SEEM5560 - HomeWork1 - Report

Student ID: 1155254873

Homework Content：Input several images of supermarket bills, plus one user query. There are two possible queries that users are interested to know: (Query 1: How much money did I spend in total for these bills? ; Query 2: How much would I have had to pay without the discount?)
And the model should have the capacity to reject irrelevant queries.

Solution:

1. **Load environment variables.** I use .env file to store my api key and base url. And I use dotenv and os modules to load the environment variables.

2. **Download and unzip the receipts images.**

Here I just use the methods which offered by professor/TA. I find that "!unzip receipts.zip" is not useful in my computer with windows system. So I use other method that can help for windows system. The receipts will be stored in the folder named 'receipts'.

```python
# This code can work for windows, unzip receipts.zip
import zipfile
import os

Trae: 解释代码 | 注释代码 | ×
def unzip_file(zip_path, extract_to):
    # 确保目标文件夹存在
    if not os.path.exists(extract_to):
        os.makedirs(extract_to)

    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
    print(f"解压完成：{extract_to}")

# 使用示例
unzip_file('receipts.zip', 'receipts')
```

3. **Transform images into format of base64, and generate a list with base64 urls.**

I use the methods to transform the receipt image into base64 and save in a list named 'receipt_urls'.

```python
receipt_urls = []
for i in range(1,8):
    filepath = f"receipts/receipt{i}.jpg"
    receipt_url = get_image_data_url(filepath)
    receipt_urls.append(receipt_url)
```

✓ 0.0s

```
receipt_urls Ctrl+U 添加上下文到对话
✓ 0.0s
['
 '
 '
 '
 '
 '
 '
```

4. **Create llm using model of gemini-2.5-flash**

The homework1 need to use a specific model called 'gemini-2.5-flash'. I have tried some methods to get a gemini/vertex api, but still can't get one. So I use a third-party API, leveraging

OpenAI's compatible interface, and successfully deployed the gemini-2.5-flash model.

```python
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough, Runnable, RunnableBranch, RunnableLambda

# Since base_url is not supported in the current version of langchain_google_genai,
# I use the Third party - AIHubMix API, and use OpenAI Compatible Interface to invoke Gemini 2.5 Flash
# So the model I use is gemini-2.5-flash actually.
llm = ChatOpenAI(
    model="gemini-2.5-flash",
    api_key=os.getenv("AIHUBMIX_API_KEY"),
    base_url=os.getenv("AIHUBMIX_API_URL"),
)
```

✓ 0.0s

I use the class of ChatOpenAI, but actually I just use this compatible interface and still use the model of gemini-2.5-flash.

5. Design the prompt

(1) Router prompt design

I design a system prompt which allow llm to decide which kind of query user asked. This system prompt asks llm to output only a single word: 'query1', 'query2', or 'unclear', then I can use this specific word in next processes.

The router_chain will just out a single word, if the query is like “How much money did I spend in total for these bills?”, then output “query1”, or if the query is like “How much would I have had to pay without the discount?”, then output “query2”. When user ask a query but not relevant to the bill price, then llm can output “unclear” for the next processes.

```python
# Router design
system_prompt = """You are a classification expert. Strictly categorize requests:

- **Output 'query1' ONLY if**:
  Explicitly mentions "discount", "after discount", "final payment", "actual payment" or payment methods (OCTOPUS/VISA/支付宝)
  Example: "How much money did I spend in total for these bills?" / "What's the total price after the discount?" / "Actual Payment Amount?"

- **Output 'query2' ONLY if**:
  Explicitly mentions "without discount", "no discount", "raw price", "original price" or "no special offer"
  Example: "How much would I have had to pay without the discount?" / "What was the price before the discount?" / "What is the original price?"

- **Output 'unclear' for**:
  - Unrelated questions (weather/time/general knowledge)
  - Ambiguous/missing context
  - Any request NOT explicitly about bill amounts

NEVER guess. Output EXACTLY one word: 'query1', 'query2', or 'unclear'."""

router_prompt = ChatPromptTemplate.from_messages([
    ("system", system_prompt),
    ("human", [
        {"type": "text", "text": "{request}"},
        # Router don't need receipt image
        # {"type": "image_url", "image_url": {"url": "{receipt_url}"}},
    ]),
])

router_chain = router_prompt | llm | StrOutputParser()
```

