Aira Chat: Sistema colaborativo de multi-agentes baseado em LLM para cadastro de informações em plataformas de gestão usando linguagem natural.

Thiago Cleto Miarelli Piedade[1] e orientado por Martín Gómez Ravetti[2]

***Abstract*— This work presents a collaborative multi-agent system called Aira Chat. It transforms natural language data from various sources into structured language, integrated with management systems, focusing on optimizing processes and reducing inconsistencies. Using LLM agents for interpretation and task execution, the proposed solution aims to automate common workflows in companies, such as contract management and billing.**

***Resumo*— Este trabalho apresenta um sistema colaborativo de multiagentes denominado Aira Chat. Ele transforma dados em linguagem natural de diferentes fontes em linguagem estruturada, integrada a sistemas de gestão, com foco em otimizar processos e reduzir inconsistências. Utilizando agentes LLM para interpretação e execução de tarefas, a solução proposta visa automatizar fluxos de trabalho comuns em empresas, como a gestão de contratos e cobranças.**

I. Introdução

Estima-se que a aplicação de inteligência artificial em processos produtivos adicione mais de 4,4 trilhões de dólares anualmente na economia mundial em ganhos de eficiência. A IA tem o potencial de transformar setores inteiros, melhorando a produtividade ao automatizar tarefas repetitivas, gerando insights mais profundos e acelerando a tomada de decisão.

Um dos cenários em que as aplicações de IA generativa têm se mostrado especialmente eficientes é na transformação de informações em diferentes formatos. A capacidade desses sistemas de capturar dados em múltiplas modalidades, como texto, imagens e planilhas, e de gerar conteúdo multi-modal, permite otimizar processos manuais como cadastro e reescrita. Esses avanços não apenas economizam tempo, mas também asseguram maior consistência nos dados processados.

Estudos indicam que erros humanos ao registrar dados de fontes variadas custam às empresas globalmente cerca de 3,3 trilhões de dólares por ano. A adoção de IA nesse contexto não só reduz drasticamente esses custos, mas também promove uma integração mais eficiente entre diferentes sistemas e plataformas, garantindo que informações críticas sejam tratadas de maneira uniforme e confiável. No projeto que desenvolveremos, otimizaremos o cadastro de informações no sistema Aira. A Aira é uma plataforma em nuvem especializada na gestão de contratos recorrentes, com um modelo de precificação baseado em volumetria, também conhecido como bilhetagem. Atualmente, regras de contrato e de precificação são frequentemente armazenadas em diversos formatos, como apresentações comerciais, contratos e planilhas de Excel, antes de serem integradas ao sistema.

Para resolver esse desafio, utilizaremos agentes baseados em modelos de linguagem. Esses agentes serão capazes de compreender o contexto dos documentos, verificar a existência de registros no sistema e, quando necessário, estruturar as informações de maneira apropriada para o cadastro via API na Aira. Essa abordagem reduzirá a complexidade do processo e garantirá maior consistência nos dados inseridos.

A. Agentes e Re-Act

Um agente é um sistema que utiliza uma LLM para tomar decisões e controlar o fluxo da aplicação. Em diversos cenários de uso, a realização de tarefas por mecanismos de inteligência artificial pressupõe a realização de passos antes e depois da execução da LLM. Por exemplo, em casos de RAG (Retrieval Augmented Generation), antes da geração da resposta pela LLM, é feita uma busca contextual em documentos relacionados que podem embasar a resposta do modelo. No caso de agentes que realizam inserção de dados em sistemas, é esperado que haja o uso de alguma ferramenta que chame a API do sistema e insira o dado após a chamada do modelo.

Em diversos casos, as tarefas performadas por modelos são tão complexas que desejamos que a própria LLM decida o fluxo de chamada de suas ferramentas. Além disso, é esperado que o agente, quando preciso, tenha autonomia de voltar-se a um humano para complemento de informações, confirmar ações e também se comportar como um ator especializado na tarefa, através de role-prompting.

Os agentes Re-Act (Reasoning and Acting) surgem como uma solução para integrar capacidades de raciocínio e ação em modelos de linguagem. Nele, o agente usa a LLM para entender o contexto, raciocinar, e se necessário, chamar uma ferramenta. O próprio agente, utilizando a LLM pode julgar a resposta da ferramenta ou chamá-la novamente, corrigindo qualquer erro que possa ter cometido anteriormente. Essa metodologia aumenta significativamente a eficácia de LLMs em cenários que demandam planejamento e execução integrados. [Figura 1]

B. Multi-agentes

O uso de agentes especializados para tarefas específicas trouxe avanços significativos na confiabilidade de execuções

[1]Graduando de Ciência da Computação na Universidade Federal de Minas Gerais.

