

Relatório 2 - Prática: Python: Criando um ReAct Agent do Zero (I)

Enzo Rodrigues Novais Dias

Descrição da atividade

Eu desenvolvi um Agente ReAct do 0, sem nenhum uso de frameworks, colocando o conhecimento adquirido no Fastcamp até então em prática.

O vídeo inicia explicando o que é um Agente ReAct e, coincidentemente, o autor do vídeo estava usando um dos materiais que também utilizei de fonte na primeira atividade do fastcamp, o "[React: Synergizing reasoning and acting in language models](#)".

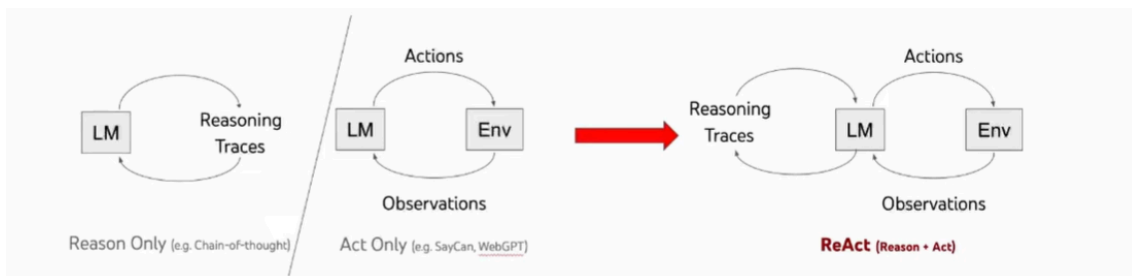


Figura 1: Funcionamento do ReAct - [3]

Quanto ao código, inicialmente compreendi e repliquei o que foi feito pelo professor na videoaula. Depois adicionei implementações novas com o que eu aprendi ao decorrer da aula.

Inicialmente, ele configura o ambiente para acessar a API do Groq e testa a conexão com uma consulta simples. Logo após ele cria a Classe Agent que, em resumo, gerencia conversas com um modelo de linguagem (llama3-70b-8192), mantendo o histórico de mensagens e facilitando a interação com a API.

```
class Agent:
    def __init__(self, client: Groq, system: str = "") -> None:
        self.client = client
        self.system = system
        self.messages: list = []
        if self.system:
            self.messages.append({"role": "system", "content": 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 = self.client.chat.completions.create(
            model="llama3-70b-8192", messages=self.messages
        )
        return completion.choices[0].message.content
```

Figura 2: Criação da Classe Agent - [1]

Após isso, ele cria o Prompt do sistema utilizando uma abordagem ReAct, definindo o formato de interação através de um loop "Thought, Action, PAUSE, Observation." Definindo também que a resposta deve finalizar com "Answer" e descrevendo duas funções:

- calculate: para executar cálculos matemáticos
- get_planet_mass: para obter a massa de um planeta, em kg

Aqui que vem a minha primeira adição, criei também as seguintes funções:

- `get_planet_gravity`: recebe o nome de um planeta e retorna sua gravidade, em m/s^2 , comparando com a da Terra
- `get_planet_temperature`: recebe o nome de um planeta e retorna sua temperatura máxima, mínima e média, em Celsius.

```
system_prompt = """
You run in a loop of Thought, Action, PAUSE, Observation.
At the end of the loop you output an Answer.
Use Thought to describe your thoughts about the question you have been asked.
Use Action to run one of the actions available to you - then return PAUSE.
Observation will be the result of running those actions.

Your available actions are:

calculate:
e.g. calculate: 4 * 7 / 3
Runs a calculation and returns the number - uses Python so be sure to use floating point syntax if necessary

get_planet_mass:
e.g. get_planet_mass: Earth
returns mass of the planet in kg

get_planet_gravity:
e.g. get_planet_gravity: Mars
returns the surface gravity of the planet in  $\text{m/s}^2$  and how it compares to Earth

get_planet_temperature:
e.g. get_planet_temperature: Mars
returns average, minimum and maximum temperature of the planet in Celsius
```

Figura 3: Prompt do sistema - [1]

```
def get_planet_gravity(planet: str) -> str:
    gravity_data = {
        "mercury": {"gravity": 3.7, "compared_to_earth": "38% of Earth"},
        "venus": {"gravity": 8.87, "compared_to_earth": "90% of Earth"},
        "earth": {"gravity": 9.81, "compared_to_earth": "reference (100%)"},
        "mars": {"gravity": 3.71, "compared_to_earth": "38% of Earth"},
        "jupiter": {"gravity": 24.79, "compared_to_earth": "253% of Earth"},
        "saturn": {"gravity": 10.44, "compared_to_earth": "106% of Earth"},
        "uranus": {"gravity": 8.87, "compared_to_earth": "90% of Earth"},
        "neptune": {"gravity": 11.15, "compared_to_earth": "114% of Earth"}
    }

    planet = planet.lower().strip()
    if planet in gravity_data:
        data = gravity_data[planet]
        return f"Gravity in {planet.capitalize()}: {data['gravity']}  $\text{m/s}^2$  ({data['compared_to_earth']})."
    else:
        return f"Gravity data not available for '{planet}'."

def get_planet_temperature(planet: str) -> str:
    planet_temps = {
        "mercury": {"min": -173, "max": 427, "avg": 167, "desc": "extremely variable due to lack of atmosphere"},
        "venus": {"min": 462, "max": 462, "avg": 462, "desc": "nearly uniform due to dense atmosphere"},
        "earth": {"min": -89, "max": 56, "avg": 15, "desc": "habitable, with variations by region and season"},
        "mars": {"min": -153, "max": 20, "avg": -63, "desc": "with large daily and seasonal variations"},
        "jupiter": {"min": -145, "max": -145, "avg": -145, "desc": "relatively constant in the upper layers"},
        "saturn": {"min": -178, "max": -178, "avg": -178, "desc": "cold and constant in the upper layers"},
        "uranus": {"min": -224, "max": -224, "avg": -224, "desc": "extremely cold and uniform"},
        "neptune": {"min": -218, "max": -218, "avg": -218, "desc": "very cold with occasional thunderstorms"}
    }

    planet = planet.lower().strip()
    if planet in planet_temps:
        data = planet_temps[planet]
        return f"Temperature for {planet.capitalize()}: Average of {data['avg']}°C (min: {data['min']}°C, max: {data['max']}°C, {data['desc']})."
    else:
        return f"No temperature data available for '{planet}'."
```

