FTEC5660: Homework 1

Receipt Data Extraction and Query Classification

- **Name**: Lei Qingyuan
- **ID**: 1155243913
- **Major**: Artificial Intelligence

1. Prompt Engineering (Core Design)

Prompts are critical to achieving precise data extraction and intent classification. Below are the design logic and constraint rules for the two core prompts passed to the API:

1.1 Receipt Data Extraction Prompt (`prompt_step1`)

Design Objective

Force the model to output structured, parsable JSON data from receipt images (including item details and total paid amount) with unified numeric formatting and no redundant information.

Core Constraint Rules (System Instruction in `prompt_step1` passed to API)

The complete system prompt sent to the API is as follows:

```
prompt_step1 = ChatPromptTemplate.from_messages([
        ("system", """You are a meticulous and accurate receipt price extraction expert. Output ONLY a valid JSON string (no other text, no code blocks, no markdown) and strictly comply with ALL the following rules:
1. Image Analysis & Item Extraction:
    - Extract EVERY visible item/line item from the single receipt image (no omissions, even discount/promotion lines)
    - For each item, populate 4 mandatory fields (exact field names, case-sensitive):
        - "item_name": Exact name of the item (string; keep original spelling/case, e.g., "Milk", "5% OFF", "Tax"); fill "Unknown" if unrecognizable
        - "original_price": Undiscounted price of the item (numeric float; directly from the receipt's price column, BEFORE discount; 0 if it's a discount/promotion line with no base price)
        - "discount_amount": Exact amount reduced by discount (numeric float; convert negative discount values on receipt to positive, e.g., "-5.00" → 5.00; 0 if no discount for the item)
        - "total_paid_amount" (global field): Final total amount paid (numeric float; extract from "Total", "Amount Paid", "Grand Total" at the bottom of the receipt; 0 if unrecognizable)
2. JSON Structure (strictly follow, no extra fields/typos):
    {{
      "image_name": "EXACT filename of the receipt image (e.g., receipt1.jpg)",
      "item_details": [list of all extracted items (each with item_name/original_price/discount_amount)],
      "total_paid_amount": numeric float (final paid total)
    }}
3. Numeric Rules:
    - All amounts are pure numbers (float/int) with NO units (e.g., $, €), NO commas (e.g., 1,000.50 → 1000.50), NO currency symbols
    - Round all numeric values to 2 decimal places (standard currency format)
    - Fill 0 only if the field is completely unrecognizable (never leave fields empty/null)
4. Output Constraints:
    - Return ONLY the pure JSON string (no ```json, no comments, no explanations, no extra text)
    - Ensure JSON is valid (can be parsed by standard JSON parsers)
    - Match field names EXACTLY as specified (case-sensitive: e.g., "item_name" not "ItemName" or "itemname")"""),
        ("human", [
            {"type": "text", "text": "Extract all item details and total paid amount from the receipt image. Return ONLY a valid JSON string following the rules above (no code blocks, no extra text)."},
            {"type": "image_url", "image_url": {"url": "{image_url}"}}
        ])
])
```

Input Combination

Adopts a multimodal input method of "text instruction + image URL": the text part reinforces extraction requirements, and the image URL is a Base64-encoded Data URL (compatible with Gemini's image input format).

1.2 Query Classification Prompt (`prompt_classify`)

Design Objective

Accurately identify user query intent, classify it into three categories, and output structured JSON results for downstream code invocation.

Core Constraint Rules (System Instruction in `prompt_classify` passed to API)