[2]Professor do departamento de Ciência da Computação, UFMG.

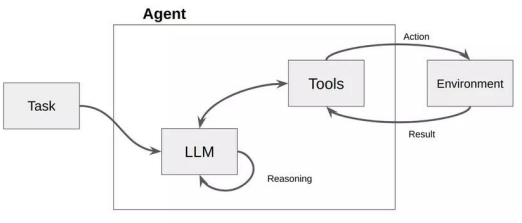

Fig. 1. Arquitetura de um agente Re-Act

pontuais. No entanto, em contextos mais amplos e complexos, a dependência de um único agente revelou-se ineficiente e limitante. Para superar essas limitações, surgiu a necessidade de sistemas de orquestração entre múltiplos agentes, onde cada agente pode interagir com ferramentas, envolver humanos no processo de raciocínio e aprovação e, se necessário, delegar tarefas a outros agentes mais especializados.

Essa evolução arquitetural foi crucial para habilitar sistemas autônomos mais robustos. O contexto de operação deixou de ser delimitado pelo modelo individual e passou a ser definido pela própria arquitetura do sistema. Isso permitiu que os modelos delegassem tarefas a agentes com maior domínio contextual, eliminando a necessidade de uma única instância carregar toda a complexidade do processo.

Entretanto, o desenvolvimento de sistemas multiagentes apresenta desafios únicos devido à sua natureza não determinística. Para enfrentar esses desafios, frameworks como Crew.ai e LangGraph foram projetados para introduzir uma camada de abstração sobre os modelos, fornecendo estruturas de controle robustas para o fluxo de execução. Esses frameworks também implementam guard-rails que ajudam a manter a consistência e a confiabilidade do sistema, mesmo em cenários mais complexos.

II. Metodologia e Implementação

A. Visão Geral

Neste projeto, implementaremos um sistema de orquestração de agentes baseados em dois subgrafos, gerenciados por um agente supervisor [Figura 2]. O fluxo começa com uma requisição do usuário, que fornece o contexto de sua tarefa. O agente supervisor analisa o pedido e pode encaminhá-lo para o agente responsável, solicitar informações adicionais ou indicar que o problema não pode ser resolvido no momento. O primeiro subgrafo é dedicado à criação de novos clientes. Inicialmente, o agente busca o CNPJ no texto fornecido como contexto. Esse agente é projetado para identificar com precisão as informações relacionadas ao cliente, diferenciando, por exemplo, entre dados da contratada e da contratante em um contrato. Uma vez encontrado o CNPJ, ele utiliza uma ferramenta integrada à API da Aira para verificar se o cliente já está cadastrado. Se o cliente existir, as informações são exibidas para o usuário. Caso contrário, o agente verifica se possui todas as informações necessárias para criar o cliente e, se houver dados faltantes, solicita essas informações ao usuário. Por fim, o agente realiza o cadastro através da API.

O segundo subgrafo trata da gestão de regras de planos. Assim como o agente de criação de clientes, o agente de planos analisa o contexto inicial e faz perguntas ao usuário enquanto não dispuser de todas as informações necessárias. Uma vez que o contexto esteja completo, o agente utiliza a ferramenta de API para criar os planos de forma automatizada.

B. Tecnologia e otimizações

Para o desenvolvimento do sistema de orquestração de multiagentes, foi utilizada a biblioteca LangGraph na linguagem Python, que tem como base tecnológica e conceitual o framework LangChain. Essa biblioteca organiza sistemas de multiagentes como grafos, utilizando diversos objetos e funções integradas do LangChain.

No LangGraph, um grafo representa o fluxo de trabalho de uma aplicação, sendo composto por nós (funções ou agentes que executam tarefas específicas) e arestas (conexões que determinam a ordem de execução e o fluxo de informações entre os nós). Essa estrutura permite uma representação clara e organizada das interações entre os agentes e seus processos.

A gestão das informações no grafo é feita através de um estado compartilhado. Esse estado é passado como entrada para os nós e atualizado com os resultados gerados, como mensagens da LLM ou saídas de ferramentas. Por exemplo, um nó que utiliza uma LLM pode adicionar novas mensagens ao estado, permitindo que outros agentes ou ferramentas acessem essas informações conforme necessário.