Figura 4: Definições das funções criadas por mim - [Github]

Após isso, o professor roda o agente sem loop, onde é gerado pensamentos e decisão de ações, depois o código externo executa as ações solicitadas, os resultados são fornecidos de volta e o processo continua até que o agente tenha a resposta final, que no caso do exemplo do professor era a massa da Terra multiplicada por 5.

Em seguida ele implementa a função `agent_loop`, que automatiza completamente o ciclo ReAct visto, gerenciando o ciclo de pensamento, ação e observação até chegar a uma resposta final, sem necessidade de intervenção manual.

Agora indo para a minha aplicação após o Prompt e funções, também utilizei da função `agent_loop`, que incrementa o contador de iterações, envia o prompt atual ao agente e imprime sua resposta, caso contenha PAUSE ou ACTION, é extraído o nome da ferramenta e do argumento e, caso ele seja válido, executa a função e prepara a observação. Já caso tenha um ANSWER, o loop é encerrado.

Com isso, realizei duas queries para testar:

```
agent_loop(10, system = system_prompt, query="If I weigh 75 kg on Earth, how much would I weigh on Mars and Jupiter?")

Thought: I need to find the surface gravity of Mars and Jupiter to calculate my weight on those planets.

Action: get_planet_gravity: Mars
PAUSE
Observation: Gravity in Mars: 3.71 m/s2 (38% of Earth).
Thought: I now have the surface gravity of Mars, but I also need the surface gravity of Jupiter.

Action: get_planet_gravity: Jupiter
PAUSE
Observation: Gravity in Jupiter: 24.79 m/s2 (253% of Earth).
Thought: I now have the surface gravity of both Mars and Jupiter. I need to calculate my weight on these planets.

Action: calculate: 75 * (3.71 / 9.81)
PAUSE
Observation: 28.363914373088686
Thought: I have now calculated my weight on Mars. I need to calculate my weight on Jupiter.

Action: calculate: 75 * (24.79 / 9.81)
PAUSE
Observation: 189.52599388379204
Thought: I have now calculated my weight on both Mars and Jupiter.

Answer: If I weigh 75 kg on Earth, I would weigh approximately 28.36 kg on Mars and approximately 189.53 kg on Jupiter.
```

Figura 5: Cálculo de massa em diferentes planetas - [\[Github\]](#)

```
agent_loop(10, system = system_prompt, query="What is the average temperature difference between Mars and Venus?")

Thought: I need to find the average temperatures of Mars and Venus.
Thought: I need to find the average temperatures of Mars and Venus.
Action: get_planet_temperature: Mars
PAUSE
Observation: Temperature for Mars: Average of -63°C (min: -153°C, max: 20°C). With large daily and seasonal variations.
Thought: Now I have the temperature of Mars. I need to get the temperature of Venus.
Action: get_planet_temperature: Venus
PAUSE
Observation: Temperature for Venus: Average of 462°C (min: 462°C, max: 462°C). Nearly uniform due to dense atmosphere.
Thought: Now I have the temperatures of both Mars and Venus. I need to calculate the difference in average temperatures.
Action: calculate: 462 - (-63)
PAUSE
Observation: 525
Thought: I have calculated the average temperature difference between Mars and Venus.
Answer: The average temperature difference between Mars and Venus is 525°C.
```

Figura 6: Diferença de temperaturas entre planetas - [\[Github\]](#)

Dificuldades

Tive um pouco de dificuldade na criação da função `get_planet_temperature`, em pesquisar de maneira coerente sobre as temperaturas e condições climáticas de cada planeta


do sistema solar, mas acabou sendo um conhecimento divertido/interessante, que eu não esperava adquirir nesse contexto.

Conclusões

A atividade foi enriquecedora, pois pude colocar em prática aquilo que aprendi na teoria da última atividade, mostrando que é sim possível resolver problemas “complexos” através desse ciclo, me deixando animado para aplicar em problemas maiores e também para as próximas atividades.

Poder pesquisar e aplicar funções “novas” como a `get_planet_gravity` e `get_planet_temperature` mostrou a grande capacidade do agente em tarefas de diferentes áreas e contextos, podendo resolver cálculos ou somente “textos”.

Referencias

- 1 -  Python: Create a ReAct Agent from Scratch
- 2 - [How To Create AI Agents With Python From Scratch \(Full Guide\)](#)
- 3 - Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). *REACT: Synergizing Reasoning and Acting in Language Models*. Disponível em: [React: Synergizing reasoning and acting in language models](#)