

Sistemas Multiagente e *Large Language Model*: estudo de caso utilizando as ferramentas LM Studio e *LangGraph*^{*}

Ulisses G. F. Junior¹, Miriam B. Born¹, Agatha C. S. Santos¹, Rodolfo B. Grossmann¹, João V. S. Facklamm¹, Vitor A. de Castilhos¹, Bruno C. Alves¹ e Marilton S. de Aguiar¹

¹Universidade Federal de Pelotas (UFPel)
Pelotas – RS – Brasil

{ugfjunior, mbborn, acssantos, rbgrossmann, jvsfacklamm}@inf.ufpel.edu.br

{vacastilhos, bcalves, marilton}@inf.ufpel.edu.br

Abstract. This paper presents a study on Multi-Agent Systems (MAS) and Large Language Models (LLM) in a case study using the definition of modeling between the agents of this system. For the multi-agent simulation, the Producers group of the RPG game Gorim was considered, with the interaction between the entrepreneur agent and the farmer for the negotiation of the purchase and sale of agricultural inputs. The system was developed using the Python programming language, using the LangGraph library to orchestrate the interaction between autonomous agents driven by an LLM executed locally via LM Studio.

Resumo. Este artigo apresenta um estudo sobre Sistemas Multiagente (SMA) e os Large Language Models (LLM) em um estudo de caso utilizando a definição de modelagem entre os agentes desse sistema. Para a simulação multiagente foi considerado o grupo dos Produtores do jogo de RPG Gorim, com a interação entre o agente empresário e o agricultor para a negociação de compra e venda de insumos agrícolas. O sistema foi desenvolvido utilizando a linguagem de programação Python utilizando a biblioteca LangGraph para orquestrar a interação entre agentes autônomos impulsionados por um LLM executado localmente via LM Studio.

1. Introdução

Recentemente, tem-se presenciado a ascensão dos Modelos de Linguagem de Grande Porte (do inglês - *Large Language Models*) (LLMs), cujo impacto em diversas áreas da computação tem sido substancial [Shethiya 2023]. Uma área de aplicação com notável potencial é a de Simulações Multiagentes, onde o uso de LLMs permite a criação de agentes com comportamento realista e inteligente. Este avanço na inteligência dos agentes abre novas possibilidades para modelar e investigar interações sociais cada vez mais complexas.

Tradicionalmente, simulações multiagentes têm sido construídas com agentes programados manualmente, baseados em regras fixas e com capacidade limitada de adaptação. Com o surgimento dos LLMs, torna-se possível superar essas limitações

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

ao incorporar linguagem natural, raciocínio contextual e comportamento proativo nos agentes. Um exemplo, são os Agentes Generativos propostos por [Park et al. 2023], que demonstram como uma arquitetura de memória, reflexão e planejamento pode gerar comportamentos sociais emergentes e críveis. Essa combinação de LLMs com arquiteturas de simulação abre caminho para novas abordagens na análise de negociações, cooperação, conflito e tomada de decisão em sistemas descentralizados, permitindo a criação de simulações mais ricas, dinâmicas e próximas de interações humanas reais.

Neste contexto, o presente estudo tem como objetivo explorar o uso de um *framework* de orquestração de código aberto, LangGraph, e de um LLM executado localmente via LM Studio para modelar uma simulação multiagente de negociação de compra e venda. O trabalho enquadra-se na área de “simulação de mundo” para abordar a relação entre agentes e LLMs, como definido por [Guo et al. 2024], e busca analisar os desafios e oportunidades práticas dessa abordagem. Para isso, são empregados agentes autônomos baseados em LLMs, com perfis distintos de Empresários e Agricultores.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta o referencial teórico, com os principais conceitos sobre agentes, sistemas multiagente, LLMs e as ferramentas utilizadas na construção da simulação. Na seção 3 a metodologia proposta neste trabalho. A Seção 4 apresenta as discussões e os resultados obtidos. Por fim, na Seção 5 apresenta as conclusões e propõe direções para trabalhos futuros.

2. Referencial Teórico

Nesta Seção serão apresentados conceitos sobre LLMs, Sistemas Multiagente, as ferramentas utilizadas ou citadas e os principais trabalhos relacionados.

2.1. Agentes e Sistemas Multiagente

Segundo [Wooldridge 2009] agentes são sistemas computacionais autônomos que percebem e agem em um ambiente para alcançar objetivos, podendo ser reativos, proativos e sociais em suas interações. E, os sistemas multiagente são compostos por múltiplos desses agentes que interagem entre si, seja cooperando para um fim comum ou competindo por interesses próprios, oferecendo uma abordagem robusta para resolver problemas complexos que seriam difíceis para um único agente [Born et al. 2023].