No projeto Aira Chat, foi implementada uma arquitetura de multi-grafos, onde os agentes interagem de forma independente em dois grafos distintos: o grafo de criação de clientes e o grafo de criação de planos. Essa separação garante que o raciocínio e as operações sejam limitados ao contexto relevante, evitando compartilhamento desnecessário de informações e resultando em uma significativa economia de tokens.

Uma estratégia adotada foi a modularização da entidade de chat, que passou a ser agnóstica ao modelo utilizado. Essa abordagem permitiu que diferentes agentes empregassem modelos variados, otimizados para suas respectivas tarefas. Essa flexibilidade facilitou a realização de experimentos rápidos e identificação de quais modelos apresentavam o melhor desempenho em diferentes cenários.

Na chamada de ferramentas, foi utilizado um agente LLM de formatação. Ele usava o contexto da conversa para formatar um JSON que seria utilizado pela API de criação. O agente recebe como parâmetros o estado do grafo e o schema do JSON e gera um output estruturado para ser consumido pela API.

Por fim, a interface foi criada utilizando a biblioteca Streamlit, com os componentes de chat, que permitiram com extrema facilidade criar uma UI amigável.

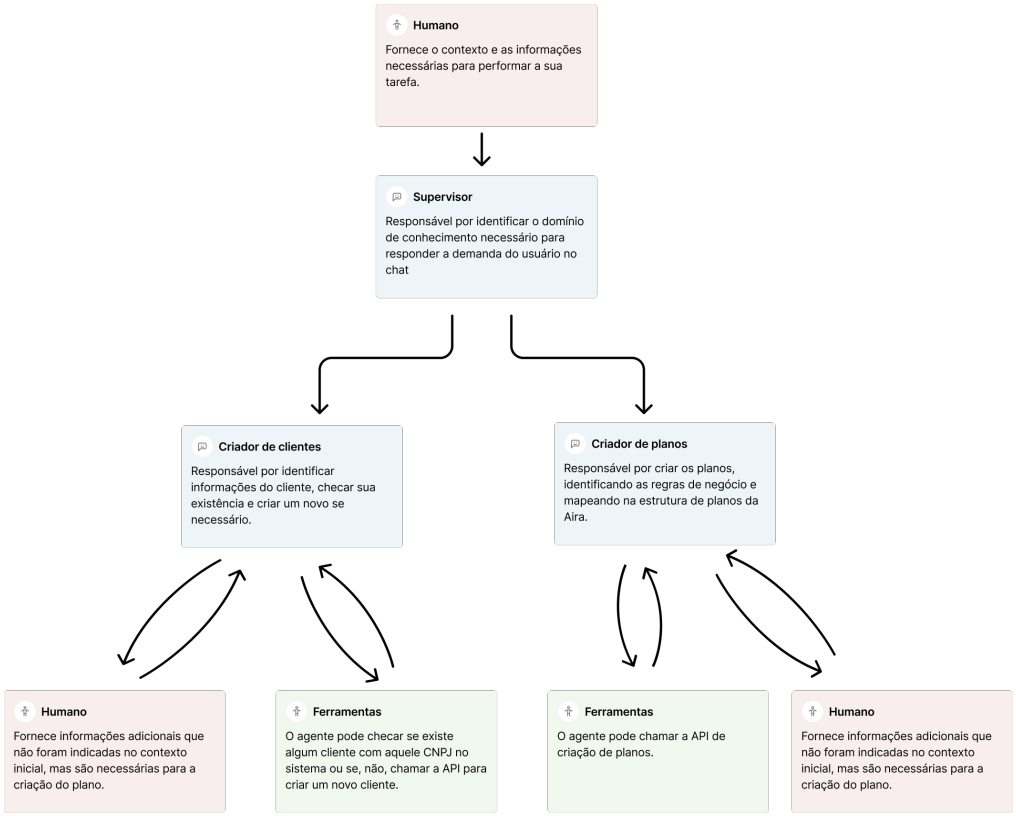

Fig. 2. Esquema simplificado do fluxo de uso dos agentes.

III. Resultados e melhorias

Após o desenvolvimento do projeto, realizamos alguns testes qualitativos para avaliar a performance dos sub-grafos de criação de clientes e planos. Os resultados foram analisados com base na precisão das respostas geradas pelos modelos e na habilidade de lidar com diferentes tipos de inputs.

