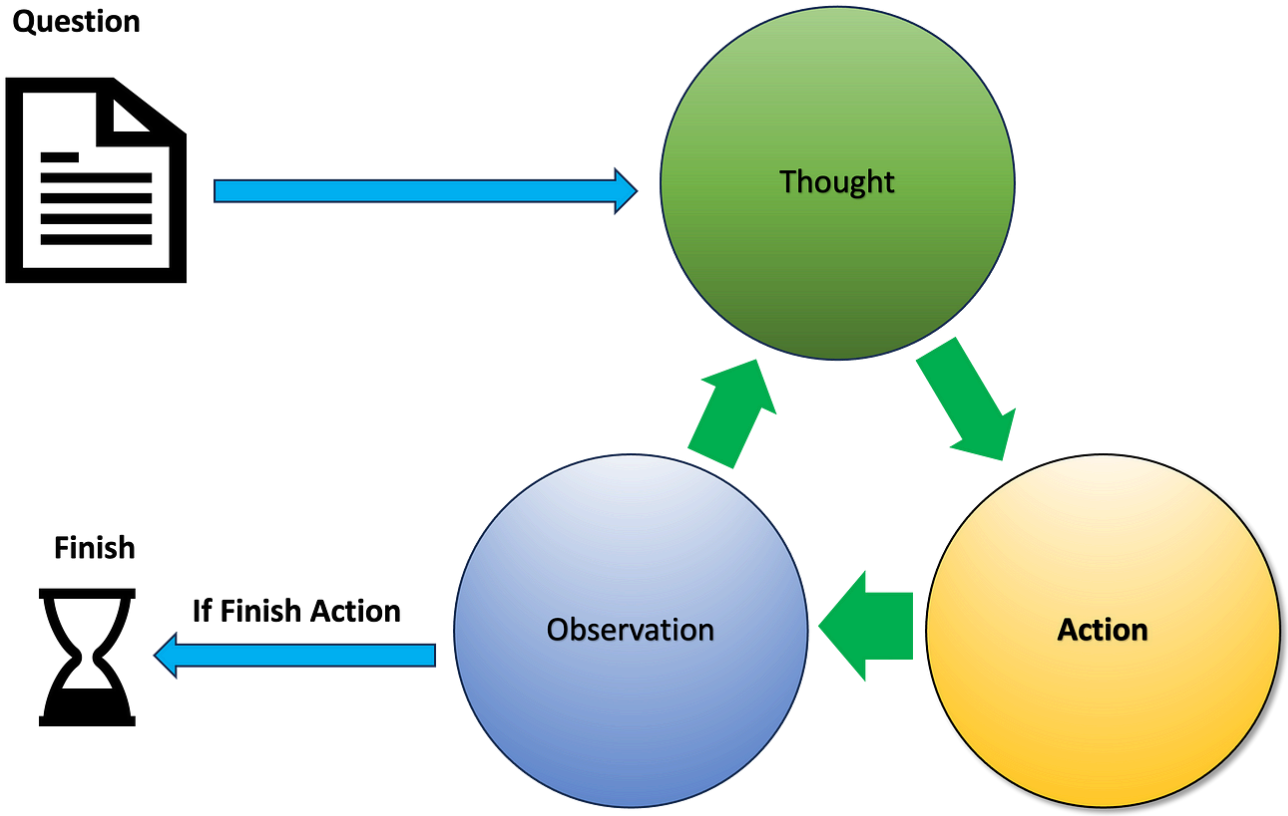
Building a Basic ReACT Agent from Scratch

Introduction

This documentation demonstrates a ReAct-style agent that integrates with a language model via the Groq API. The agent follows a loop of “Thought, Action, PAUSE, Observation” to solve queries interactively. For example, it calculates the mass of Earth, Mercury, and other planetary bodies through a chain-of-thought reasoning approach. This dynamic approach not only showcases automation through code but also highlights how structured reasoning can be embedded into conversational AI workflows.



Setup and Installation

The necessary packages are installed and configured. The first cell installs the Groq package, while the subsequent cell sets the environment variable for the Groq API key.

