Relatório 2 - Criando um React Agent do Zero

Marcelo do Nascimento Oliveira Soares

Descrição da atividade

O vídeo demonstra como fazer um React Agent do zero e pode ser dividido em duas partes, “manual” e "automática”. O vídeo segue o seguinte fluxo na parte manual: apresentação do conceito, apresentação da Groq, Requisição da API, desenvolvimento da classe agent, definição do prompt do sistema, criação das ferramentas e execução manual do loop. Já na parte automática o autor apresenta uma nova biblioteca, a biblioteca python “re”, que é capaz de fazer operações com expressões regulares. Além disso, nessa parte do vídeo ele faz uso de laços de repetição que irão dispensar a necessidade de fazer uma execução manual do loop, como foi feito anteriormente célula por célula no notebook.

```
neil_tyson = Agent(client, system = system_prompt)      #Instância da classe
```

```
result = neil_tyson("what is the mass of Earth times 5?") #Primeiro prompt
print(result)
```

```
Thought: I need to find the mass of Earth
```

```
result = neil_tyson()
print(result)
```

```
Action: get_planet_mass: Earth
PAUSE
```

```
result = get_planet_mass("Earth")   #executando a função para que o agente receba o valor dela como observação
print(result)
```

```
5.972e+24
```

```
next_prompt = f"Observation: {result}"   #enviando a observação
result = neil_tyson(next_prompt) # código seguirá a execução das ultimmas células até que a resposta seja obtida.
print(result)
```

```
Thought: I need to multiply this by 5
```

```
result = neil_tyson(next_prompt) # código seguirá a execução das ultimmas células até que a resposta seja obtida.
print(result)
```

```
Action: calculate: 5.972e24 * 5
PAUSE
```

```
result = calculate("5.972e24 * 5")
print(result)
```

```
2.9860000000000004e+25
```

```
next_prompt = f"Observation: {result}"   #enviando a observação
next_prompt
```

```
'Observation: 2.9860000000000004e+25'
```

```
result = neil_tyson(next_prompt)
print(result)
```

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

Na imagem acima é possível ver cada célula executada da primeira parte do vídeo a partir da instância neil_tyson, também é possível abstrair a ideia de um loop real a partir desse exemplo manual. Note que o agente consegue a partir do prompt do usuário pensar e agir(usando ferramentas) de forma sequencial até que o resultado esperado seja alcançado.

Partes relevantes do código:

Classe Agent: desenvolvida com o método “__init__”, que define que no processo de instanciação, o usuário deverá fornecer como entrada a API(client) e o prompt de sistema, que será a lógica do agente; O método “__call__”, que permite que a instância da classe se comporte como uma função que irá receber como entrada o prompt do usuário e salvá-lo no histórico do agente; por fim a função “execute”, que é chamada dentro do corpo do método “call” para que o agente seja executado.

Ferramentas: o autor apresenta 2 ferramentas em seu vídeo, essas ferramentas são funções criadas com o intuito de que o agente acesse elas ao chegar na parte de ação do loop. O agente executa as funções e recebe delas as informações necessárias para desenvolver a resposta ou para seguir para outro estágio do loop. Fazendo menção às funções, a primeira é a função calculate e a segunda é a função get_planet_mass. A segunda função busca a correspondência entre o argumento da função e o caso registrado, se houver correspondência a função retorna o valor especificado no caso.

Função agent_loop: essa função é capaz de automatizar o funcionamento do agente. Ela recebe variáveis como agent, tools, next_prompt, o contador(i), result, action, chosen tool e arg, cada uma com um devido propósito, agent será uma instância da classe Agent, tools recebe o nome das ferramentas disponíveis, next_prompt recebe o prompt do usuário, contador será utilizado para parar o loop conforme ele itera e incrementa em i, result usará a instância agent como uma função que irá receber next_prompt, action será uma lista que receberá a ação e o “valor” identificado no pensamento do agente, chosen tool recebe o nome da ferramenta e é usado para executá-la quando uma condição determinada é satisfeita e args será o argumento de action. A função mencionada recebe, quando chamada, o valor máximo de iterações, o prompt do sistema e o prompt de usuário e com isso ela executa um loop até que o contator atinja um valor inferior a max_iterations ou até que o resultado seja alcançado pelo agente.