Nos cenários de criação de clientes, os testes com o modelo GPT-4o apresentaram resultados altamente precisos. O agente foi capaz de identificar corretamente as informações em contratos e prompts, mesmo naqueles em que as informações se encontravam separadas em diversos pontos do documento.

Em contrapartida, ao utilizar o modelo GPT-4o mini, observou-se uma leve queda na precisão. Embora a maioria dos resultados fosse correta, ocorreram confusões pontuais entre os dados das partes envolvidas nos contratos, como a dificuldade em diferenciar informações do contratado e do contratante. Esse comportamento sugere limitações do modelo em relação à complexidade de interpretação em cenários mais detalhados, principalmente em textos maiores e menos diretos.

Já nos testes de geração de planos, a performance do subgrafo apresentou resultados mistos, mesmo utilizando o modelo GPT-4o. Em inputs bem descritos, envolvendo casos menos complexos – como faixas de preço de mesma modalidade, sem aplicação de preços mínimos ou franquias – o agente gerou respostas satisfatórias e consistentes.

Por outro lado, em cenários mais desafiadores, como planos definidos a partir de planilhas com preços mínimos aplicados apenas a parte dos recursos ou faixas de preço misturando quantias variáveis e fixas, o modelo frequentemente apresentou erros, evidenciando uma capacidade limitada para tarefas que exigem um raciocínio lógico e matemático.

Não foi possível testar os modelos com reasoning da OpenAI, como o GPT-o1 e o GPT-o1-mini, que possuem uma performance reconhecidamente superior em tarefas matemáticas e lógicas, como é o caso da criação de planos. Essa limitação decorre desses modelos não terem integração nativa com ferramentas externas.

No entanto, ao modificar ligeiramente a arquitetura do sistema, removendo as chamadas de API e retornando apenas o output estruturado, o modelo GPT-o1 demonstrou resultados excelentes. Ele foi capaz de gerar planos corretos mesmo nos cenários mais complexos, incluindo aqueles baseados em planilhas com regras de precificação mistas.

Um caso interessante ocorreu na interpretação de uma planilha com a fórmula =SE(A2>100; A2*1,1; 1000), onde o GPT-4o gerou uma regra de preço fixo de R$10 para valores abaixo de 100 unidades, ignorando a característica descrescente do preço, impossível de atingir na plataforma. O GPT-o1, por sua vez, retornou que a plataforma não suportava aquele tipo de plano.

Essa diferença ilustra a limitação de modelos de propósito geral em tarefas que exigem parsing matemático preciso, mesmo quando equipados com ferramentas de código.

Os testes e seus resultados qualitativos estão documentados na tabela 1.

A. Melhorias em futuras iterações

Para aprimorar a performance e a eficiência do sistema de multiagentes em futuras iterações, algumas estratégias podem ser implementadas de forma combinada. Primeiramente, o uso de RAG (Retrieval Augmented Generation) permitiria que os modelos de linguagem acessassem casos anteriores de criação de planos armazenados em bases de conhecimento, como um repositório vetorial. Esses exemplos seriam inseridos nos prompts dos LLMs para aumentar a precisão na interpretação de regras complexas e garantir consistência nas respostas, reduzindo a necessidade de ajustes frequentes nos modelos. No entanto, isso exigiria uma curadoria cuidadosa dos casos e monitoramento da relevância das buscas.

Em segundo lugar, a implementação de caching para prompts de instrução repetitivos (como diretrizes de formatação ou políticas fixas) poderia reduzir custos, já que prompts cacheados têm custo 50% menor em modelos como os da OpenAI. Isso seria útil para agentes com tarefas especializadas ou fluxos repetitivos.. A invalidação do cache ocorreria apenas quando houvesse atualizações nas políticas, usando versionamento para manter a integridade.

Por outro lado, a limpeza de contexto para remover mensagens desnecessárias — como erros de API não críticos ou interações repetidas — ajudaria a otimizar o uso de tokens.

Por fim, o streaming de caracteres na interface seria uma solução para diminuir a percepção de latência pelos usuários. Em vez de aguardar a geração completa da resposta, o sistema exibiria partes do conteúdo em tempo real. Isso melhoraria a experiência de uso, dando a sensação de agilidade, mesmo que o tempo total de processamento permaneça o mesmo. Frameworks como LangChain poderiam facilitar essa integração.

