

Methodology

Our approach employs the ReACT (Reasoning and Acting) framework, a dynamic and iterative process that intertwines cognitive reasoning steps with tool-driven actions to manage personal financial transactions effectively. By using ReACT, we aim to ensure that user queries—ranging from expense management and budgeting to financial inquiries—are thoroughly understood, processed, and responded to with both precision and adaptability.

ReACT Framework Overview

The ReACT framework is built around two primary components: a Reasoning (RE) module and an Acting (ACT) module. These components form a continuous loop where reasoning informs action and the outcomes of actions, in turn, refine further reasoning.

1. **Reasoning (RE):**

Utilizing a large language model (ChatGPT), the reasoning step interprets the user's input, identifies the nature of the request, and determines the necessary financial tasks. The system pinpoints whether the query involves expense tracking, budgeting advice, or other financial services. By engaging in flexible, context-aware reasoning, the framework ensures that complex financial inquiries are broken down into actionable steps.

2. **Acting (ACT):**

Informed by the reasoning component, the acting stage deploys a suite of specialized tools and agents. These tools can range from internal modules built specifically for expense management to external integrations that offer real-time data fetching and advanced financial calculations. Acting tools perform the required operations—such as extracting details from receipts, creating or updating expense entries, categorizing transactions, or parsing time-based financial data—and return the results to the reasoning component.

Integration with Personalized and External Tools

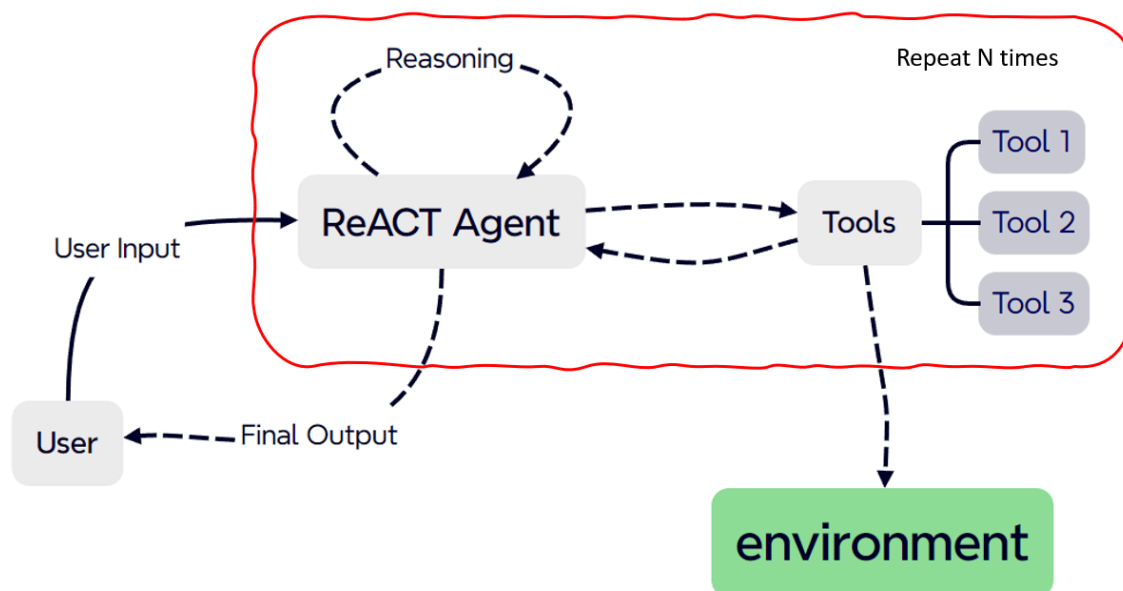
To enhance the system's versatility, we incorporate both custom-built and externally sourced tools:

- **LangChain Tools:** External services like Google-serper supply current financial data, while llm-math supports complex financial computations.
- **Custom Tools:** Developed in-house for tailored financial tasks, these include:
 - *process_receipt*: Extracts and structures transaction details from receipts.
 - *create_or_update_expense_tool*: Manages expense records, creating new entries or updating existing ones, and stores them to the database
 - *category_merchant_tool*: Automatically categorizes expenses based on merchants and transaction types, and stores them to the database
 - *parse_relative_time* and *get_related_time*: Determine relevant timeframes (e.g., last month, this week) to anchor financial queries to specific periods.

Operational Cycle of ReACT

1. **Input Evaluation:**
The reasoning module assesses the user's query, identifying what financial insight or action is required.
2. **Tool Selection and Action Execution:**
Guided by the reasoning outcomes, the system activates the corresponding acting tools. These agents perform their assigned tasks—whether it's pulling external data, recalculating budgets, or reorganizing expense categories.
3. **Observation and Iteration:**
Results from the acting agents are fed back into the reasoning module. This iterative loop allows for ongoing refinement, ensuring that partial insights are clarified and that the final output is both comprehensive and contextually accurate.
4. **Final Thought Synthesis:**
After one or more iterations, the reasoning module integrates all learned insights and observed outcomes, producing a coherent and informed response.

