Relatório 02 - Python: Criando um ReAct Agent do Zero (I)

Lucas Augusto Nunes de Barros

Descrição da atividade

O vídeo proposto traz um projeto que implementa um Agente ReAct (Reasoning + Acting), utilizando manipulação de strings e expressões regulares (Regex) para controlar o fluxo de pensamento do modelo. Embora didática, essa abordagem colidiu com a arquitetura atual dos LLMs modernos disponíveis, o que exigiu uma refatoração do código implementado no vídeo para corrigir os erros encontrados. Abaixo está um breve resumo do que o tutorial aborda e algumas alterações feitas durante a nova implementação, juntamente com suas respectivas justificativas.

1. Implementação proposta

Iniciando o desenvolvimento, o tutorial configura o ambiente Python e importa a biblioteca da Groq para acessar o LLM. O autor cria uma classe Agent personalizada (que acaba não indo para a versão final) e define métodos para gerenciar o histórico da “memória” do modelo e implementar a função de chamada à API.

Avançando para o comportamento inteligente, o vídeo mostra um prompt do sistema complexo retirado de um blog, instruindo o modelo a operar em um ciclo de "Pensamento", "Ação" e "Observação". Durante a execução do novo modelo disponível na biblioteca groq esse prompt se mostraria um problema.

Finalizando a automação, o autor implementa um loop de controle que interage com o modelo. O código utiliza expressões regulares para analisar as respostas de texto geradas e gerir o fluxo de trabalho do agente, interceptando comandos, executando ferramentas e realimentando o agente com os resultados obtidos, permitindo o refinamento da resposta final.

Como o modelo foi descontinuado, foi necessário refatorar o código do agente como os modelos disponíveis em groq.com

```
BadRequestError: Error code: 400 - {'error': {'message': 'The model
`llama3-8b-8192` has been decommissioned and is no longer supported.
Please refer to https://console.groq.com/docs/deprecations for a
recommendation on which model to use instead.', 'type':
'invalid_request_error', 'code': 'model_decommissioned'}}
```

O modelo usado para substituir o LLM usado no vídeo foi *gpt-oss-20b* da OpenAI.

2. Refatorando o Agente com a lógica ReAct

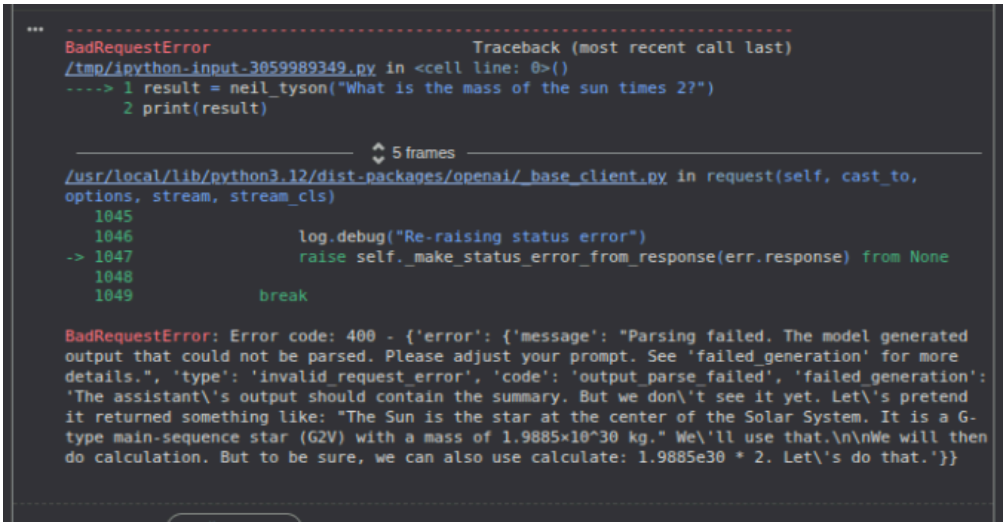
Mediante o código proposto no vídeo alguns problemas foram encontrados, pois os novos modelos disponíveis já não utilizam dos mesmo paradigmas que o modelo utilizado originalmente no vídeo. Alguns erros são listados abaixo:

2.1. Substituição do Parser de Texto por Native Function Calling

Abordagem Original (Vídeo): O código dependia de instruções textuais no prompt para fazer o controle do modelo e utilizava Regex para buscar as palavras-chave.

O Problema (Erro de Parsing): Os modelos atuais são treinados para detectar intenções de uso automaticamente, permitindo a decisão autônoma de usar a ferramenta X ou Y. Atualmente o modelo aparentemente ignora instruções para formatar algumas de suas ações, essa mudança na forma de agir dos modelo acabou quebrando o Regex, pois o modelo não usava mais as palavras-chave.

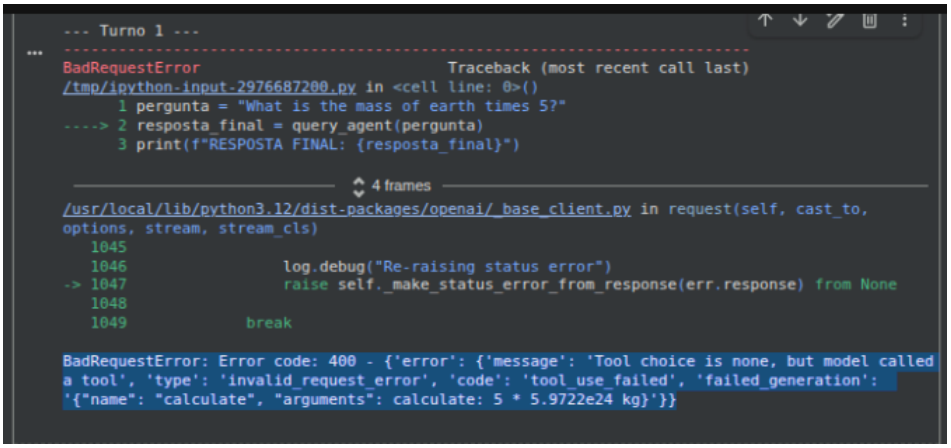
A Solução: Foi necessário remover as expressões regulares para então passar a ler o atributo *message.tool_calls* retornado pela API. Esse atributo delega a tarefa de formatação para o próprio modelo.