Cenário	GPT-4o	GPT-4o mini	GPT-o1
Criar clientes com informações bem estruturadas no prompt.	Satisfatório	Satisfatório	Não testado
Criar clientes com prompts onde as informações do cliente estão distribuídas no texto junto a outras informações.	Satisfatório	Boa, mas com confusões	Não testado
Criar clientes com prompts que misturam informações da contratante e da contratada	Satisfatório	Boa, mas com confusões	Não testado
Criar planos de complexidade baixa e média usando linguagem natural e descritiva	Satisfatório	Não testado	Satisfatório
Criar planos de complexidade baixa e média a partir de fórmulas de Excel	Boa, mas com confusões	Não testado	Satisfatório
Criar planos de complexidade alta usando linguagem natural e descritiva	Insuficiente	Não testado	Satisfatório
Criar planos de complexidade alta usando fórmulas de Excel	Insuficiente	Não testado	Satisfatório

TABLE I
TABELA DE PERFORMANCE QUALITATIVA POR MODELO

IV. CONCLUSÃO

Através do desenvolvimento do sistema Aira Chat, foi implementada uma solução colaborativa de multi-agentes baseada em LLM com o objetivo de automatizar o cadastro de informações em plataformas de gestão. O sistema adotou uma abordagem modular, utilizando dois subgrafos independentes, otimizando processos como a criação de clientes e a gestão de planos. Foram realizados testes qualitativos que evidenciaram o desempenho satisfatório em cenários de complexidade baixa e moderada, mas apontaram limitações em casos mais desafiadores, particularmente nas tarefas que envolvem lógica e cálculos complexos.

As análises dos resultados mostraram que, enquanto modelos como GPT-4o foram eficientes em tarefas estruturadas, os modelos mais avançados, como o GPT-o1, apresentaram maior precisão em cenários matemáticos e de raciocínio lógico, embora não tenham sido plenamente integrados às ferramentas externas. Identificou-se ainda o potencial para futuras melhorias, como a introdução de técnicas de RAG e caching, visando maior eficácia e redução de custos operacionais.

Os resultados evidenciam que sistemas multiagentes baseados em LLMs requerem uma arquitetura híbrida para escalabilidade. Tarefas de baixa complexidade (ex.: extração de CNPJ) podem ser delegadas a modelos menores, reduzindo custos operacionais, enquanto tarefas críticas (ex.: cálculo de preços) demandam modelos especializados. Futuros trabalhos explorarão a integração de modelos de domínio específico (ex.: finetuning em contratos de SaaS) e técnicas de validação formal via verificação simbólica para garantir correção matemática nas regras geradas.

REFERENCES

[1] McKinsey & Company, "The Economic Potential of Generative AI: The Next Productivity Frontier," McKinsey & Company, 2023. Available at: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontierkey-insights.

[2] SAP Community, "Bad Data Costs the U.S. $3 Trillion per Year," SAP Community, 2023. Available at: https://community.sap.com/t5/technology-blogs-by-sap/bad-data-costs-the-u-s-3-trillion-per-year/ba-p/13575387.

[3] LangChain, "Multi-Agent Systems," LangChain, 2023. Available at: https://langchain-ai.github.io/langgraph/concepts/multi$_agent/$.

[4] D. Allist, "Maximizing the Power of LLM by Using ReAct Agent," Medium, 2023. Available at: https://medium.com/@daneallist/maximizing-the-power-of-llm-by-using-react-agent-1bb6ebff2a3b.

[5] Amazon Web Services, "Unlocking Complex Problem-Solving with Multi-Agent Collaboration on Amazon Bedrock," AWS Blog, 2023. Available at: https://aws.amazon.com/blogs/machine-learning/unlocking-complex-problem-solving-with-multi-agent-collaboration-on-amazon-bedrock/.

[6] LangChain, "Human-in-the-Loop: Interrupt," LangChain, 2023. Available at: https://langchain-ai.github.io/langgraph/concepts/human$_in_the_loop/interrupt$.

[7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," *arXiv preprint arXiv:2210.03629*, 2022. Available at: https://arxiv.org/abs/2210.03629.

[8] E. Carter, "Why the ReAct Agent Matters: How AI Can Now Reason and Act," Wordware, 2024. Available at: https://www.wordware.ai/blog/why-the-react-agent-matters-how-ai-can-now-reason-and-act.