Name: Zhang Xun Student ID: 1155245409

Homework 1

1. Problem description

We need to build a receipt assistant model. The model needs to fulfill the following functions. *1) Identify receipts, convert the images into text and extract the corresponding information. 2) Calculate. Sum up the amounts on the receipts. 3) Analyze the user's questions and provide appropriate responses.*

2. Implementation Details

2.1 Environment Setup

Includes the package installation and API key setup.

2.2 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]

2.3 Intelligent Document Extraction (1st chain)

locate_chain = prompt_locate | llm |JsonOutputParser()

We built 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 provide 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.

code 2

```
for i in image_paths:
  filename = os.path.basename(i)
  key_name = os.path.splitext(filename)[0]
  image_data_url = get_image_data_url(i)
  response[key_name] = locate_chain.invoke({
    "question": "What is in this picture?",
    "image_url": image_data_url,
    })
```

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.

2.4 Analyze the queries and design responses (2nd 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"]

2.5 Reusability of the model

code 3

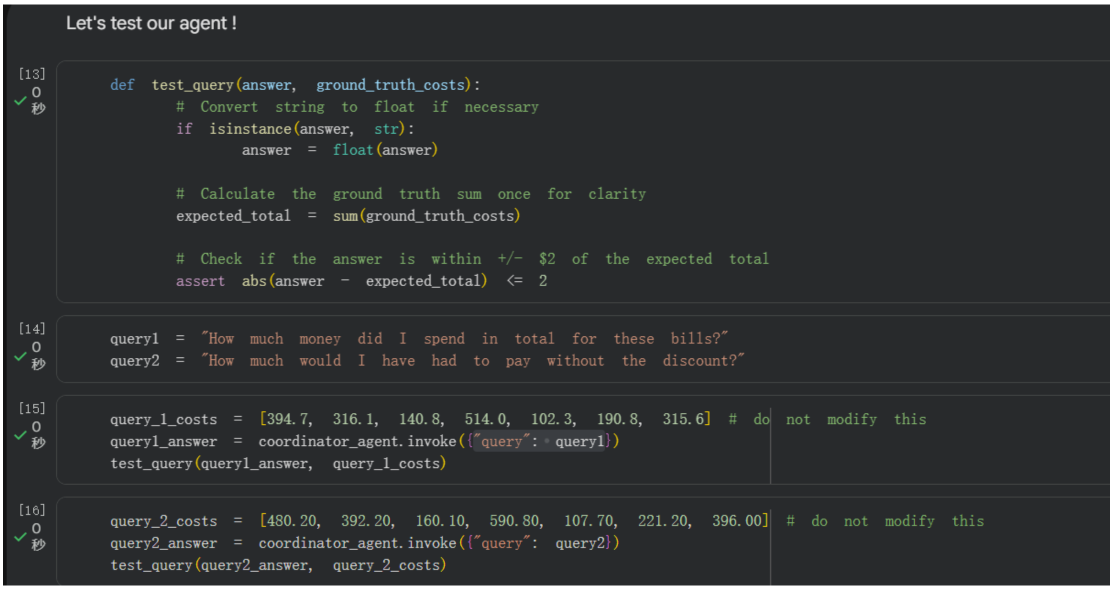
```
image_paths = [
    "/content/receipt1.jpg", "/content/receipt2.jpg", 
"/content/receipt3.jpg",
    "/content/receipt4.jpg", "/content/receipt5.jpg", 
"/content/receipt6.jpg",
    "/content/receipt7.jpg"
]

num=len(image_paths)
```

In order to ensure that the model can operate properly and conveniently, when uploading other receipts, all images will be placed in a list as input. Additionally, a length function will be used to determine the number of elements in the list, which is then used to sum up the amounts of multiple receipts.

2.6 Evaluation Code

The output for query 1 is **1974.3**.

The output for query 2 is **2348.2**.



Let's test our agent !

```
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
```

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

```
query_1_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, query_1_costs)
```

```
query_2_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, query_2_costs)
```

The evaluation code measures the difference between the actual value and the answer given by the model.

Finally, I run the test_query code blocks, and the blocks do not return any error. This means the chain design is correct.