The complete system prompt sent to the API is as follows:

```
prompt_classify = ChatPromptTemplate.from_messages([
    ("system", """You are a professional query classification assistant. Classify user queries about supermarket receipts strictly following these rules, and output only JSON string (no other text, no code block tags):

### Classification Rules
Category 1 (Query total paid amount):
- Core intent: Ask for the total actual paid amount of all supermarket receipts
- Typical examples:
  - "How much money did I spend in total for these bills?"
  - "What is the total amount I actually paid for all these receipts?"
  - "Sum of total paid amount for all receipts?"

Category 2 (Query total original amount without discount):
- Core intent: Ask for the total amount to pay if there were no discounts for all supermarket receipts
- Typical examples:
  - "How much would I have had to pay without the discount?"
  - "What is the total original amount (no discount) for all receipts?"
  - "Total price without considering discounts for all bills?"

Category 0 (Other irrelevant questions):
- All questions not matching Category 1/2, including but not limited to:
  - Asking amount of a single receipt or price of a single item
  - Asking discount amount or discount rate
  - Irrelevant small talk, questions in other fields
  - Unrecognizable vague questions

### Output Requirements
1. Output format: Pure JSON string with 3 fields:
   - "query_text": Input user query text (preserved as-is)
   - "category": Classification result (string, options: "Query total paid amount", "Query total original amount (no discount)", "Other irrelevant questions")
   - "category_code": Classification code (integer, options: 1, 2, 0)
2. Currency unit and language (Chinese/English) do not affect classification, only focus on core intent
3. Return only JSON string, no extra explanations, comments, or code block tags
4. Unrecognizable questions are classified as Category 0 uniformly
"""),
    ("human", "Classify the following user query and return only valid pure JSON string:\n{user_query}")
])
```

2. Core Code Logic and Workflow

2.1 Receipt Data Extraction Workflow

Step 1: Image Preprocessing (Integrated into Context)

Convert local receipt images to Base64-encoded Data URLs (compatible with Gemini's image input requirements) via the `get_image_data_url` function, and filter invalid image paths to avoid model invocation failures:

```python
for img_path in image_paths:
    try:
        data_url = get_image_data_url(img_path)
        image_data_urls.append(data_url)
        valid_image_paths.append(img_path)
    except Exception as e:
        print(f"❌ Failed to convert {os.path.basename(img_path)}, skipped: {str(e)}")
        continue
```

Step 2: Model Invocation and Response Cleaning

- Invoke `chain_step1.invoke({"image_url": data_url})` to pass the image Data URL to the model and retrieve the response;
- Clean the response text via the `clean_model_response` function: remove ```json tags, extra spaces/newlines, and extract pure JSON strings to resolve irregular model output formatting:

```
def clean_model_response(response_text):
    cleaned = re.sub(r'^```json\s*|\s*```$', '', response_text.strip(),
flags=re.MULTILINE)
    cleaned = re.sub(r'\s+', ' ', cleaned).strip()
    return cleaned
```

Step 3: Total Original Price (No Discount) Calculation

The core function `calculate_original_price` implements the logic: **Total Original Price = Total Paid Amount + Sum of All Discount Amounts**, ensuring alignment with business scenarios:

```
def calculate_original_price(step1_result):
    discount_sum = 0.0
    for item in step1_result.get("item_details", []):
        discount = float(item.get("discount_amount", 0))
        if discount > 0:
            discount_sum += discount
    paid_total = float(step1_result.get("total_paid_amount", 0))
    final_original = paid_total + discount_sum
    return round(final_original, 2)
```

```
========== Processing image 1: receipt1.jpg ==========
🔹 Extracting item price details for receipt1.jpg...
✅ API call succeeded! Extracted 22 items

📜 receipt1.jpg item bill details:
  1. 韭菜豬肉雲吞20粒裝 → Original price: 24.9 | Discount: 0.0
  2. 包裝變形 → Original price: 0.0 | Discount: 12.4
  3. 韭菜豬肉雲吞20粒裝 → Original price: 24.9 | Discount: 0.0
  4. 包裝變形 → Original price: 0.0 | Discount: 12.4
  5. 韭菜豬肉雲吞20粒裝 → Original price: 24.9 | Discount: 0.0
  6. 包裝變形 → Original price: 0.0 | Discount: 12.4
  7. IF100% COCONUT WATER → Original price: 57.8 | Discount: 0.0
  8. Buy 2 Save $12.8 → Original price: 0.0 | Discount: 12.8
  9. 玉芒 → Original price: 29.7 | Discount: 0.0
  10. Buy 3 Save $10.8 → Original price: 0.0 | Discount: 10.8
  11. Fresh綜合莓汁 → Original price: 35.8 | Discount: 0.0
  12. Buy 2 Save $3.9 → Original price: 0.0 | Discount: 3.9
  13. 雀巢脫脂高鈣牛奶飲品 → Original price: 21.9 | Discount: 0.0
  14. 蔬菜先生包裝蕃茄(中 → Original price: 9.9 | Discount: 0.0
  15. 蔬菜先生翠玉瓜1磅( → Original price: 7.9 | Discount: 0.0
  16. 咸蛋蒸肉餅 → Original price: 36.9 | Discount: 0.0
  17. 金御膳龍口粉絲 → Original price: 23.8 | Discount: 0.0
  18. (業務用)急凍蟶子肉 → Original price: 130.0 | Discount: 0.0
  19. 白柳姬菇 → Original price: 14.9 | Discount: 0.0
  20. 蒜汁雞扒 → Original price: 36.9 | Discount: 0.0
  21. 5% OFF (CU) → Original price: 0.0 | Discount: 20.78
  22. ROUNDING → Original price: 0.0 | Discount: 0.02

💰 Calculation result → Total paid: 394.7 | Original total (no discount): 480.2
```