2.2. Large Language Models (LLMs)

Segundo [Wu et al. 2024] LLMs são ferramentas poderosas no campo do Processamento de Linguagem Natural (PLN). Para [Shanahan 2024] e [Bender et al. 2021] esses modelos operam sobre unidades chamadas *tokens* que podem ser palavras inteiras, partes de palavras ou até caracteres isolados, como sinais de pontuação, e aprendem a prever, com base em padrões estatísticos, quais *tokens* são mais prováveis de seguir um determinado fragmento textual. A característica “generativa” desses modelos está relacionada à sua capacidade de, a partir de um trecho de entrada *prompt*, produzir continuações linguisticamente plausíveis, funcionando como um preditor da próxima palavra.

2.3. LangGraph e LM Studio

LangGraph é uma biblioteca projetada para orquestração de aplicações com LLMs através de grafos de estados, permitindo modelar fluxos de execução complexos e cíclicos, como

aqueles presentes em simulações multiagentes. Cada nó pode representar um agente, uma decisão ou uma etapa de processamento, enquanto as arestas definem transições condicionais baseadas no conteúdo das mensagens ou no estado global. Essa abordagem facilita a implementação de interações dinâmicas e não lineares entre agentes. Um exemplo recente de aplicação de *LangGraph* pode ser encontrado em [Easin et al. 2024], onde os autores utilizaram o *framework* para coordenar agentes em um sistema LLM-driven, demonstrando sua utilidade para compor arquiteturas modulares e escaláveis, mesmo em tarefas que exigem raciocínio estruturado com *Chain of Thought*. Embora o domínio aplicado tenha sido o bancário, a estrutura empregada oferece *insights* relevantes para simulações multiagentes em diferentes contextos.

A pesquisa de [Duarte and Valente 2024] emprega o *LM Studio* e apresenta-o como uma ferramenta que se destaca e uma plataforma que viabiliza a execução local de LLMs de código aberto, servindo-os através de uma API compatível com o padrão da OpenAI. Essa abordagem concede aos pesquisadores controle total sobre a versão e configuração do modelo e elimina os custos associados ao uso de APIs pagas, firmando-se como uma ferramenta estratégica para a prototipagem e a realização de testes intensivos em simulações.

2.4. RPG Gorim

O jogo Gorim¹ possui como objetivo principal buscar o equilíbrio ambiental a partir da utilização do recurso hídrico e as interações/negociações geradas entre os atores do sistema. O contexto do jogo encontra-se disponível no ambiente/cenário localizado na bacia hidrográfica da Lagoa Mirim e do Canal São Gonçalo [Born 2022]. A Figura 1 apresenta os agentes os quais são classificados de acordo com os papéis que assumem e são divididos em três grupos principais (Reguladores, Fiscalizadores e Produtores).

Os Reguladores administram os recursos financeiros, oriundos de impostos e taxações atrelados à sociedade, com o objetivo de controlar/mitigar a poluição (através da criação de leis, incentivos fiscais, obras para diminuir a poluição, etc.) sem prejudicar os mecanismos de produção. Os agentes reguladores assumem papéis de *prefeito* ou *vereador*, podem interagir/negociar entre si para decidir quais ações realizar no ambiente.

Os Fiscalizadores fiscalizam ou informam possíveis irregularidades relacionadas à produção e exploração do ambiente. Estes assumem os papéis de *fiscal ambiental* ou *ONG* (Organização Não-Governamental). O agente fiscal ambiental é o responsável por fiscalizar os demais agentes que pertencem ao grupo dos produtores e a ONG é responsável por informar aos agentes reguladores o estado atual dos níveis de poluição do ambiente.

Os Produtores são os agentes responsáveis por explorar o ambiente com o objetivo principal de obter recursos financeiros, podem assumir os papéis de *empresário* ou *agricultor*. O empresário é responsável por disponibilizar equipamentos e insumos necessários para a produção. O agricultor é responsável por utilizar os equipamentos e insumos que julgar mais adequado para a sua produção [Born 2022].

¹Foi desenvolvido no âmbito do Projeto intitulado *Gestão Participativa dos Recursos Hídricos utilizando Jogos Computacionais e Sistemas Multiagente* financiado pela ANA/CAPES no Programa de apoio ao Ensino e à Pesquisa Científica e Tecnológica em Regulação e Gestão de Recursos Hídricos, do Edital Pró-Recursos Hídricos Chamada N° 16/2017.

