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

Marcos Dos Anjos

Descrição da atividade

Criando um ReAct Agent do Zero (I)

1. Contexto:

Os Modelos de Linguagem de Grande Escala (LLMs) revolucionaram a forma como interagimos com sistemas computacionais, mas apresentam limitações fundamentais:

1. Conhecimento estático limitado ao momento do treinamento,
2. Incapacidade de executar ações no mundo real,
3. Ausência de raciocínio estruturado para problemas complexos.

Para superar essas limitações, temos como alternativa o conceito de Agentes de IA - sistemas que combinam a capacidade de raciocínio dos LLMs com a habilidade de executar ações através de ferramentas externas. Este relatório documenta a implementação prática de um agente baseado no padrão ReAct (*Reasoning + Acting*), desenvolvido sem o uso de frameworks abstratos como *LangChain ou LlamaIndex,* utilizando Python puro e a API Groq.

O trabalho foi desenvolvido com base na videoaula "Python: Create a ReAct Agent from Scratch" do canal Alejandro AO [1], que apresenta os fundamentos teóricos e práticos para construção de agentes inteligentes.

2. O Padrão ReAct: Fundamentos Teóricos

O padrão ReAct, estrutura o comportamento do agente em um ciclo de três etapas principais: Thought (Pensamento), Action (Ação) e Observation (Observação).

Na etapa de Thought, o agente articula explicitamente seu raciocínio sobre o problema. Ao invés de simplesmente executar ações de forma mecânica, o modelo "pensa em voz alta" sobre o que precisa fazer, quais informações ainda faltam e qual ferramenta seria mais adequada para obtê-las. Esta verbalização do raciocínio é crucial - ela torna o processo transparente e permite que o modelo siga uma linha lógica de pensamento estruturada.

Na etapa de Action, o agente seleciona uma ferramenta disponível e define os parâmetros para sua execução. A ação é expressa em um formato textual estruturado que pode ser parseado pelo sistema de controle. Importante destacar que neste momento o agente deve emitir a palavra-chave "PAUSE", sinalizando que aguarda a execução externa da ferramenta antes de continuar seu raciocínio.

Na etapa de Observation, o resultado real da execução da ferramenta é reinserido no contexto do agente como uma nova mensagem. Essa informação torna-se parte do histórico conversacional, permitindo que o modelo a utilize nas próximas iterações de pensamento.

O ciclo se repete quantas vezes forem necessárias até que o agente determine possuir informações suficientes para fornecer uma resposta final ao usuário.

Visualização do Ciclo ReAct

Para facilitar a compreensão do fluxo descrito, apresento abaixo o diagrama conceitual do padrão ReAct:

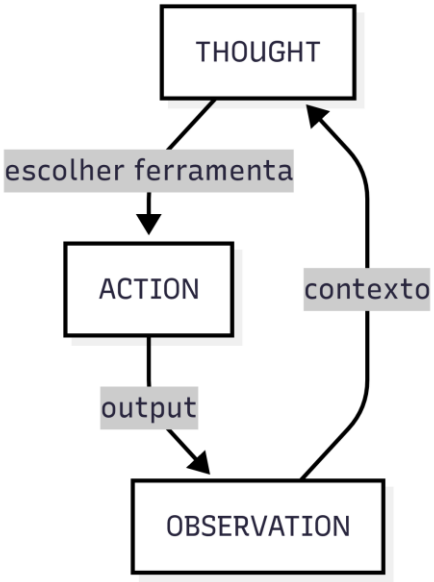

Figura 1: Representação visual do ciclo fundamental ReAct. O diagrama mostra três caixas principais (THOUGHT, ACTION, OBSERVATION) conectadas por setas direcionais. A seta de THOUGHT para ACTION é rotulada "escolher ferramenta", indicando o processo de decisão. A seta de ACTION para OBSERVATION é rotulada "output", representando o resultado da execução da ferramenta. A seta de OBSERVATION de volta para THOUGHT é rotulada "contexto", mostrando como a informação obtida alimenta o próximo ciclo de raciocínio. Este loop continua até que o agente gere uma resposta final.

Engenharia de Prompt: O Coração do Agente

A implementação de um agente ReAct sem frameworks exige extrema atenção à engenharia de prompt. Diferentemente de sistemas que abstraem essa complexidade, aqui o System Prompt é o principal mecanismo de instrução do comportamento do agente.