Step 4: Result Aggregation

Iterate through extraction results of all valid receipts, accumulate the total paid amount (`total_paid_all`) and total original amount (`total_original_all`) for all receipts, and output details and aggregated results for subsequent query invocation.

```
==================== Final Results Summary for All Images ====================
📄 receipt1.jpg → Total paid: 394.7 | Original total (no discount): 480.2
📄 receipt2.jpg → Total paid: 316.1 | Original total (no discount): 392.2
📄 receipt3.jpg → Total paid: 140.8 | Original total (no discount): 160.1
📄 receipt4.jpg → Total paid: 514.0 | Original total (no discount): 590.8
📄 receipt5.jpg → Total paid: 102.3 | Original total (no discount): 107.7
📄 receipt6.jpg → Total paid: 190.8 | Original total (no discount): 221.2
📄 receipt7.jpg → Total paid: 315.6 | Original total (no discount): 396.0

📊 All receipts summary:
✅ Total paid amount for all receipts: 1974.3
✅ Total original amount (no discount) for all receipts: 2348.2
```

2.2 User Query Classification Workflow

Step 1: Classification Model Invocation

Encapsulate classification logic via the `classify_user_query` function, invoke `chain_classify.invoke({"user_query": user_query})` to retrieve model responses, and parse responses via `parse_classification_result`:

- Clean response text (same logic as data extraction cleaning);
- Return "Other irrelevant questions" (Category 0) by default if JSON parsing fails, ensuring program robustness.

Step 2: Classification Result Application

Return corresponding results based on classification codes (`category_code`) via the `get_answer_by_category` function:

- Code 1: Return total paid amount;
- Code 2: Return total original amount (no discount);
- Code 0: Return irrelevant question prompt.

3. Testing and Validation

3.1 Receipt Data Extraction Validation

Validation Result

The `total_paid_all` (total paid amount) and `total_original_all` (total original amount without discount) for all receipts passed validation, with errors complying with requirements.

Run the following code block to evaluate query 1:

> How much money did I spend in total for these bills?

```
[37]
✓ 0 秒
query_1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this
query1_answer = total_paid_all
test_query(query1_answer, query_1_costs)
```

Run the following code block to evaluate query 2:

> How much would I have had to pay without the discount?

```
[38]
✓ 0 秒
query_2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this
query2_answer = total_original_all
test_query(query2_answer, query_2_costs)
```

```
[39]
✓ 0 秒
sum([480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00])
```

```
2348.2
```

3.2 Query Classification Testing

Test Case Design

8 test queries covering three intent categories (including Chinese/English and different expressions) to ensure full coverage of classification logic:

```
...
================== Test: User Query Classification ==================

📝 Test Query 1: How much money did I spend in total for these bills?
   📊 Classification Result → Category: Query total paid amount | Code: 1

📝 Test Query 2: How much would I have had to pay without the discount?
   📊 Classification Result → Category: Query total original amount (no discount) | Code: 2

📝 Test Query 3: What is the total amount I actually paid?
   📊 Classification Result → Category: Query total paid amount | Code: 1

📝 Test Query 4: What is the total price without discounts?
   📊 Classification Result → Category: Query total original amount (no discount) | Code: 2

📝 Test Query 5: How much did I spend on the first receipt?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

📝 Test Query 6: What is the original price of apples?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

📝 Test Query 7: How is the weather today?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

📝 Test Query 8: What is the total discount amount for these receipts?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

================== Example: Return Answer by Classification ==================

🔍 User Query: What is the total amount I actually paid?
📋 Classification Result: Query total paid amount
💡 Answer: The total paid amount for all receipts is: 1974.3 HKD
```

Validation Result

All test cases achieved 100% classification accuracy. The model accurately identified core intents, and all irrelevant questions were correctly categorized.