COGNILENS: UMA PLATAFORMA PARA COORDENAÇÃO E ANÁLISE AUTOMATIZADA DE TAREFAS

MORAES, Marcos Roberto de
Faculdade Santa Lúcia
marcos.moraes6@fatec.sp.gov.br

MACEDO, Rita de Cássia Catini
Fatec Mogi Mirim – Arthur de Azevedo
rita.catini@fatec.sp.gov.br

MACEDO, Paulo César de
Faculdade Santa Lúcia
paulo.macedo@fatec.sp.gov.br

RESUMO

As ferramentas de IA surgiram com o objetivo principal de resolver problemas do cotidiano de forma mais assertiva e rápida, simulando a inteligência humana. Sabe-se que essas ferramentas utilizam também da linguagem natural, ou seja, de nossa forma de comunicação para seu funcionamento. O desenvolvimento de sistemas que integram modelos de linguagem natural à resolução de problemas complexos tem ganhado espaço em diversas áreas. Este artigo apresenta a criação do CogniLens, uma plataforma baseada no modelo da empresa já conhecida Google, chamado Gemini, para coordenação e análise de tarefas com múltiplos agentes. O sistema permite aos usuários configurar tarefas, definir agentes com diferentes especialidades e gerar relatórios detalhados, promovendo maior eficiência em processos colaborativos. O artigo discute o desenvolvimento, funcionalidades, arquitetura e potenciais impactos do CogniLens, além de explorar oportunidades de aprimoramento, bem como um conjunto de testes da aplicação. Como resultado, pode-se observar a simplificação dos

Recebido em 26-02-2025 • Aceito em 08-05-2025

processos, a coordenação dos agentes de forma dinâmica e responsiva, insights *relevantes em um modelo adaptável e com um bom nível de detalhamento.*

PALAVRAS-CHAVE: *coordenação de tarefas; inteligência artificial; modelo de linguagem; Gemini; automação.*

INTRODUÇÃO

Nos dias atuais sabe-se que os modelos de linguagem natural (*Natural Language Processing* - NLP) são sistemas baseados em inteligência artificial que entendem, geram e interagem com textos de forma semelhante aos humanos.

Existem inúmeras aplicações que podem se pautar nesse tipo de tecnologia, que ajuda e impacta a vida das pessoas oferecendo as mais variadas soluções, tais como traduções automáticas, resumo de textos, análise de sentimentos e assistentes virtuais. As tecnologias mais relevantes e famosas em NLP incluem modelos, bibliotecas e plataformas usadas em diversas aplicações, pode citar inúmeras delas, destacando as mais famosas e relevantes temos: GPT 3, 4 – Criado pela OpenAI, BERT – Desenvolvido pelo *Google,* LLaMA – Criado pela Meta (*Facebook*), Claude – Desenvolvido pela *Anthropic*, entre outras. Para esse artigo pretende-se explorar um modelo de inteligência artificial desenvolvida pelo *Google DeepMind* para Processamento de Linguagem Natural (NLP), considerando sua facilidade e disponibilidade.

A evolução de modelos de linguagem natural, como o Gemini, tem permitido avanços significativos na automação de processos complexos e na solução de problemas que exigem alta capacidade analítica (Brown *et al*., 2020; Yang *et al.*, 2021). Apesar dos avanços em modelos de linguagem, ainda há uma lacuna na automação da coordenação de tarefas complexas que exigem a colaboração de diversos especialistas. O CogniLens visa colaborar com o preenchimento dessa lacuna, oferecendo uma solução para gerenciar e otimizar fluxos de trabalho que envolvem múltiplos agentes. Desde o surgimento de modelos como o *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin *et al.*, 2019), *Robustly Optimized BERT Approach* (RoBERTa) (Liu *et al*., 2019) e XLNet (Yang *et al.,* 2019), houve uma evolução notável em termos de capacidade de compreensão contextual e de geração de texto. Esses modelos baseiam-se em arquiteturas de transformadores que revolucionaram o processamento de linguagem natural (Vaswani *et al*., 2017).