Here is the Operational cycle flowchart:



Also, here is an easy example of how react agent work with gpt 4o as reasoning with 2 tools of google_serper and Calculator with input: What is the new building of Georgetown University? What is the name of the new building? And what is the age of the university raised to the 0.23 power?

google_serper - A low-cost Google Search API. Useful for when you need to answer questions about current events. Input should be a search query.

Calculator(*args: Any, callbacks:
Union[list[langchain_core.callbacks.base.BaseCallbackHandler],

langchain_core.callbacks.base.BaseCallbackManager, NoneType] = None, tags: Optional[List[str]] = None, metadata: Optional[Dict[str, Any]] = None, **kwargs: Any) -> Any - Useful for when you need to answer questions about math.

Thinking and combining thoughts processes:

To answer your questions, I need to find the information about the new building at Georgetown University and calculate the age of the university raised to the 0.23 power. I will start by searching for information about the new building.

Action: google_serper

Action Input: "new building at Georgetown University 2023"

Observation: *Redevelopment of the former Henle Village (May 2023 – August 2025) Event: Construction of a new residential hall featuring 780 beds, expanded health and wellness spaces, fully-equipped apartments with kitchens, increased space for CAPS, amenity areas, and an exterior courtyard.*

Thought: *The new building at Georgetown University is a residential hall being constructed as part of the redevelopment of the former Henle Village. Now, I need to find the age of Georgetown University.*

Action: google_serper

Action Input: "Georgetown University founding year"

Observation: *Georgetown University: Private university in Washington, D.C.. Georgetown University is a private Jesuit research university in the Georgetown neighborhood of Washington, D.C., United States. Founded by Bishop John Carroll in 1789, it is the oldest Catholic institution of higher education in the United States and the nation's first federally chartered university... [309 words]*

Thought: *Georgetown University was founded in 1789. Now, I need to calculate the age of the university raised to the 0.23 power. The current year is 2023, so the age of the university is 2023 - 1789.*

Action: Calculator

Action Input: 2023 - 1789

Observation: Answer: 234

Thought: *The age of Georgetown University is 234 years. Now, I need to calculate 234 raised to the 0.23 power.*

Action: Calculator

Action Input: 234 ** 0.23

Observation: Answer: 3.506869841422834

Thought: *I now know the final answers.*

Final Answer: *The new building at Georgetown University is a residential hall being constructed as part of the redevelopment of the former Henle Village. The age of Georgetown University raised to the 0.23 power is approximately 3.51.*

Tools for our financial management

The ReACT framework incorporates a suite of agents, each tailored to handle specific aspects of financial management. Below, we introduce each agent and elucidate how it interacts with the reasoning component to fulfill user requests.

1. LangChain Tools

a. Google-serper

- **Function:** Facilitates real-time data fetching from the web.
- **Interaction with Reasoning:**
 - **Activation:** When the reasoning component identifies the need for up-to-date financial information (e.g., current exchange rates or stock prices), it activates the **Google-serper** agent.
 - **Execution:** The agent retrieves the requested data from the internet.
 - **Feedback:** The fetched data is returned to the reasoning component, which integrates it into the final response or uses it to inform further actions.

b. Ilm-math

- **Function:** Performs complex mathematical calculations.
- **Interaction with Reasoning:**
 - **Activation:** Triggered when the reasoning component requires advanced calculations, such as budgeting projections or statistical analyses.
 - **Execution:** The agent executes the necessary mathematical operations.
 - **Feedback:** The results are sent back to the reasoning component for incorporation into the user's financial summary or advice.

2. Personalized Tools

a. **process_receipt**

- **Function:** Extracts detailed information from receipt images, including merchant name, transaction time, and a list of purchased goods with their prices.
- **Interaction with Reasoning:**
 - **Activation:** Initiated when a user uploads a receipt image for expense tracking.
 - **Execution:** The agent processes the image to extract structured data.
 - **Feedback:** The extracted data is provided to the reasoning component, which then directs further actions such as expense logging and categorization.