O prompt de sistema precisa conter quatro elementos essenciais:

1. Definição clara do loop operacional: O modelo precisa ser explicitamente instruído de que deve operar no formato "Thought → Action → PAUSE → Observation", e que ao final do loop deve fornecer uma "Answer".

2. Instruções detalhadas de cada etapa: Como formular pensamentos estruturados, como especificar ações no formato correto, quando pausar a execução e como interpretar as observações recebidas.

3. Catálogo completo de ferramentas disponíveis: Cada ferramenta que o agente pode utilizar deve ser documentada com nome, sintaxe de uso e exemplo prático. Por exemplo: "calculate: realiza cálculos matemáticos usando sintaxe Python. Exemplo: calculate: 4 * 7 / 3".

4. Few-shot example: Um exemplo completo de sessão, demonstrando todo o ciclo desde a pergunta inicial até a resposta final. Este exemplo serve como template para o modelo seguir.

A técnica de few-shot prompting é particularmente poderosa aqui. Ao fornecer um exemplo concreto de interação bem-sucedida, ensinamos ao modelo o padrão esperado sem necessidade de fine-tuning. Wei et al. [3] demonstraram que essa abordagem de "chain-of-thought prompting" melhora significativamente a capacidade de raciocínio dos LLMs.

Parte Prática

Arquitetura de Implementação

A implementação desenvolvida estrutura-se em três componentes principais:

Classe Agent: Esta classe gerencia todo o histórico conversacional e orquestra as chamadas à API do LLM. Ela mantém um array de mensagens que funciona como memória de curto prazo do agente. A cada nova interação, a mensagem do usuário é adicionada ao histórico, o modelo é consultado com todo o contexto acumulado, e a resposta é novamente armazenada. Este mecanismo garante que o agente tenha conhecimento de todas as etapas anteriores do raciocínio.

System Prompt: Como descrito anteriormente, é o conjunto de instruções que define o comportamento do agente. Ele é inserido como a primeira mensagem do histórico com role "system", estabelecendo as regras que o modelo deve seguir durante toda a sessão.

Tools (Ferramentas): São funções Python que o agente pode invocar. Na implementação básica da aula, duas ferramentas foram disponibilizadas: uma função de cálculo matemático e uma função que retorna a massa de planetas do sistema solar. Na minha extensão do projeto, implementei três ferramentas adicionais que consomem APIs REST externas: obtenção de informações geográficas de países, conversão de moedas e cálculo de diferença de fuso horário.

Loop de Controle: Orquestrando o Ciclo

O loop de controle é o mecanismo que automatiza o ciclo ReAct. Sua lógica funciona da seguinte forma:

Primeiro, o agente é inicializado com o system prompt e a lista de ferramentas disponíveis. A query inicial do usuário é fornecida como primeiro input.

Em cada iteração do loop, o agente é chamado com o prompt atual (inicialmente a query do usuário, depois as observações das ferramentas). O modelo retorna uma resposta textual que pode conter pensamentos, ações ou a resposta final.

O sistema verifica se a resposta contém as palavras-chave "PAUSE" e "Action". Se sim, isso indica que o agente deseja executar uma ferramenta. Um parsing com expressão regular (regex) extrai o nome da ferramenta e seus argumentos do texto gerado pelo modelo. A ferramenta correspondente é então executada, e seu resultado é formatado como uma observação ("Observation: resultado") que se torna o próximo input para o agente.

Se a resposta contém a palavra "Answer", o loop identifica que o agente chegou à conclusão e termina a execução, retornando a resposta final ao usuário.

Um contador de iterações máximas previne loops infinitos caso o modelo apresente comportamento errático e não consiga convergir para uma resposta.

Extensão Implementada: APIs REST

Para demonstrar a versatilidade do padrão ReAct, estendi a implementação básica com três ferramentas que consomem APIs REST públicas:

get_country_info: Consulta a REST Countries API para obter dados demográficos e geográficos de qualquer país (população, capital, área, moedas, idiomas, etc.).

convert_currency: Utiliza a Exchange Rate API para realizar conversões entre diferentes moedas em tempo real.

calculate_timezone_diff: Acessa a WorldTime API para calcular a diferença de fuso horário entre duas localizações.