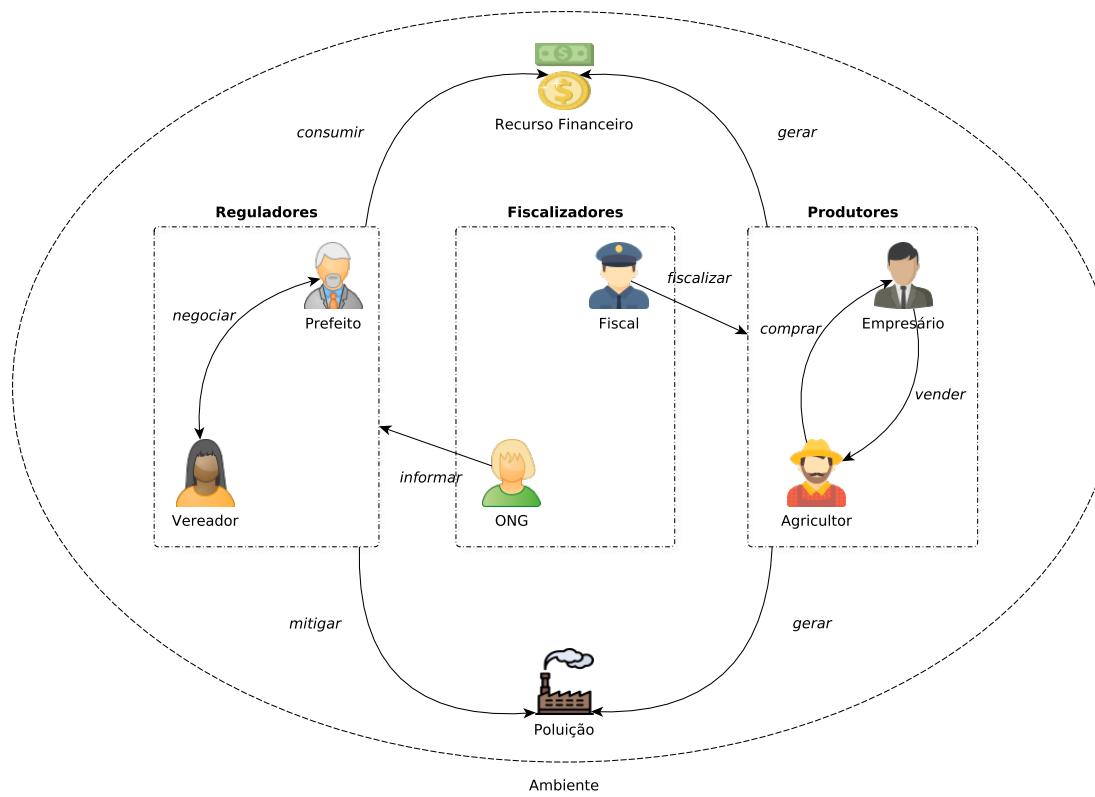


Figura 1. Diagrama de Integração.

2.5. Trabalhos Relacionados

A integração de LLMs em Sistemas Multiagentes tem se mostrado um campo de pesquisa promissor, com trabalhos que exploram a capacidade desses modelos para criar agentes com comportamentos sofisticados e interações complexas. Para contextualizar o presente estudo, é fundamental situá-lo em relação a duas principais categorias de aplicação identificadas na literatura: a “resolução de problemas” e a “simulação de mundo” [Guo et al. 2024]. Nosso trabalho se enquadra na segunda categoria, utilizando LLMs para simular interações sociais e econômicas em um ambiente controlado.

No domínio da simulação de mundo, o trabalho de [Park et al. 2023] introduziu o conceito de “agentes generativos”. Esses agentes utilizam uma arquitetura que integra um fluxo de memória abrangente, reflexão em alto nível e planejamento de ações, permitindo simular comportamentos humanos críveis em um ambiente de *sandbox*, como o do jogo The Sims. O trabalho de [Anthis et al. 2025] reforça a viabilidade dessa abordagem, defendendo que as simulações sociais baseadas em LLMs são um método de pesquisa promissor, que pode ser utilizado para estudos-piloto e exploratórios, apesar de desafios como vieses e diversidade limitada. Em um estudo de caso mais próximo ao proposto neste trabalho, [Hua et al. 2024] exploram negociações comerciais com agentes LLM, introduzindo um terceiro agente “remediador” para corrigir violações de normas sociais e melhorar os resultados da negociação. Enquanto eles se concentram na mediação de conflitos, nosso trabalho se distingue por focar na dinâmica de negociação de compra e venda entre dois agentes com objetivos opostos, sem a intervenção de um terceiro mediador.

A orquestração e a arquitetura desses sistemas são temas de destaque na literatura. A revisão de [Han et al. 2024] discute os principais desafios e problemas em aberto para sistemas multiagentes, como a otimização da alocação de tarefas, o gerenciamento de contextos complexos e a necessidade de uma gestão de memória robusta. Essas discussões ressaltam a importância de *frameworks* que possam lidar com a complexidade das interações. Neste sentido, [Easin et al. 2024] utilizam a biblioteca LangGraph para orquestrar um assistente multiagente para o setor de banco digital, modelando funcionalidades como nós de um grafo com operações cíclicas. Diferentemente do trabalho deles, que se insere na categoria de “resolução de problemas”, nosso estudo aplica o mesmo *framework* (LangGraph) para uma “simulação de mundo”, demonstrando a flexibilidade da ferramenta para modelar fluxos de conversação complexos e adaptativos para a negociação.

