

## Agentes Autônomos – Análise de CSV

Nome do Grupo responsável pelo projeto: AgentCraft-CTBA

Integrantes do Grupo:

Nome	E-mail	Telefone
Nicholas Arand	<a href="mailto:nicholas.arand@inspiriabr.com">nicholas.arand@inspiriabr.com</a>	+5541984993036
Alessandro Silveira	<a href="mailto:alessandro.silveira@yahoo.com">alessandro.silveira@yahoo.com</a>	+15142477417.0
André Ricardo Spolti	<a href="mailto:andre@spolti.com.br">andre@spolti.com.br</a>	+554198383504

### A framework escolhida: LangChain/LangGraph

LangChain e LangGraph são frameworks projetados para o desenvolvimento de aplicações impulsionadas por LLMs. Embora trabalhem juntos e sejam parte do mesmo ecossistema, eles têm focos ligeiramente diferentes:

LangChain é um framework de código aberto que simplifica o desenvolvimento de aplicações usando LLMs. Ele fornece uma interface padronizada e modular para interagir com modelos de linguagem e outros componentes, como fontes de dados externas e memória. É ideal para construir aplicações LLM com fluxos de trabalho lineares e bem definidos, como chatbots simples, sistemas de perguntas e respostas e geradores de conteúdo.

O LangGraph é uma extensão do LangChain, construída para lidar com cenários mais complexos e dinâmicos, especialmente no desenvolvimento de agentes de IA com estado (state-machines). Ele permite que você defina interações de agente como um grafo com estado persistente. Pode ser a escolha preferida quando se precisa construir agentes de IA que exigem interações cíclicas, processos de tomada de decisão complexos, colaboração entre múltiplos agentes e gerenciamento robusto de estado.

### Como a solução foi estruturada:

Testamos dois “approaches” diferentes. Começamos com um agente relativamente simples e enxuto, contendo apenas um orquestrador e um executor de ferramentas, usando ferramentas simples (*@tools*) um agente do próprio framework (*create\_csv\_agent*). O agente recebe a pergunta do usuário, localiza o arquivo zip no diretório de execução, descompacta o arquivo em uma pasta, identifica os arquivos csv e os disponibiliza para um agente RAG csv que procura responder a pergunta da melhor maneira possível. A solução funciona a contento porém possui poucos recursos para aprimoramento, e melhorias. Por isso procurou-se explorar outra solução.

O segundo agente trabalha de forma iterativa com o usuário, como um chat e funciona com um fluxo mais elaborado o que lhe permite agir com maior flexibilidade e personalização. Os inputs do usuário passam sempre por um ‘node’ que identifica o

estado de execução da tarefa e decide, às vezes de forma programática, às vezes por meio de uma LLM, qual a próxima ação a ser tomada.

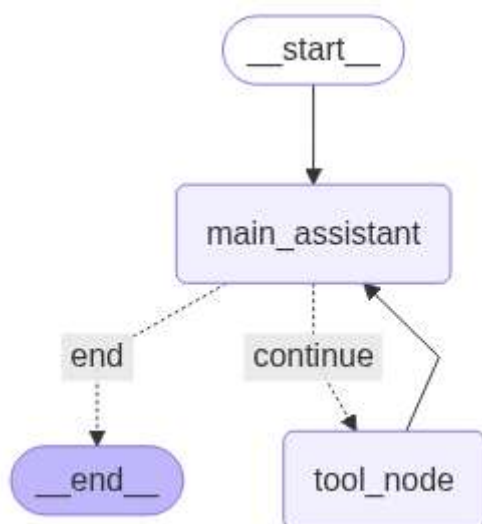
## Agente 1. LangGraphAgent: solução básica

Como apresentado na figura a seguir, são apenas dois elementos. O orquestrador (**main\_assistant**) recebe o input e decide como e quando usará as ferramentas. O executor de tarefas (**tool\_node**) realiza a tarefa com a ferramenta que o orquestrador lhe pede e com os argumentos fornecidos pelo mesmo.

Foram necessárias apenas três ferramentas:

1. Encontrar o nome do arquivo .zip na pasta do projeto (biblioteca **os**)
2. Descompactar o arquivo zip e encontrar o nome dos arquivos .csv (biblioteca **zipfile**)
3. Uma agente RAG do próprio LangChain (**langchain\_experimental.agents.agent\_toolkits.create\_csv\_agent**)

A seguir a estrutura criada pelo LangGraph:



## Agente 2. GraphRAGAgent: RAG GraphDB

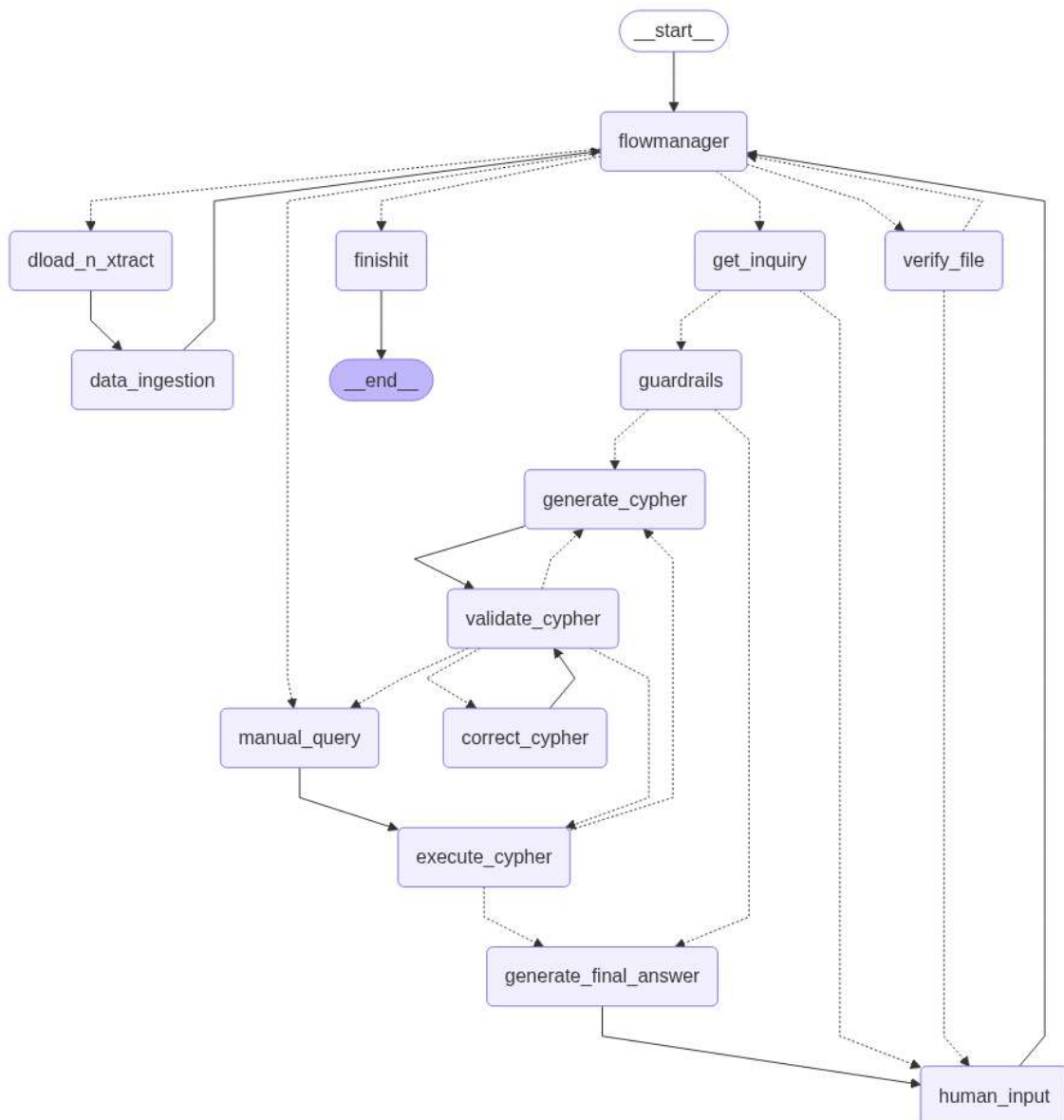
Como pode ser visto na figura a seguir, o agente é bastante mais complexo porém prevê maior grau de flexibilidade, personalização e melhoria de performance. Composto de um componente central que coordena a execução da tarefa (**flowmanager**) conforme interações com o usuário (**human\_input**) e com demais assistentes.

A primeira etapa do fluxo total é identificar o nome do arquivo zip (**verify\_file**) e busca-lo na pasta google drive do i2a2 para o download e a extração dos arquivos csv (**dload\_n\_xtract**). Feticia a extração o agente passa automaticamente para a etapa de ingestão dos dados em uma base Neo4j (**data\_ingestion**) quando cria também índices de busca textuais, por datas e também um índice semântico (vector embeddings) para os campos de produtos e serviços das notas fiscais.

Executadas estas tarefas de preparação o agente entra no ciclo de inquisição aos dados, interagindo com o usuário que realiza suas perguntas. Identificada uma pergunta ao usuário (**get\_inquiry**) o agente passa a pergunta por uma verificação de pertinência (**guardrails**) de onde pode ou recusar a resposta ou passar para a geração do código CYPHER que faz o *query* na base de dados.

O ciclo de geração (**generate\_cypher**), validação (**validate\_cypher**), correção (**correct\_cypher**) e execução (**execute\_cypher**) da consulta CYPHER ocorre automaticamente entre assistentes especialistas, até que uma consulta seja aprovada e a extração de dados seja passada ao assistente que gera a resposta final (**generate\_final\_answer**). No caso de três tentativas (configurável) de pesquisas CYPHER errôneas, o agente oferece ao usuário a opção de um CYPHER manual.