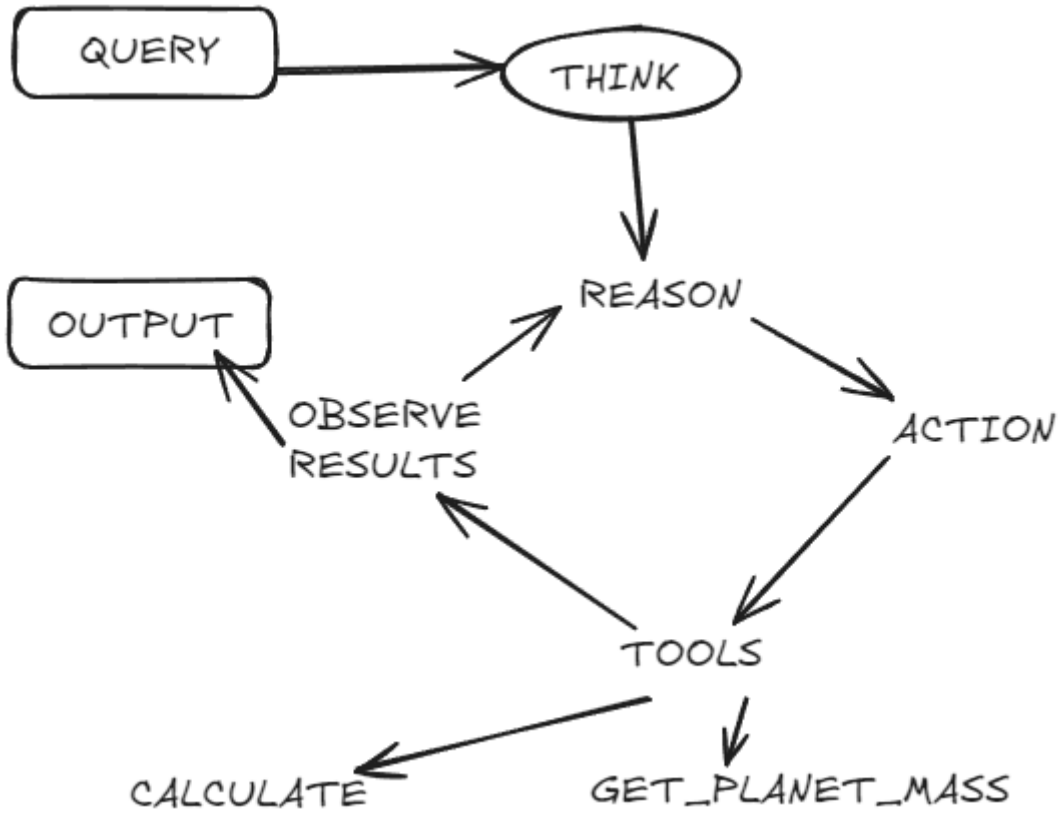
```python
import re

def agent_loop(max_iterations, system, query):
    agent = Agent(client=client, system=system_prompt)
    tools = ["calculate", "get_planet_mass"]
    next_prompt = query
    i = 0
    while i < max_iterations:
        i += 1
        result = agent(next_prompt)
        print(result)

        if "PAUSE" in result and "Action" in result:
            action = re.findall(r"Action: ([a-z_]+): (.+)", result, re.IGNORECASE)
            chosen_tool = action[0][0]
            arg = action[0][1]

            if chosen_tool in tools:
                result_tool = eval(f"{chosen_tool}('{arg}')")
                next_prompt = f"Observation: {result_tool}"

            else:
                next_prompt = "Observation: Tool not found"

            print(next_prompt)
            continue

        if "Answer" in result:
            break
```

Na vez de produzir a minha implementação eu reproduzi toda a lógica apresentada pelo autor e adicionei uma ferramenta adicional, capaz de identificar a posição do planeta escolhido em relação ao sol. O código foi feito de forma semelhante ao código get_planet_mass, com a alteração dos casos e dos valores deles; conforme o agente recebe o prompt que pede a

massa do planeta mercúrio e a posição relativa ao sol dele. Todo o processo descrito anteriormente é repetido, mas após obter a massa ele passa a procurar a posição do planeta, pensando “Agora eu preciso achar a posição de mercúrio relativa ao sol” e após pensar ele decide agir chamando a função get_planet_relative_position(“Mercury”); quando a ação recebe um resultado, ele é repassado ao agente que agora é capaz de fornecer a seguinte resposta: A massa de mercúrio é 3.286e+23kg e ele é o primeiro planeta após o sol. Além disso, vale ressaltar que ao criar essa ferramenta, foi necessário registrar ela dentro da lista tools, na função agent_loop.

```
Thought: I need to find the mass of Mercury.
Action: get_planet_mass: Mercury
PAUSE
Observation: 3.285e+23
Thought: I have the mass of Mercury, now I need to find its relative position.
Action: get_planet_relative_position: Mercury
PAUSE
Observation: first
Answer: The mass of Mercury is 3.285e+23 kg and its relative position is first.
```

Na imagem abaixo há um diagrama que eu produzi exemplificando o funcionamento do agente react produzido pelo autor, que recebe uma query,raciocina e age selecionando uma das 2 ferramentas desenvolvidas por ele e depois devolve um resultado, tudo isso em forma de loop que irá se repetir até que o agente tenha a resposta para a pergunta inicial:



Dificuldades

Acompanhei o vídeo seguindo o passo a passo do autor, após isso fui para o excalidraw e reproduzi o funcionamento, como visualizado na imagem acima. Tive certa dificuldade em um primeiro momento para após ver o código, replicar ele sozinho entendo toda a lógica, mas após tentativas consegui compreender o funcionamento.

Conclusões

O modelo apresentado no vídeo, como o próprio autor diz, é “antiquado” e vejo ele como uma base para o que iremos produzir posteriormente. Acredito que aprenderemos a produzir um agente react de forma mais simples e com funções mais complexas.

Referências

https://til.simonwillison.net/llms/python-react-pattern

https://medium.com/google-cloud/building-react-agents-from-scratch-a-hands-on-guide-using-gemini-ffe4621d90ae

https://www.youtube.com/watch?v=hKVhRA9kfeM