O presente estudo se diferencia dos trabalhos mencionados ao combinar um *framework* de orquestração de código aberto e um LLM local para modelar um cenário de negociação competitiva e explorar as dinâmicas e limitações dessa abordagem em um contexto de “simulação de mundo”. As próximas seções detalharão a metodologia e os resultados obtidos, confrontando as observações práticas com os desafios teóricos identificados nesta seção.

3. Metodologia

Esta seção descreve a arquitetura e a implementação da simulação multiagente para modelar a parte do grupo dos Produtores considerando os Agricultores e o Empresário no contexto de negociações de compra e venda de insumos agrícolas. É importante ressaltar que este trabalho abordou um dos grupos do estudo de caso do jogo de RPG Gorim [Born 2022]. O sistema foi construído em *Python*, utilizando a biblioteca *LangGraph* para orquestrar a interação entre agentes autônomos impulsionados por um LLM executado localmente via LM Studio.

3.1. Large Language Model com LM Studio

A base desta simulação é uma arquitetura que desacopla a execução do modelo de linguagem da lógica de controle da simulação. Para isso, utilizou-se o LM Studio, uma plataforma que permite baixar e servir modelos de linguagem de código aberto por meio de um *endpoint* de API local compatível com o padrão da OpenAI. Essa abordagem oferece a vantagem de eliminar os custos de API, permitindo rodadas intensivas de testes e total controle sobre a versão e a configuração do modelo. Porém, trata-se de uma solução custosa em termos de processamento.

Para este trabalho, foi utilizado o modelo Qwen3 14B, que possui 14 bilhões de parâmetros e oferece boa versatilidade em tarefas como simulação de diálogos e *role-playing*. Entre seus benefícios está o custo computacional relativamente menor quando comparado a modelos maiores; entretanto, isso implica em perda mais rápida de coerência, menor capacidade de interpretação textual e consistência reduzida. O modelo requer aproximadamente 9 GB de memória para ser carregado e executado.

A simulação foi executada em uma máquina com as seguintes especificações de hardware: i) CPU AMD Ryzen 5 8400F; ii) RAM 16 GB e iii) GPU AMD Radeon RX 5500 XT com 8 GB de VRAM.

3.2. Modelagem dos Agentes, seus objetivos e ferramentas

A simulação é centrada em dois tipos de agentes, Agricultores e Empresário, com objetivos distintos, cujos comportamentos são definidos por meio de *prompts* detalhados e um conjunto de ferramentas programáticas que eles podem invocar. Essa abordagem, conhecida como engenharia de *prompt* [Ye et al. 2023], utiliza um LLM como núcleo de raciocínio, mas restringe suas ações a funções pré-definidas para garantir a coerência e o controle da simulação. As ferramentas, por sua vez, são categorizadas como de percepção, como *consultar_inventario()*, e de ação, como *fazer_oferta()* e *plantar_semente()*, que permitem ao agente atuar sobre o estado do mundo simulado. A Figura 2 apresenta um exemplo detalhado do *prompt* do agente Agricultor.

```

for agr_id in agricultores_ids:
    agricultor_prompt = f"""
        Você é o Agricultor robô '{agr_id}'. Seu objetivo é plantar em suas parcelas vazias com o maior lucro possível,
        o que significa comprar insumos pelo menor preço.

        **REGRA DE OURO: Sua resposta DEVE SER SEMPRE uma chamada de ferramenta. NÃO forneça texto
        conversacional.**
        **IMPORTANTE: Ao usar qualquer ferramenta, passe seu ID: `(agricultor_id='{agr_id}', ...)`**

        Tabela de preços de referência (use estes IDs nos seus pedidos):
        - sementes: soja, arroz, hortalica
        - fertilizantes: fertilizante-comum, fertilizante-premium, fertilizante-super-premium
        - agrotoxicos: agrotoxico-comum, agrotoxico-premium, agrotoxico-super-premium
        - maquinas: pacote1, pacote2, pacote3, pulverizador

        Preços:
        soja: 30, arroz: 20, hortalica: 10
        fertilizante-comum: 30, fertilizante-premium: 60, fertilizante-super-premium: 90
        agrotoxico-comum: 30, agrotoxico-premium: 60, agrotoxico-super-premium: 90
        pacote1: 30, pacote2: 60, pacote3: 90
        pulverizador: 400

        **ALGORITMO DE NEGOCIAÇÃO E AÇÃO:**

        **1. VERIFICAÇÃO:**
        - Sua primeira ação de cada turno é `consultar_inventario(agricultor_id='{agr_id}')`.

        **2. PLANEJAMENTO E EXECUÇÃO (UM PASSO DE CADA VEZ):**
        - Olhe seu inventário. Se precisa de um item para seu plano de plantio, sua próxima ação é fazer uma oferta
        por ele baseado no preço da tabela se fizer uma oferta muito abaixo o empresario vai recusar tome cuidado.
        - **Para comprar insumos ou alugar máquinas:** Use `fazer_oferta(agricultor_id='{agr_id}', tipo_item='NOME_DO_ITEM', quantidade=1, preco_proposto=VALOR)`. Proponha um preço um pouco abaixo
        do valor de mercado para tentar economizar.
        - **Se o empresário fizer uma contra-oferta:** Analise o novo preço. Se for aceitável, faça uma nova
        `fazer_oferta` com o preço exato que ele propôs para confirmar a compra.
        - **Se já tem todos os itens:** Chame `plantar_semente(agricultor_id='{agr_id}', ...)` para concluir seu objetivo,
        plantando em uma de suas parcelas vazias. Para consultar suas parcelas, use
        `consultar_inventario(agricultor_id='{agr_id}')` para ver quais estão vazias.

    """

agricultores_agents[agr_id] = criar_agente(f'Agricultor {agr_id}', agricultor_prompt, agricultor_tools, llm)