Com a resposta final voltamos à interação humana para a realização de mais perguntas, para se buscar um novo arquivo no drive da i2a2 ou terminando-se a execução do agente.



## Perguntas e respostas

Elaboramos 5 perguntas para as quais as respostas poderiam ser encontradas na base de dados com o objetivo de testar a capacidade de cada modelo encontrar as respostas corretas. Quadro abaixo podemos verificar os resultados;

PERGUNTA 01: Quantas notas fiscais estão na base de dados?

AGENTE 1 - There are 200 unique invoices in the database.

AGENTE 2 - Na base de dados, há um total de 100 notas fiscais.

RESPOSTA CORRETA: Temos 100 notas fiscais na base de dados

PERGUNTA 02: Qual o valor total das notas emitidas entre os dias 10 de Janeiro de 2024 e 20 de Janeiro de 2024

AGENTE 1 - 1,340,519.56

AGENTE 2 - O valor total das notas emitidas entre os dias 10 de Janeiro de 2024 e 20 de Janeiro de 2024 é de R\$ 1.340.519,56.

RESPOSTA CORRETA: Neste período, foram emitidas notas que somam o valor de R\$ 1,340,519.56

PERGUNTA 03: Qual a empresa que mais fez compras (recebeu notas) no estado do Paraná?

AGENTE 1 - According to the data provided, there are no recipients (companies) that received invoices in the state of Paraná.

AGENTE 2 - A empresa que mais fez compras (recebeu notas) no estado do Paraná é a UNIVERSIDADE FEDERAL DO PARANÁ, com um total faturado de R\$ 7.486,50.

RESPOSTA CORRETA: A empresa, no estado do Paraná, que mais recebeu emissão de notas foi a UNIVERSIDADE FEDERAL DO PARANÁ

PERGUNTA 04: Alguma empresa comprou alimentos?

AGENTE 1 - Sim, algumas empresas compraram alimentos. Identificamos a "FUNDACAO UNIVERSIDADE FEDERAL DE MS" e "SUPERINTENDENCIA REGIONAL SUDESTE I" comprando "AGUA MINERAL NATURAL".

AGENTE 2 - Sim, várias empresas compraram alimentos. As informações mostram que a UNIVERSIDADE FEDERAL DO RIO GRANDE NORTE comprou alho branco, o COLEGIO MILITAR DE PORTO ALEGRE adquiriu tomate salada, e o MINISTÉRIO DO DESENVOLVIMENTO E ASSISTÊNCIA SOCIAL, FAMÍLIA comprou quiabo e batata doce. Além disso, o MINISTÉRIO DO DESENVOLVIMENTO SOCIAL, FAMÍLIA E COMBATE À FOME também adquiriu batata doce.

RESPOSTA CORRETA: Sim. Algumas empresas compraram produtos alimentícios, como pão francês, tomate salada, quiabo e batata doce.

PERGUNTA 05: Qual o valor total das notas emitidas sobre materiais escolares?

AGENTE 1 - The total value of invoices emitted for school materials is 531.82.

AGENTE 2 - O valor total das notas emitidas sobre materiais escolares, com base nos resultados fornecidos, é de R\$ 118.000,00. Esse valor é a soma dos preços dos itens relacionados a materiais escolares, como livros e outros suprimentos.

RESPOSTA CORRETA: Parece que foram gastos cerca de R\$ 294,673.35 com materiais escolares

## Conclusões:

Obviamente que uma avaliação rigorosa poderia ser executada com mais questionamentos e, principalmente, diversas rodadas de respostas, visto que a natureza probabilística dos modelos faz com que suas respostas não sejam sempre as mesmas. Independentemente disso, vemos claramente que o segundo modelo, apesar de muito mais complexo e com muitas mais linhas de código que o primeiro (1.161 VS 265) é não apenas mais capaz de responder satisfatoriamente a perguntas complexas, mas também, no permitiria ajustes e aprimoramento detalhado de cada etapa do agente.

Enquanto o modelo LangGraph padrão pode nos fornecer uma solução rápida e barata para a construção de agentes que utilizam ferramentas e consultam bases de dados, o modelo personalizado nos permite, potencialmente, mergulhar muito mais fundo em estruturas de dados complexas (grafos), personalização total das tarefa e da interação com o usuário (human-in-the-loop) e, também potencialmente, ajustes finos de prompt engineering, das lógicas e métodos para a criação, verificação e correção dos queries cypher, e muito mais.

## Disponibilização dos arquivos

GitHub AgentCraft-CTBA:

<https://github.com/agentcraft-cwb/Atividade-Obrigatoria-2025-06-18>

Streamlit Agente 1:

<https://agente-langgraph-simples-nditxccivq-rj.a.run.app>

Streamlit Agente 2:

*Em breve... um pouco de dificuldade com os tempos de atualização do graph Neo4j Aura online*