Recebido em 26-02-2025 • Aceito em 08-05-2025

O modelo BERT, desenvolvido pela equipe do *Google*, foi um dos pioneiros na utilização de aprendizado bidirecional, permitindo que a compreensão contextual de palavras em uma frase fosse significativamente melhorada (Devlin *et al*., 2019). Posteriormente, o RoBERTa introduziu refinamentos no treinamento do BERT, como a remoção de mascaramento dinâmico e maior volume de dados, o que resultou em melhorias expressivas na precisão em diversas tarefas de linguagem (Liu *et al.,* 2019). Outro avanço relevante foi o XLNet, que combinou elementos autoregressivos e de aprendizado bidirecional, abordando limitações encontradas no BERT e oferecendo maior versatilidade em tarefas de entendimento e geração de texto (Yang *et al*., 2019). Mais recentemente, modelos como o Gemini destacaram-se por sua capacidade de integração em sistemas colaborativos, como o CogniLens, aproveitando algoritmos otimizados para coordenação de tarefas complexas (Brown *et al*., 2020). Enquanto modelos como o GPT-3 oferecem um grande potencial, eles frequentemente apresentam desafios em termos de custo computacional e interpretação de resultados em contextos práticos. O CogniLens, por outro lado, foi desenvolvido nesse artigo com foco na eficiência e na praticidade, integrando-se a fluxos de trabalho já existentes e oferecendo uma solução otimizada para tarefas complexas, como a automação de processos de análise de dados e planejamento estratégico.

Os transformadores, a base de todos esses avanços, foram introduzidos por Vaswani *et al.* (2017) e representam uma abordagem que elimina a necessidade de redes recorrentes, substituindo-as por mecanismos de atenção auto-regressivos. Esse paradigma não apenas melhorou a eficiência computacional, mas também possibilitou uma compreensão contextual mais profunda, alavancando avanços em tarefas como tradução automática e resposta a perguntas (Vaswani *et al*., 2017). Adicionalmente, avanços recentes em modelos de grande escala, como o GPT-3, trouxeram à tona discussões sobre a aplicabilidade e limitações desses sistemas em termos de custo computacional e interpretação de resultados (Brown *et al*., 2020). O Gemini, por sua vez, foi projetado para otimizar essas demandas, integrando-se a fluxos de trabalho práticos, como o do CogniLens.

O CogniLens é uma plataforma desenvolvida com o objetivo de aproveitar esses avanços para orquestrar agentes em tarefas colaborativas, proporcionando uma interface intuitiva para a definição de responsabilidades e habilidades dos agentes. Diferentemente de abordagens anteriores, que dependiam de regras fixas ou programas mais restritivos, o CogniLens explora as capacidades dinâmicas e adaptativas do modelo Gemini. Este artigo discute os fundamentos e as motivações por trás da criação do CogniLens,

Recebido em 26-02-2025 • Aceito em 08-05-2025

destacando sua arquitetura e funcionalidade. Sendo assim, espera-se que esse trabalho seja relevante para profissionais da área de tecnologia que trabalham com automação de processos, inteligência artificial, e desenvolvimento de sistemas colaborativos.

2. FUNDAMENTAÇÃO TEÓRICA

A evolução dos modelos de linguagem natural desenvolvidos pelo *Google* tem sido marcada por avanços significativos que culminaram no lançamento do Gemini 2.0. Inicialmente, o BERT foi introduzido como um modelo pré-treinado capaz de compreender o contexto de palavras em ambas as direções de uma sentença, revolucionando tarefas de processamento de linguagem natural (Devlin *et al*., 2019).