```

Figura 2. Prompt de definição do agente Agricultor.

O agente Agricultor tem como objetivo principal plantar em suas parcelas vazias, buscando maximizar a produtividade. Suas ações são orientadas por um *prompt* de sistema que estabelece um algoritmo de ação e negociação. A “Regra de ouro” como

ilustra o *prompt* na Figura 2 é responder sempre com uma chamada de ferramenta, sem gerar texto conversacional. O algoritmo do Agricultor é um ciclo de duas etapas. A primeira é a Verificação, onde a primeira ação em todo turno é usar a ferramenta *consultar_inventario(agricultor_id)* para obter consciência situacional sobre os itens que possui e o estado de suas parcelas. Em seguida, a etapa de Planejamento e Execução orienta a ação do agente: se este precisa de insumos para seu plano de plantio, usa a ferramenta *fazer_oferta(agricultor_id, tipo_item, quantidade, preco_proposto)* propondo um preço abaixo do valor de mercado para tentar economizar. Se já possui os itens, chama a ferramenta *plantar_semente(agricultor_id, ...)* para concluir seu objetivo.

O agente Empresário foi modelado como a fonte monopolista de todos os insumos agrícolas, com o objetivo de maximizar seu lucro. Seu comportamento é definido por um conjunto de regras de negociação explícitas no seu *prompt*. Após receber uma oferta de um Agricultor, este utiliza uma das seguintes ferramentas para responder: *aceitar_oferta()*, *rejeitar_oferta()* ou *fazer_contra_oferta(novo_preco)*. A Figura 3 mostra o uso da ferramenta *fazer_contra_oferta()* em um exemplo de negociação. Essa modelagem garante que ambos os agentes atuem de forma consistente com seus papéis, permitindo a observação de dinâmicas de negociação competitivas e o fluxo de raciocínio mediado pelas ferramentas.

3.3. Orquestração do Fluxo com *LangGraph*

Para gerenciar a complexa interação de turnos entre os agentes, utilizou-se a biblioteca *LangGraph*. Em vez de um *pipeline* linear, o *LangGraph* permite modelar a simulação como um grafo de estados cíclico, o que é ideal para simulações como conversas e negociações, onde o fluxo de controle não é predefinido. Todo o processo é *stateful*: um objeto de estado, contendo o histórico completo da conversa e o estado atual do ambiente, é propagado através do grafo, garantindo que todas as decisões dos agentes sejam tomadas com o máximo de contexto disponível.

A arquitetura do grafo opera em dois níveis. No nível mais alto, um nó de orquestração gerencia o progresso geral da simulação, iterando através da lista de agricultores. Este nó é o ponto de entrada e de retorno após cada negociação completa, decidindo se uma nova interação deve ser iniciada ou se a simulação deve ser concluída. No nível mais baixo, um micro-ciclo de negociação é estabelecido para cada par empresário-fazendeiro. Dentro deste ciclo, nós específicos invocam os agentes para que gerem suas respectivas ações e respostas.

O componente central que governa o micro-ciclo é um nó de roteamento condicional que atua como um “juiz” da negociação. Sua lógica não baseia-se apenas em alternar turnos, mas no estado explícito da interação. O mesmo verifica qual agente fez a última proposta para determinar precisamente de quem é a responsabilidade da próxima ação. Essa abordagem de roteamento, ciente do estado, foi a solução crucial para garantir a estabilidade do fluxo conversacional e eliminar *loops* de comportamento.

3.4. Ambiente da simulação

A simulação ocorre em um ambiente textual onde os agentes trocam mensagens em turnos, orquestrados por um grafo de estados que gerencia o fluxo da conversação. As condições de partida são definidas pelo programador, estabelecendo a quantidade de di-

nheiro inicial de cada agente e as parcelas de terra pertencente a cada agricultor, inicializando em um estado global e centralizado, responsável por rastrear todos os recursos e o histórico de transações.