For example for this Costco receipt, process_receipt tool will extract information and output as json like:

```
{
  "merchant": "Costco Wholesale",
  "time": "12/05/2024 17:50",
  "good_list": [
    { "name": "KSWTR40PK", "price": 3.99 },
    { "name": "KSWTR40PK", "price": 3.99 },
    { "name": "UNREAL BARS", "price": 10.89 },
    { "name": "ORG BANANAS", "price": 2.49 },
    { "name": "3 WHOLE MILK", "price": 2.75 }
  ]
}
```

b. category_merchant_tool

- **Function:** This tool is designed to verify whether a given merchant or category exists in the system. If the requested merchant or category is already registered, it returns the existing record with its corresponding ID. If it does not exist, the tool can create a new entry and return the newly generated ID. By ensuring that merchant and category information is properly indexed, this tool helps maintain a consistent, structured database of financial data.
- **Example Usage:**

1. Check for Existing Merchant

Action:	Observation	{'id': 11, 'name': 'frozen foods'}
<pre>{ "action": "create", "class": "category", "category_name": "frozen foods" }</pre>	In this step, the tool confirms that "Costco" is an existing merchant and returns the merchant's ID (1). No new entry is needed.	

2. Check for Existing Category

Action:	Observation:	"The object does not exist: frozen foods"
<pre>{ "action": "get", "class": "category", "category_name": "frozen foods" }</pre>	Since "frozen foods" does not exist as a category, the tool alerts us that no record is found.	

2. Create a New Category

Action:

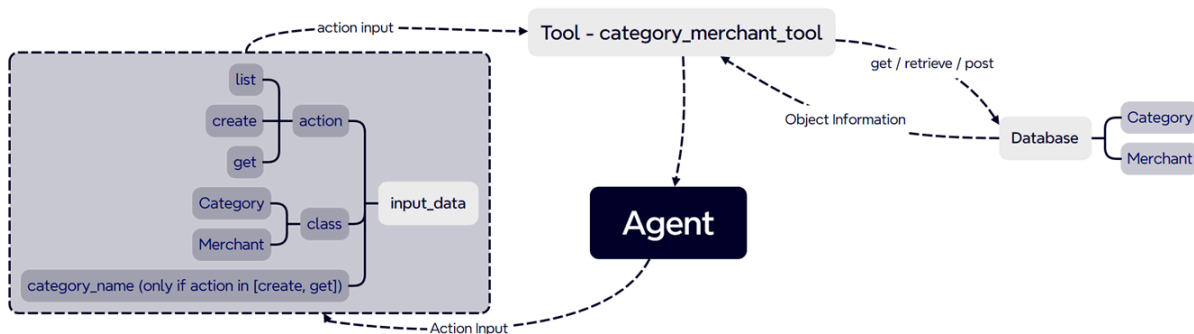
```
{  
  
  "action": "create",  
  
  "class": "category",  
  
  "category_name": "frozen  
foods"  
}
```

Observation:

```
{'id': 11, 'name': 'frozen foods'}
```

The tool successfully creates a new category with ID 11. This ID can now be used for subsequent expense recording activities.

Below is the flow chart for this tool:



c. create_or_update_expense_tool

- **Function:** This tool is responsible for creating a new expense record or updating an existing one. It takes inputs such as the expense amount, merchant ID, category ID, timestamp, and a description of the transaction. If the `expense_id` is null or absent, it creates a new record. If an `expense_id` is provided, it updates the existing record with the new details.
- **Example Usage:**

Create a New Category

Action:

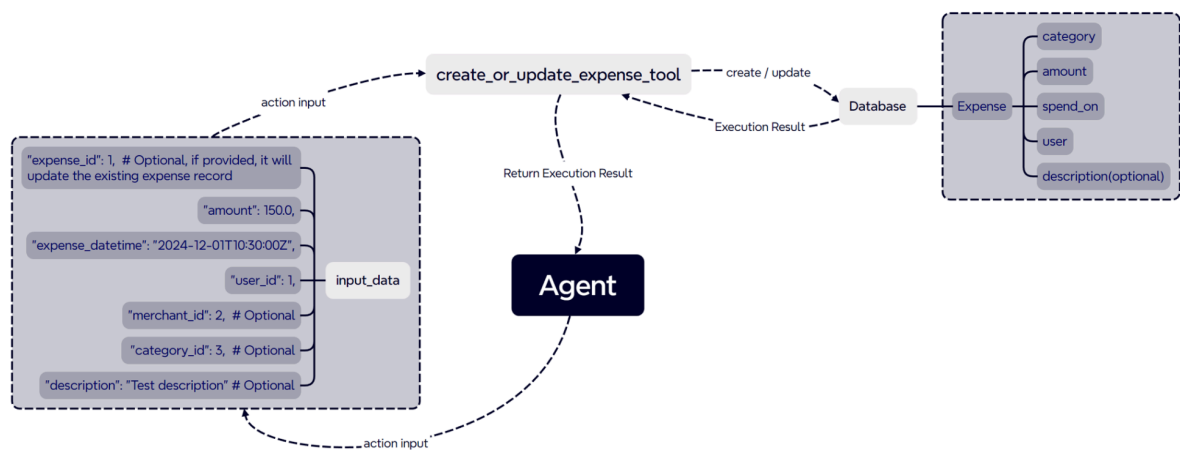
```
{
  "expense_id": null,
  "amount": 10,
  "merchant_id": 1,
  "category_id": 11,
  "expense_datetime":
    "2024-12-09T20:00:58Z",
  "description": "Purchased
    frozen foods"
}
```

