

# Homework 1

## 1. Environment Setup

Includes the package installing and API key setup.

## 2. Data Preparation & Pipeline Setup

Firstly, we should download receipts.zip.

Then, we need to encode these images. We convert the image into a string format that the API can understand. as the API interface of LLM usually can only receive text (in JSON format), and cannot directly receive binary image files.

### code 1

```
def image_to_base64(img_path):  
    with open(img_path, "rb") as img_file:  
        return  
base64.b64encode(img_file.read()).decode('utf-8')
```

What will the codes do?

[Images] -- (rb, Read Binary) → [Binary data, type:bytes] -- (Base64, encoding)→  
[Secure binary characters, type: bytes] -- (utf-8, decoding) → [Python string, type: str]

## 3. Intelligent Document Extraction (1<sup>st</sup> chain)

```
locate_chain = prompt_locate | llm | JsonOutputParser()
```

We build our first chain in order to locate and extract the key information by OCR. The processed images will be used as input for the LLM.

For system prompt, we should give some specification to get a certain output format(eg. {items:[ ], all\_printed\_totals: [ ]}), which will make it more convenient for the subsequent processing of data.

**Note:** We use a **for loop** to process each image one by one, instead of having LLM process all the images at once. As LLM often encounters the "Attention Decay" when dealing with long contexts or multi-modal tasks.(When I initially input seven pictures

at a time, I discovered the problem of incorrect digit extraction.)

#### `JsonOutputParser()`

Function: Extract the content from the AI Message object (json text) returned by LLM and convert it into a Python dictionary or list.

Reason: We need to perform some calculations using Python instead of relying on LLM for the computations, as LLM is not very good at doing calculations.

#### **4. Analyze the queries and design responses (2<sup>nd</sup> chain)**

Firstly, we should design each response for three possible query types.

*For Query 1: How much money did I spend in total for these bills?*

- A function called "totalcost" is defined to return the total sum of all the subtotals of the receipts.

*For Query 2: How much would I have had to pay without the discount?*

- A function called "ori" is defined to return the total sum of original prices of all the items in all receipts.

*Query 3: Other irrelevant queries. Reject.*

- Return "Sorry, the question is irrelevant"

However, the user's query expression may vary to some extent, so we need to understand the true purpose of the query. In order to analyze the user's query we should build the second chain.

```
processing_router_chain = processing_router_prompt | llm | StrOutputParser()
```

We use LLM to analyze which category the user's query belongs to. And `RunnableBranch` is used to select which branch to run based on the routing by LLM.

Therefore, we further refined the second chain.

```
processing_router_chain | delegation_branch | (lambda x: x["output"])
```

#### **5. Evaluation Code**

The output for query 1 is 1974.3.

The output for query 2 is 2348.2.

We run the test\_query code blocks, and the blocks does not return any error.

Let's test our agent !

[13]  
✓ 0  
秒

```
def test_query(answer, ground_truth_costs):  
    # Convert string to float if necessary  
    if isinstance(answer, str):  
        answer = float(answer)  
  
    # Calculate the ground truth sum once for clarity  
    expected_total = sum(ground_truth_costs)  
  
    # Check if the answer is within +/- $2 of the expected total  
    assert abs(answer - expected_total) <= 2
```

[14]  
✓ 0  
秒

```
query1 = "How much money did I spend in total for these bills?"  
query2 = "How much would I have had to pay without the discount?"
```

[15]  
✓ 0  
秒

```
query1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this  
query1_answer = coordinator_agent.invoke({"query": query1})  
test_query(query1_answer, query1_costs)
```

[16]  
✓ 0  
秒

```
query2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this  
query2_answer = coordinator_agent.invoke({"query": query2})  
test_query(query2_answer, query2_costs)
```