✓ 0.0s

(2) Design three prompts/chains for conditions of query1, query2 and irrelevant

(i) The query1 ‘s system prompt asks the llm to extract the actual final price of bill. I find that the answer would always shown behind the “OCTOPUS” or “VISA”. So the llm just need to extract that number without signal $. The image shows part of the system prompt.

```
query1_system_prompt = """You are a bill data extractor. Follow these steps:
1. Locate the ACTUAL PAYMENT amount (after discount)
2. Focus on sections labeled: "OCTOPUS", "VISA", "支付宝", "实付金额", "最终支付"
3. IGNORE all other numbers (prices, subtotals, taxes)
4. Output ONLY the numeric value (e.g. "352.00")
```

```
query1_prompt = ChatPromptTemplate.from_messages([
    ("system", query1_system_prompt),
    ("human", [
        # Dont need text here, only image urls are needed.
        # {"type": "text", "text": "{request}"},
        {"type": "image_url", "image_url": {"url": "{receipt_url}"}},
    ]),
])
```

```
query1_single_chain:Runnable = query1_prompt | llm | StrOutputParser() | (lambda x: round(float(x), 2))
```

To fulfil the function about putting in several urls, I create a function that can multiple call the query1_single_chain, and let it be runnable. Here I finish the query1_chain.

```
def process_query1(receipt_urls) -> float:
    # Map over each URL
    results = []
    for url in receipt_urls:
        result = query1_single_chain.invoke({"receipt_url": url})
        results.append(result)
    return results

query1_chain = RunnableLambda(lambda x: process_query1(x["input"]["receipt_urls"]))
```

(ii) The query2 need agent to calculate the price without discount. I write the system prompt and ask the llm to indentify every items' positive price, and ignore all negative price. I order llm just output a list of numeric positive price. Most of times it works, but sometimes llm will talking nonsense, so in system prompt I repeat the output format demand triple times.

```
query2_system_prompt = """You are a bill data extractor. Follow these steps:
1. Extract each individual item price before SUBTOTAL or "小计" in the image.
2. Extract all positive item prices. Every item price with a sign "+$" is a positive item price. (e.g. "$29.70")
3. IGNORE amounts with negative sign (e.g. "-$9.70")
4. YOU MUST NOT LEAVE OUT ANY POSITIVE ITEM PRICE!
5. LIST ALL POSITIVE ITEM PRICE WITHOUT POSITIVE SIGN (e.g. "29.70" -> 29.70), the list is like [29.70, 30.00, 15.00, 15.00].
6. ONLY output the list of positive item prices WITHOUT any other text.
7. ONLY output the list of positive item prices WITHOUT any other text.
8. ONLY output the list of positive item prices WITHOUT any other text.
```

```
query2_prompt = ChatPromptTemplate.from_messages([
    ("system", query2_system_prompt),
    ("human", [
        # {"type": "text", "text": "{request}"},
        {"type": "image_url", "image_url": {"url": "{receipt_url}"}},
    ]),
])
```

To get the sum of the list, I create a function that can transfer the JSON format to python list, and change each item into float number.

```
import json
Trae: 解释代码 | 注释代码 | ×
def calculate_sum(json_str):
    try:
        parsed_result = json.loads(json_str)
        float_list = [round(float(item), 2) for item in parsed_result]
        return round(sum(float_list), 2)
    except:
        return "query2_error"

query2_single_chain:Runnable = query2_prompt | llm | StrOutputParser() | RunnableLambda(calculate_sum)
```