Observation:

```
{
  "success": True,
  "expense": {
    "id": 26,
    "amount": 10,
    "expense_date":
      "2024-12-09T20:00:58+00:00",
    "merchant": "Costco",
    "category": "frozen foods",
    "description": "Purchased
      frozen foods",
    "created_at":
      "2024-12-10T18:37:59.731275+
      00:00",
    "updated_at":
      "2024-12-10T18:37:59.734444+
      00:00"
  }
}
```

In this instance, a new expense record is created. The tool returns a success status and a JSON object containing all relevant details, including the IDs for the merchant and category used, the amount spent, and the timestamp of the transaction. It ensures that expenses are accurately recorded and easily retrievable for later financial analysis or reporting.

Below is the flow chart for this tool:



d. parse_relative_time

- **Function:** Decodes natural language time expressions into precise datetime formats.
- **Initial Limitation:** Initially struggled with ambiguous phrases like "last night."
- **So we decided not to use this tool**

e. `get_related_time`

- **Function:** Accurately decodes complex natural language time expressions, addressing the limitations of `parse_relative_time`.
- **Interaction with Reasoning:**
 - **Activation:** Engaged when `parse_relative_time` fails to interpret ambiguous time expressions.
 - **Execution:** Utilizes advanced parsing techniques. First to get today's date and calculate the `day_diff` to get exactly time the user or input mean to accurately determine the intended datetime like last "night".
 - **Feedback:** Provides the precise datetime to the reasoning component for correct transaction logging.

Result (Demo)

Below is a demonstration of how our agent interprets a user's request, reasons through the steps involved, and engages with the appropriate tools to achieve the desired outcome. You can copy and paste this example into your report as an illustrative use case.

User Input:

"I've spent 10 purchasing frozen foods in Costco last night, please help me record it."

System Reasoning and Action Trace:

Thought Process:

To record the expense of purchasing frozen foods at Costco, the system will:

1. Identify or create the merchant ID for "Costco."
2. Identify or create the category ID for "frozen foods."
3. Determine the exact timestamp for "last night."
4. Create a new expense entry with the given details (amount: \$10, merchant: Costco, category: frozen foods, and the determined datetime).

Step-by-Step Execution:

- **Retrieving Merchant ID**
Action: `category_merchant_tool`
Action Input: `{"action": "get", "class": "merchant", "category_name": "Costco"}`

Observation: {"id": 1, "name": "Costco"}

Thought: The merchant "Costco" already exists and has an ID of 1.

- **Retrieving Category ID**

Action: category_merchant_tool

Action Input: {"action": "get", "class": "category",
"category_name": "frozen foods"}

Observation: "The object does not exist: frozen foods"

Thought: The category "frozen foods" does not exist, so it needs to be created.

- **Creating Category**

Action: category_merchant_tool

Action Input: {"action": "create", "class": "category",
"category_name": "frozen foods"}

Observation: {"id": 11, "name": "frozen foods"}

Thought: The category "frozen foods" is now created with ID 11.

- **Determining the Timestamp for "Last Night"**

To interpret "last night," we assume a specific time slot (e.g., 8:00 PM).

Action: get_related_time

Action Input: {"day_diff": -1, "time_slot": "20:00"}

Observation: 2024-12-09T20:00:58Z

Thought: The datetime corresponding to "last night" is determined as December 9, 2024, at 20:00:58 UTC.

Creating the Expense Record

Action: create_or_update_expense_tool

Action Input:

json

Copy code

```
{  
  "expense_id": null,  
  "amount": 10,  
  "merchant_id": 1,  
  "category_id": 11,  
  "expense_datetime": "2024-12-09T20:00:58Z",  
  "description": "Purchased frozen foods"  
}
```

Observation:

json

Copy code

```
{  
  "success": True,  
  "expense": {
```

```
{
  "id": 26,
  "amount": 10,
  "expense_date": "2024-12-09T20:00:58+00:00",
  "merchant": "Costco",
  "category": "frozen foods",
  "description": "Purchased frozen foods",
  "created_at": "2024-12-10T18:37:59.731275+00:00",
  "updated_at": "2024-12-10T18:37:59.734444+00:00"
}
```

-

Thought: The expense is successfully recorded in the system.

Final Answer:

The expense of \$10 for purchasing frozen foods at Costco has been recorded successfully, including the correct category and timestamp.