query into the prompt for classification.

Create prompt template for query classification, transforms the static prompt rules into a runnable logic for intent recognition.

```

coordinator_router_prompt = ChatPromptTemplate.from_messages([
    ("system", system_prompt),
    ("user", user_prompt)
])

```

Build classification chain to classify request and then parse output

Uses LangChain's LCEL (LangChain Expression Language) to chain three

components into a single executable pipeline:

```
coordinator_router_chain = coordinator_router_prompt | llm | StrOutputParser()
```

Retrieve and standardize the list of receipt image files to ensure consistent processing of receipts in the system.

Remember to run this again when insert new receipt photos.

```
image_paths = glob.glob("*.jpg")  
image_paths.sort()
```

Finally we can run the code to achieve our goals successfully.