To fulfil the function about putting in several urls, I also create a function that can multiple call the query2_single_chain, and let it be runnable. Here I finish the query2_chain.

```python
def process_query2(receipt_urls) -> float:
    results = []
    for url in receipt_urls:
        result = query2_single_chain.invoke({"receipt_url": url})
        results.append(result)
    return results

query2_chain = RunnableLambda(lambda x: process_query2(x["input"]["receipt_urls"]))
```

(iii) As for users' irrelevant query, I use unclear_chain to process. The only thing llm need to do is replying the user that "Your request is unclear, I can't help you with that."

```python
unclear_prompt = ChatPromptTemplate.from_messages([
    ("system", "Please tell user that the request is unclear directly, and don't try to guess, JUST SAY 'Your request is unclear, I can't help you with that.'"),
    ("human", "User request: {request}")
])
unclear_chain:Runnable = unclear_prompt | llm | StrOutputParser()
```

So now I have three branches which are query1_chain, query2_chain and unclear_chain.

6. Delegate the branch

After router decides the type of users' query, one branch will be activated.

```python
delegation_branch = RunnableBranch(
    (lambda x: x["decision"].strip() == "query1", query1_chain),
    (lambda x: x["decision"].strip() == "query2", query2_chain),
    unclear_chain,
)
```

7. Define the receipt_agent

```python
receipt_agent = {
    "decision": router_chain,
    "input": RunnablePassthrough(),
} | RunnablePassthrough.assign(
    receipt_url=lambda x: x["input"]["receipt_urls"],
    request=lambda x: x["input"]["request"]
    ) | delegation_branch
```

The initial input are a "request" and a "receipt_urls". The router will output a specific word in "decision", for example "decision": "query1". The "input" will save all the initial input items, for example "input" : {"request": "...", "receipt_urls":[url1, url2, ...]}. And then, since I need the contents of "input", so I use RunnablePassthrough.assign() and unpack the "input" (otherwise the llm can't see the "receipt_urls" and "request"). Finally, I use the delegation_branch to distribute the chain for each conditions.

Test examples:

The receipt_urls is a list of base64 urls.

```python
    # review the receipt_urls
    receipt_urls
```

```
['
 '
 '
 '
 '
 '
 '
```

Test query 1, put the results in "result1".

```
input_dict = {
    "request": "How much money did I pay in total?",
    "receipt_urls": receipt_urls,
}
result1 = receipt_agent.invoke(input_dict)
result1
```

```
[394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6]
```

Test query 2, put the results in “result2”.

```
input_dict = {
    "request": "How much would I have had to pay without the discount?",
    "receipt_urls": receipt_urls,
}
result2 = receipt_agent.invoke(input_dict)
result2
```

```
[480.2, 392.2, 160.1, 590.8, 107.7, 221.2, 396.0]
```

Test query 3.

```
input_dict = {
    "request": "How's the weather today?",
    "receipt_urls": [receipt_urls[0], receipt_urls[1], receipt_urls[2]],
}
response = receipt_agent.invoke(input_dict)
print(response)
```

```
Your request is unclear, I can't help you with that.
```

Put the result1 and result2 in method of test_query.

```
def test_query(answer, ground_truth_costs):
    # Convert string to float if necessary
    if isinstance(answer, str):
        answer = float(answer)
    answer = sum(answer)
    expected_total = sum(ground_truth_costs)
    assert abs(answer - expected_total) <= 2
    print(f"Test passed: Answer is within $2 of the expected total.")
```

```
query_1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this
query1_answer = result1
test_query(query1_answer, query_1_costs)
```

```
Test passed: Answer is within $2 of the expected total.
```

```
query_2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this
query2_answer = result2
test_query(query2_answer, query_2_costs)
```

```
Test passed: Answer is within $2 of the expected total.
```