Estas ferramentas demonstram como o agente pode ser capacitado a interagir com serviços externos, superando completamente a limitação de conhecimento estático dos LLMs. O agente pode, por exemplo, responder perguntas como "Qual a diferença de população entre Brasil e Alemanha em 2026?" buscando dados atualizados em tempo real.

Sistema de Logging

Para facilitar a análise e debugging do comportamento do agente, implementei um sistema de logging estruturado que registra cada iteração em arquivo de texto. O log captura:

- O número da iteração
- A resposta completa do modelo (incluindo pensamentos e ações)
- A ferramenta escolhida e seus argumentos
- O resultado da execução da ferramenta
- Eventuais erros encontrados

Este registro detalhado permite entender exatamente como o agente "pensou" para chegar à resposta, tornando o processo totalmente auditável.

Fluxo Completo da Implementação

Para consolidar o entendimento de todos os componentes e suas interações, apresento abaixo o diagrama completo da arquitetura implementada:

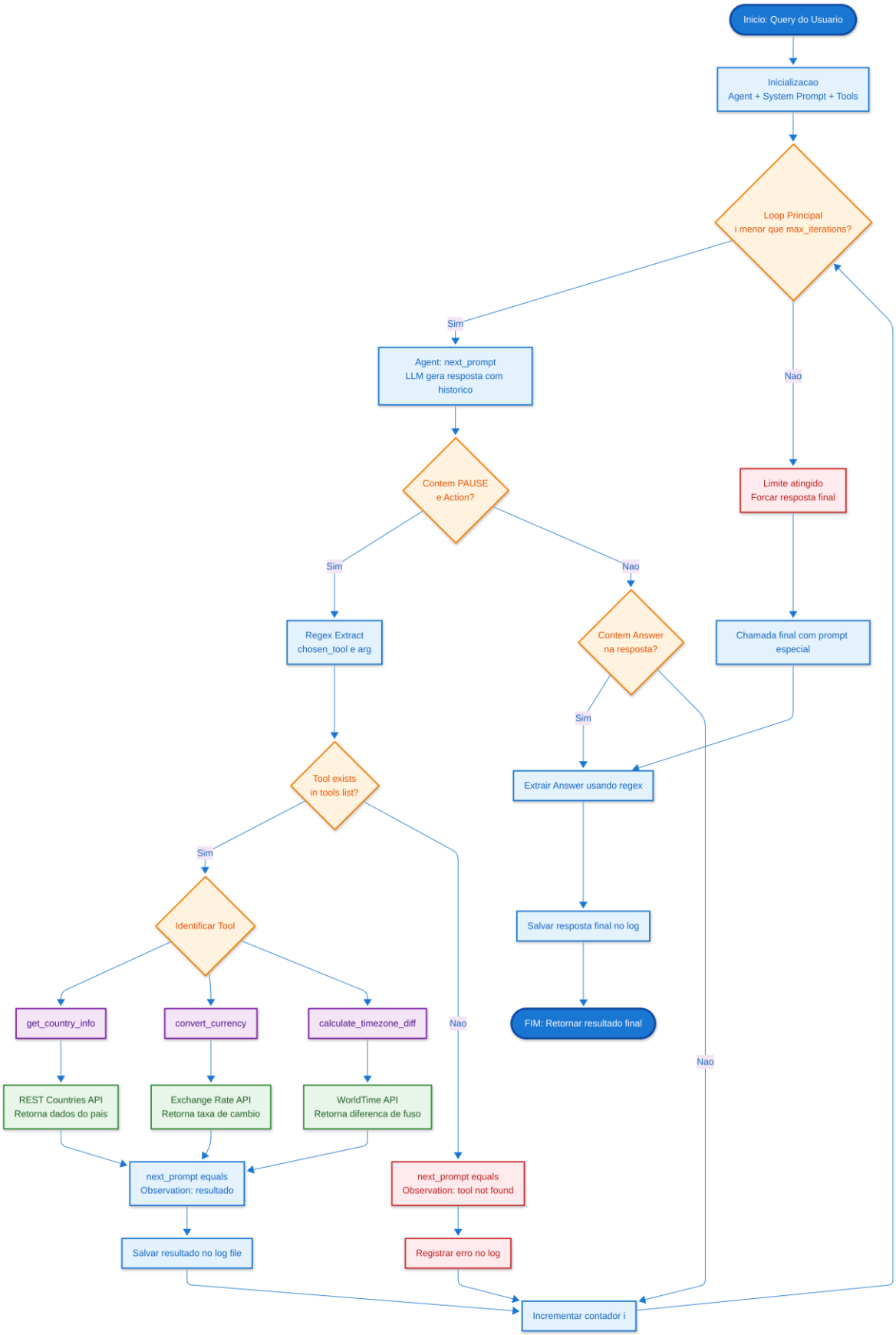

Figura 2: Diagrama de fluxo completo mostrando toda a arquitetura do sistema implementado no arquivo do aluno. Agente de tira dúvidas de países.

Exemplo de Execução Real

Para ilustrar o funcionamento prático, apresento um exemplo de execução do agente respondendo à pergunta: "Qual é a população do Brasil e como ela se compara à da França?"

Iteração 1:

O agente pensa que precisa primeiro obter informações sobre o Brasil. Escolhe a ferramenta get_country_info com argumento "Brazil" e emite PAUSE.

Iteração 2:

O sistema executa a ferramenta e retorna uma observação contendo os dados do Brasil (população: 214 milhões, capital: Brasília, etc.). O agente analisa essa observação e pensa que agora precisa dos dados da França. Escolhe novamente get_country_info, desta vez com argumento "France" e emite PAUSE.

Iteração 3:

O sistema retorna a observação com dados da França (população: 67 milhões). O agente verifica que agora possui ambas as informações necessárias e gera a resposta final: "O Brasil tem aproximadamente 214 milhões de habitantes, enquanto a França tem cerca de 67 milhões. A população do Brasil é aproximadamente 3,2 vezes maior que a da França."

Este exemplo demonstra como o agente decompõe autonomamente um problema complexo em subtarefas sequenciais, executando múltiplas consultas a ferramentas e compondo as informações obtidas em uma resposta coerente.