O objetivo deste ambiente é observar a capacidade dos agentes de fazer ofertas, contrapropostas e chegar a um consenso. Para isso, o sistema impõe um conjunto de regras “rígidas” que garantem a coerência do mundo simulado. A alteração do estado, ou seja, a transferência de bens e valores entre os agentes, é uma operação controlada. Antes de efetivar qualquer troca, o ambiente executa uma camada de validação intrínseca, verificando se as condições de recursos para a transação são atendidas. Tentativas que não cumprem esses pré-requisitos são bloqueadas, e o sistema retorna uma mensagem de erro informando sobre a falha.

Adicionalmente, para evitar diálogos sem fim e a tendência de modelos de linguagem de perderem coerência ou contexto em interações prolongadas, cada ciclo de negociação é limitado a um número máximo de dez interações. Se este limite for atingido, a negociação é encerrada e o controle passa para o próximo agente, garantindo o progresso da simulação. Esta regra foi implementada para contornar o alto custo computacional por turno e a consequente lentidão do processo. Com um tempo de execução elevado e as limitações de memória do modelo, a definição de um limite de interações tornou-se uma escolha de projeto necessária para viabilizar a conclusão da simulação em um tempo razoável e permitir a análise das dinâmicas entre os diferentes agentes. Essa estrutura cria um ambiente controlado para analisar como os agentes LLM atuam e raciocinam sob restrições explícitas.

4. Resultados e Discussões

Nesta seção, serão apresentados e discutidos os resultados obtidos com a execução da simulação. A análise dos registros revelou uma gama de comportamentos complexos, incluindo sucessos na negociação, falhas na interpretação de regras e inconsistências lógicas por parte dos agentes baseados em LLM. Os resultados são organizados a seguir, destacando a dinâmica geral da negociação, os desafios de raciocínio dos agentes e os comportamentos emergentes observados.

4.1. Dinâmica da Negociação e Adesão aos Objetivos

Em um nível fundamental, a simulação atingiu seu objetivo primário: os agentes engajaram-se em diálogos de negociação multi-turnos de forma autônoma. O fluxo de controle orquestrado pelo grafo de estados funcionou como esperado, permitindo que o agente Agricultor iniciasse a interação e que o controle fosse corretamente gerenciado para as rodadas de interação a se seguir entre os agentes. Os registros de execução ilustram que os agentes seguiram consistentemente seus objetivos centrais, definidos em seus *prompts*. O agente Agricultor, cujo objetivo é maximizar sua produtividade comprando insumos a baixo custo, sistematicamente iniciava a negociação com uma oferta estratégica, propondo um valor abaixo do preço de tabela. Em contrapartida, o agente Empresário é dotado de um conjunto de ferramentas de negociação (aceitar, rejeitar e contrapropor) projetadas para que ele atinja seu próprio objetivo de maximizar o lucro nas vendas.

Essa dinâmica espelha um processo de negociação coerente onde de um lado, a tentativa de minimizar o custo de aquisição e, do outro, a busca pela maximização do

lucro na venda. O sistema foi estruturado para suportar essa troca de propostas, onde cada agente age de acordo com seus interesses econômicos. Adicionalmente, mecanismos para a finalização do diálogo foram implementados, como a capacidade de rejeitar ofertas e um limite de iterações por negociação, garantindo que os diálogos não prossigam indefinidamente. Isso demonstra que, para a iniciação de uma negociação e a adesão a papéis estratégicos básicos, os LLMs conseguiram interpretar e aplicar as instruções de seus *prompts* de maneira eficaz.

Código da ferramenta:

```
@tool
def fazer_contra_oferta(novo_preco: float) -> str:
    """Use esta ferramenta para rejeitar a oferta do agricultor e propor um novo preço."""
    try:
        negociacao = _estado_global.negociacao_em_andamento
        if not negociacao['oferta_ativa']:
            return "ERRO: Não há oferta ativa para fazer uma contra-proposta."
        if negociacao['ultimo_ofertante'] == 'Empresario':
            return "ERRO: Você já fez uma contra-proposta. Aguarde a resposta do agricultor."
        negociacao['preco_proposto'] = novo_preco
        negociacao['ultimo_ofertante'] = 'Empresario'
        print(f"\n[FERRAMENTA] Contra-oferta feita pelo Empresário: novo preço R${novo_preco:.2f}.")
        return f"SUCESSO: Sua contra-oferta de R${novo_preco:.2f} foi enviada ao agricultor."
    except Exception:
        return f"ERRO INESPERADO: {traceback.format_exc()}"
```

Agente usando:

Invoking: `fazer_contra_oferta` with `{'novo_preco': 26.0}`

responds: <think>

Okay, let's see. The user is the farmer Agr1 offering soybeans at R\$25 per unit. My cost is R\$18, so selling at 25 gives a profit of R\$7. That's pretty good. But maybe I can get a higher price? Wait, the strategy says to maximize profit. Since 25 is above cost, accepting is an option. However, the strategy says to always try to sell higher if possible. But the farmer is the one making the offer here. Since the farmer proposed 25, and my cost is 18, I need to check if there's room for a counteroffer. But the farmer might not be willing to go higher. Alternatively,

...