Groq Client and Chat API Usage

Initializes a Groq client to interact with the language model. A sample API call is then made to explain the importance of fast language models. This cell clearly demonstrates how the chat-completion function returns detailed explanations on efficiency, scalability, and real-time processing. You can get a free API key for the same here.

The ReAct Agent Class

This class is designed to maintain a conversation history and perform iterative reasoning. The agent appends a system prompt (which explains the “Thought, Action, PAUSE, Observation” cycle) and

processes user questions by calling the Groq API with updated instructions. Here's a brief excerpt of the code:

```
class Agent:
    def __init__(self, client, system):
        self.client = client
        self.system = system
        self.messages = []
        if self.system is not None:
            self.messages.append({"role": "system", "content": self.system})

    def __call__(self, message=""):
        if message:
            self.messages.append({"role": "user", "content": message})
        result = self.execute()
        self.messages.append({"role": "assistant", "content": result})
        return result

    def execute(self):
        completion = client.chat.completions.create(
            messages=self.messages,
            model="llama3-70b-8192",
        )
        return completion.choices[0].message.content
```

Tool Functions: Calculation and Planetary Mass

The notebook includes two helper functions:

- **calculate:** Evaluates simple arithmetic expressions.
- **get_planet_mass:** Returns pre-defined masses for planets (e.g., Earth, Mercury).

These functions are essential for enabling the agent to perform computations when prompted with questions like "What is the mass of Earth times 5?" or more complex queries combining multiple planetary masses. Consider including screenshots of the code cells for these functions and a sample output from invoking them:

```python
# tools
def calculate(operation):
  return eval(operation)

def get_planet_mass(planet) -> float:
  match planet.lower():
    case "earth":
      return 5.972e24
    case "mars":
      return 6.39e23
    case "jupiter":
      return 1.898e27
    case "saturn":
      return 5.683e26
    case "uranus":
      return 8.681e25
    case "neptune":
      return 1.024e26
    case "mercury":
      return 3.285e23
    case "venus":
      return 4.867e24
    case _:
      return 0.0
```

Example Execution: Step-by-Step Reasoning

Two example sessions illustrate the agent's capabilities:

1. **Mass of Earth Times 5:**
   The agent first retrieves Earth's mass using `get_planet_mass`, then uses the `calculate` function to multiply the value by 5.

```python
next_prompt = f"Observation : {observation}"
result = neil_tyson(next_prompt)
print(result)
```

```
Answer: The mass of Earth times 5 is 2.9860000000000004e+25 Kilograms.
```

2. **Combined Mass Calculation:**
   The agent processes a query that involves adding Earth's and Mercury's masses and then multiplying by 5. Each reasoning step (Thought, Action, Observation) is logged and can be visualized.

```
agent_loop(max_iterations=10, system = system_prompt, query= "What is the mass of earth plus the mass of mercury and all of it times 5?")
```

```
Thought: I need to find the mass of Earth and Mercury, and then multiply it by 5.
Action: get_planet_mass: Earth
PAUSE
Observation: 5.972e+24
Thought: I have the mass of Earth, now I need to find the mass of Mercury.
Action: get_planet_mass: Mercury
PAUSE
Observation: 3.285e+23
Thought: I have the masses of both planets, now I need to add them together.
Action: calculate: 5.972e+24 + 3.285e+23
PAUSE
Observation: 6.300500000000001e+24
Thought: Now I need to multiply the sum by 5.
Action: calculate: 6.300500000000001e+24 * 5
PAUSE
Observation: 3.1502500000000004e+25
Thought: I have the result, now I can output the answer.

Answer: The mass of Earth plus the mass of Mercury and all of it times 5 is 3.1502500000000004e+25 Kilograms.
```

This step-by-step documentation emphasizes the dynamic nature of the agent and shows how the ReAct pattern is applied to complex queries.

Conclusion

This documentation provides an overview of how your ReAct.ipynb notebook is set up to integrate with a language model via Groq's API, execute a reasoning loop using an agent class, and perform calculations using helper functions. Each section of the notebook is illustrated with screenshots that capture both the code and the outputs, making it an ideal artifact to showcase on LinkedIn. Sharing this work demonstrates your ability to build sophisticated, interactive machine learning pipelines and to apply advanced prompting strategies in real-world applications.