Dificuldades

1. **Limite de iterações e comportamento de força:** O limite de iterações, pois o consumo gratuito acaba rápido, para prevenir loops infinitos criou o problema de como proceder quando o limite era atingido antes da conclusão. Implementei um prompt de força na última iteração ("Output your Answer NOW with whatever information you have collected") garantindo que sempre haja resposta ao usuário, mesmo que incompleta.

2. **Parsing com Regex vs. Respostas estruturadas:** Expressões regulares para extrair informações do texto são inerentemente frágeis - pequenas variações na formatação ("Action : calculate" vs "Action: calculate") causam falhas. Identifiquei que abordagens mais robustas envolvem function calling nativo ou instrução para retorno em JSON estruturado, eliminando completamente a necessidade de regex.

Conclusões

A implementação de um agente ReAct do zero demonstrou que sistemas agenticos transcendem fundamentalmente as limitações dos LLMs isolados, combinando raciocínio estruturado com execução de ações no mundo real. O padrão ReAct provou-se uma arquitetura elegante que força o modelo a explicitar seu processo de pensamento através do ciclo Thought → Action → Observation, tornando o sistema auditável e melhorando a qualidade do raciocínio. A experiência validou que o system prompt não é mero texto de configuração, mas código propriamente dito, exigindo o mesmo rigor de versionamento e testes que código tradicional. A técnica de few-shot prompting mostrou-se mais eficaz que instruções abstratas, confirmando que LLMs aprendem melhor por demonstração que por descrição. Apesar da fragilidade inerente desses sistemas - exigindo múltiplas camadas de defesa como limites de iteração, validação de ferramentas e tratamento robusto de erros - a capacidade emergente de compor ferramentas criativamente revela o verdadeiro potencial dos agentes: não em ferramentas individuais, mas na orquestração inteligente de múltiplas capacidades para resolver problemas complexos. O futuro da IA aplicada será de sistemas agenticos que coordenam modelos de linguagem, APIs especializadas e bases de conhecimento, superando definitivamente o paradigma de modelos isolados.

Referencias

[1] ALEJANDRO AO. Python: Create a ReAct Agent from Scratch. YouTube, 2024. Disponível em: https://www.youtube.com/watch?v=hKVhRA9kfeM. Acesso em: 12 fev. 2026.