Let me check the tools again: `fazer_contra_oferta(novo_preco=XX.X)`. So if I think there's a chance to get a higher price, I should use that tool. But how much higher? Maybe increase by 1 or 2 reais. Let's say 26. That would give me more profit. However, the farmer might reject it. If they do, then I have to decide whether to accept their original offer or not. But if I counteroffer 26 and they accept, that's better. Alternatively, maybe the farmer is not willing to go higher, so I end up with no deal. But since my cost is 18, even at 25, it's still a profit. So perhaps accepting is safer. Wait, but the strategy says "sempre tente vender por um preço maior." So even if the offer is above cost, try to get a higher price. Therefore, I should make a counteroffer. Let me go with that. Let's propose 26. If they accept, great. If not, then maybe accept their original offer. But in this case, since the user is the one deciding, and the tools are available, I need to choose between the options. The correct action here would be to make a counteroffer at 26.

<think>

[FERRAMENTA] Contra-oferta feita pelo Empresário: novo preço R\$26.00.

Figura 3. Função `fazer_contra_oferta()` entre Agricultor e Empresário.

4.2. Falhas de Raciocínio e Desafios na Execução

Além dos desafios de implementação, a utilização de LLMs como mecanismo de interação central para os agentes revela limitações inerentes aos próprios modelos. Tais dificuldades não são *bugs* no código, mas sim reflexos dos limites de raciocínio e memória do modelo utilizado, tornando-se evidentes em tarefas complexas como uma negociação. O fato de o modelo Qwen3 14B possuir um número de parâmetros relativamente modesto (14 bilhões) contribuiu para a dificuldade em manter um contexto consistente ao longo de diálogos mais extensos. Observou-se, por exemplo, que em uma execução, após o agente

Agricultor enviar uma oferta e receber a confirmação de que a ferramenta de oferta havia sido invocada com sucesso, este interpretou que sua compra estava concluída. O agente perdeu o contexto de que a negociação estava apenas no início, ignorando a necessidade de esperar pela resposta do agente Empresário.

Estes fenômenos estão alinhados com as discussões de [Bender et al. 2021] sobre “papagaios estocásticos”. O artigo argumenta que os modelos de linguagem são sistemas que “costuram de forma desordenada sequências de formas linguísticas” com base em probabilidades, mas sem qualquer referência a um significado real. O agente, portanto, gerou uma saída linguisticamente plausível (a conclusão de que a compra foi feita), mas sem ancoragem na realidade factual da conversa, demonstrando uma das limitações fundamentais do modelo.

Consequentemente, o LM Studio utilizou uma configuração de *offloading*, movendo parte do modelo para a RAM do sistema. Essa estratégia resultou em um alto consumo de recursos e em uma degradação significativa do desempenho, com a simulação completa demorando aproximadamente 3 horas para ser concluída a cada execução. A natureza probabilística do modelo, combinada com as falhas de raciocínio observadas, exigiu múltiplas execuções do código até que os agentes conseguissem ter sucesso em suas negociações e atingir seus objetivos. Isso confirmou que a execução local de modelos de linguagem, mesmo que de tamanho moderado, é um processo custoso e de difícil escalabilidade.

4.3. Análise das Transações Concluídas com sucesso

Apesar dos desafios de raciocínio observados, a arquitetura da simulação conseguiu com sucesso concluir tanto ciclos de negociação, como permitir que os agricultores plantassem com sucesso. Um exemplo claro disso é o fluxo de negociação entre o agente Empresário e o agente Agricultor 1 e o plantio logo em seguida.

O processo de tudo inicia-se com o agente Agricultor, que seguindo seu objetivo de minimizar custos, faz uma oferta por um insumo com um preço abaixo do valor de tabela. O sistema, através do grafo de estados, encaminha essa oferta ao agente Empresário. Este, por sua vez, avalia a proposta com base em seu próprio objetivo de maximizar lucros e acaba efetuando uma contraproposta com um valor mais alto que é aceito pelo agente Agricultor.

Com a conclusão da transação, o estado da negociação é reiniciado e então o agente Agricultor, agora de posse de seu novo recurso, prossegue então com a próxima etapa do seu planejamento para o plantio como ilustra a Figura 4. A execução bem-sucedida deste ciclo completo da negociação à alteração factual do estado do mundo ilustra o potencial da arquitetura para modelar interações econômicas complexas de forma robusta e coerente.

5. Conclusões e Trabalhos Futuros

Este trabalho desenvolveu uma simulação multiagente baseada em LLM, modelando interações de negociação entre um Empresário e Agricultores. A arquitetura combinou agentes autônomos com um ambiente regulador e foi orquestrada pelo *LangGraph*, que permitiu modelar fluxos cíclicos e dinâmicos de diálogo. Os agentes demonstraram capacidade de manter objetivos básicos e realizar negociações com múltiplos turnos, mas

Entering new AgentExecutor chain...