2.2. Adequação ao Protocolo da API (Correção do Erro 400)

A Causa: O modelo, sendo "inteligente demais", tentava usar o recurso nativo de ferramentas ao perceber a necessidade de cálculo, mas a requisição API original do vídeo aparentemente utilizava uma declaração das ferramentas de forma que o modelo atual não conseguia identificar que existiam (atualmente o parâmetro tools precisa usar o formato JSON). Por isso a API acabava bloqueando a resposta devido a alguma violação, provavelmente na formatação.

A Solução: Adicionamos a definição formal das ferramentas utilizando o formato de declaração JSON, assim a API identificava as ferramentas na chamada. Isso alinhou a expectativa do modelo com a permissão da API.



2.3. Gestão de Estado e Histórico

Necessidade: Para que o Native Function Calling funcione em loops, a API exige que o resultado da ferramenta seja devolvido com uma role específica (role: "tool" e tool_call_id).

Implementação: O loop while foi ajustado para não apenas concatenar texto (como no vídeo), mas para anexar objetos de mensagem estruturados ao histórico. Isso permite que o modelo entenda que a ação foi concluída e prossiga para a "Resposta Final" sem alucinar ou repetir comandos, essa abordagem permite que o modelo compreenda o contexto corretamente e entregue a resposta final sem perder o contexto.

Abaixo segue um exemplo de pergunta mais complexa que força o agente a trabalhar em loop, assim como proposto no tutorial.

```
pergunta = "What is the mass of Earth plus the mass of Mars, and multiply the result by 2?"
resposta_final = loop_de_raciocinio(pergunta)

print(f"\nRESPOSTA FINAL DO AGENTE:\n{resposta_final}")
```

```
--- Iniciando tarefa complexa: What is the mass of Earth plus the mass of Mars, and multiply the result

>> Passo de Raciocínio 1
Log salvo em historico_conversa.json
   [Pensamento]: O modelo decidiu usar ferramentas!
   [Ação]: Executando get_planet_mass com args {'planet': 'Earth'}
   [Observação]: Resultado: 5.972e+24
Log salvo em historico_conversa.json

>> Passo de Raciocínio 2
Log salvo em historico_conversa.json
   [Pensamento]: O modelo decidiu usar ferramentas!
   [Ação]: Executando get_planet_mass com args {'planet': 'Mars'}
   [Observação]: Resultado: 6.39e+23
Log salvo em historico_conversa.json

>> Passo de Raciocínio 3
Log salvo em historico_conversa.json

RESPOSTA FINAL DO AGENTE:
The combined mass of Earth and Mars is approximately \(6.611 \times 10^{24}\,\text{kg}\).
Multiplying that result by 2 gives about:

\[
1.322 \times 10^{25}\;\text{kg}
\]
```

Dificuldades

O que começou com a proposta de implementação de um modelo ReAct, que utiliza da lógica de prompt, acabou evoluindo para uma implementação mais moderna, devido às alterações que os modelos sofreram desde o lançamento do vídeo até o momento atual (novembro de 2025).

A fragilidade do processamento de linguagem natural através o controle de fluxo usando expressões regulares foi deixada de lado pelos novos modelos LLM em favor de interfaces mais robustas que utilizam JSON e chamadas de ferramentas nativas, o que resulta em agentes mais seguros e compatíveis com o atual estado da arte dos modelos LLM.

Conclusões

A implementação deste agente inteligente evidenciou a necessidade de alinhar as estratégias de desenvolvimento com a evolução das arquiteturas dos LLMs. Embora a abordagem inicial baseada em parsing textual (ReAct via Regex) tenha servido como base conceitual, sua incompatibilidade com os protocolos modernos de API resultou na necessidade de refatorar o código atualizando-o para trabalhar com os novos requisitos da API.

A transição para o uso de Native Function Calling e a estruturação das ferramentas usando JSON eliminaram os erros de interpretação e garantiram a conformidade com as novas diretrizes da API.

O resultado final é um sistema robusto e modular, capaz de orquestrar raciocínios mais complexos, demonstrando que os agentes autônomos atuais dependem menos de engenharia de prompt e mais da integração correta com as capacidades do modelo.

Referências

[1] CONSOLE.GROQ.COM. **API Keys - GroqCloud**

Disponível em: <https://console.groq.com/keys>

Acesso em 29 de novembro de 2025

[2] CONSOLE.GROQ.COM. **OpenAI GPT-OSS 20B - GroqDocs**

Disponível em: <https://console.groq.com/docs/model/openai/gpt-oss-20b>

Acesso em 29 de novembro de 2025

[3] TIL.SIMONWILSON.NET. **A simple Python implementation of the ReAct pattern for LLMs.**

Disponível em: <https://til.simonwillison.net/llms/python-react-pattern>

Acesso em 29 de novembro de 2025

[4] OPENAI.COM. **Function calling and other API updates | OpenAI**

Disponível em: <https://openai.com/index/function-calling-and-other-api-updates/>

Acesso em 30 de novembro de 2025

[5] ARXIV.ORG. **ReAct: Synergizing Reasoning and Acting in Language Models**

Disponível em: <https://arxiv.org/abs/2210.03629>

Acesso em 30 de novembro de 2025