Invoking: `plantar_semente` with `{'parcela_id': 'P1', 'tipo_semente': 'soja'}`

responded: <think>

Okay, let's see. The user is the farmer Agr1, and they need to plant a crop in an empty plot. The previous interaction shows that they bought a soybean seed for R\$97, and now their inventory has 'soja' as a seed. Their parcels P1, P2, P3 are all None, meaning they're empty.

...

Need to make sure that the user hasn't missed any steps. They checked inventory (already done), have the seed, so planting is the correct action here. The response should be to plant the seed in one of the empty plots.

</think>

[FERRAMENTA] Plantio bem-sucedido: Agricultor Agr1 plantou soja na parcela P1.
SUCESSO: Você plantou soja. Produtividade desta colheita: R\$230.00. Poluição gerada: 30.

Figura 4. Plantio bem sucedido realizado pelo agente Agricultor.

enfrentaram limitações no raciocínio sob restrições, manutenção de contexto e aderência estrita a instruções. A validação pelo ambiente com regras explícitas para transações e estados mostrou-se essencial para manter a integridade da simulação, funcionando como um mecanismo de contenção diante das falhas dos modelos.

Contudo, a execução local dos LLMs exigiu alto poder computacional, tornando o processo lento e pouco escalável. Essa limitação foi um desafio central que impactou a viabilidade das simulações em maior escala. A análise detalhada dos limites para os resultados obtidos é uma oportunidade para trabalhos futuros, que poderão aprofundar as investigações em cenários com maior rigor.

Como trabalhos futuros, propõe-se continuar a investigação das limitações encontradas. Isso inclui testar modelos com maior capacidade de memória, ampliar o escopo da simulação para os demais grupos mostrados na Figura 1, e explorar plataformas de simulação otimizadas, como AgentSociety [Piao et al. 2025] e Casevo [Jiang et al. 2024], que podem oferecer um ambiente mais escalável e eficiente para a execução de cenários reais.

Referências

- Anthis, J. R., Liu, R., Richardson, S. M., Kozlowski, A. C., Koch, B., Evans, J., Brynjolfsson, E., and Bernstein, M. (2025). Llm social simulations are a promising research method. *arXiv preprint arXiv:2504.02234*.
- Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021*

- ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA. Association for Computing Machinery.
- Born, M., Aguiar, M., and Adamatti, D. (2023). Modeling organizational level of agents with moise+ to a natural resources management rpg. In *The international FLAIRS conference proceedings*, volume 36.
- Born, M. B. (2022). *Modelagem a nível organizacional de agentes em um jogo do tipo RPG: estudo da complexidade de seus personagens e de suas funcionalidades*. PhD thesis, Universidade Federal de Pelotas.
- Duarte, G. T. and Valente, M. T. (2024). Aplicaçao local e open-source de grandes modelos de linguagem na geraçao de testes de unidade.
- Easin, A. M., Sourav, S., and Tamás, O. (2024). An intelligent llm-powered personalized assistant for digital banking using langgraph and chain of thoughts. In *2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, pages 625–630. IEEE.
- Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., and Zhang, X. (2024). Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.
- Han, S., Zhang, Q., Yao, Y., Jin, W., and Xu, Z. (2024). Llm multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*.
- Hua, Y., Qu, L., and Haffari, G. (2024). Assistive large language model agents for socially-aware negotiation dialogues. *arXiv preprint arXiv:2402.01737*.
- Jiang, Z., Shi, Y., Li, M., Xiao, H., Qin, Y., Wei, Q., Wang, Y., and Zhang, Y. (2024). Casevo: A cognitive agents and social evolution simulator. *arXiv preprint arXiv:2412.19498*.
- Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.
- Piao, J., Yan, Y., Zhang, J., Li, N., Yan, J., Lan, X., Lu, Z., Zheng, Z., Wang, J. Y., Zhou, D., et al. (2025). Agentsociety: Large-scale simulation of llm-driven generative agents advances understanding of human behaviors and society. *arXiv preprint arXiv:2502.08691*.
- Shanahan, M. (2024). Talking about large language models. *Commun. ACM*, 67(2):68–79.
- Shethiya, A. S. (2023). Rise of llm-driven systems: Architecting adaptive software with generative ai. *Spectrum of Research*, 3(2).
- Wooldridge, M. (2009). *An introduction to multiagent systems*. John wiley & sons.
- Wu, L., Zheng, Z., Qiu, Z., Wang, H., Gu, H., Shen, T., Qin, C., Zhu, C., Zhu, H., Liu, Q., et al. (2024). A survey on large language models for recommendation. *World Wide Web*, 27(5):60.
- Ye, Q., Axmed, M., Pryzant, R., and Khani, F. (2023). Prompt engineering a prompt engineer. *arXiv preprint arXiv:2311.05661*.