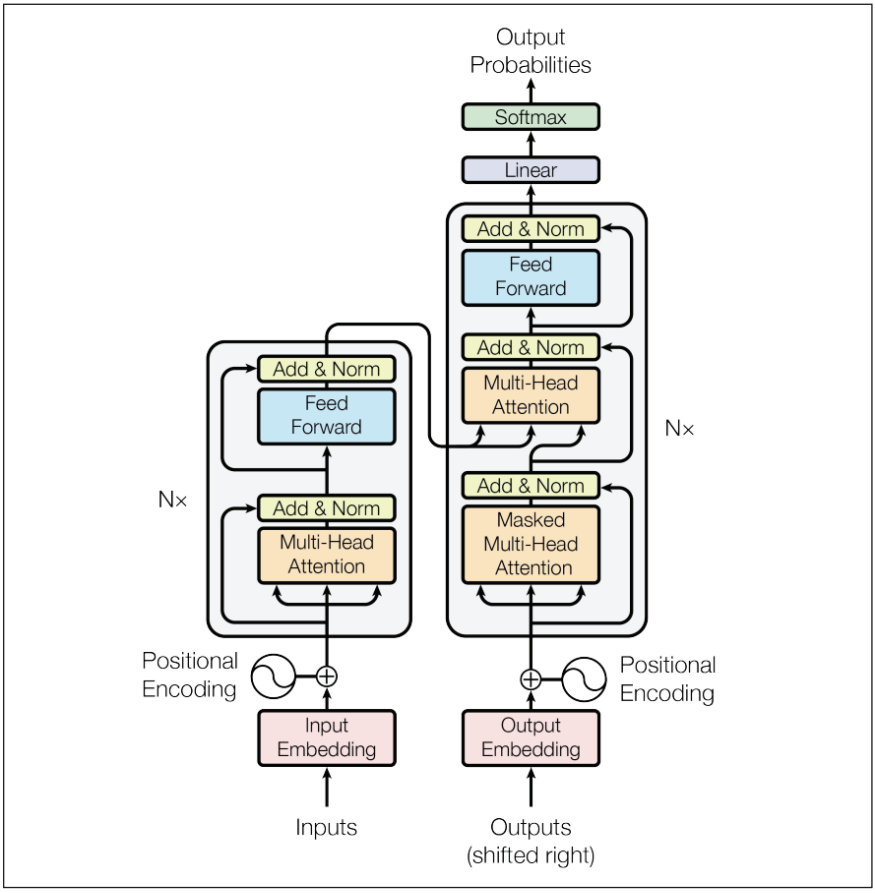
Posteriormente, o *Language Model for Dialogue Applications* (LaMDA) foi desenvolvido para aprimorar a capacidade de manter diálogos mais naturais e contextualmente relevantes, atendendo às necessidades de aplicações interativas (*Google,* 2024). Na sequência, o *Pathways Language Model* (PaLM) trouxe avanços ao integrar capacidade de processamento multilinguístico e lidar com tarefas complexas de maneira mais eficiente.

Esses avanços baseiam-se principalmente nas arquiteturas de transformadores introduzidas por Vaswani *et al.* (2017), que revolucionaram o processamento de linguagem ao utilizar mecanismos de atenção autorregressiva em vez de redes recorrentes. Essa abordagem não apenas melhorou a eficiência computacional, mas também possibilitou uma compreensão contextual mais profunda, alavancando tarefas como tradução automática e resposta a perguntas. A **Figura 1** - Modelo de Arquitetura - *Transformer* apresenta a arquitetura do modelo *Transformer*, introduzido por Vaswani *et al*. (2017) no artigo *Attention Is All You Need*. O modelo *Transformer* utiliza o mecanismo de atenção para processar entradas e gerar saídas de forma eficiente e paralela. A estrutura é composta por dois componentes principais: o codificador (à esquerda) e o decodificador (à direita). O codificador (à esquerda) converte a entrada em vetores de *embeddings*, que são combinados com codificações posicionais para incluir informações sobre a ordem das palavras, sendo composto por camadas repetidas (N vezes), cada uma integrada por um mecanismo de atenção *multi-head*, que permite ao modelo focar em diferentes partes da sequência de entrada, além de uma etapa de adição de conexões residuais e normalização de camada (*Add & Norm*) para melhorar a estabilidade do treinamento, seguida de uma rede totalmente conectada (*Feed Forward*) que processa os vetores de atenção.

Recebido em 26-02-2025 • Aceito em 08-05-2025

O decodificador (à direita) realiza um processo similar, onde as saídas são convertidas em *embeddings* e combinadas com codificações posicionais, utilizando atenção mascarada (*Masked Multi-Head Attention*) para garantir que apenas *tokens* anteriores sejam considerados na sequência de saída, preservando a causalidade, e atenção *multi-head* para integrar informações do codificador às saídas. Adicionalmente, possui camadas de *Feed Forward e Add & Norm*, semelhantes às do codificador. Por fim, as probabilidades de saída são obtidas ao processar a saída do decodificador por uma camada totalmente conectada (Linear) e pela função Softmax, que gera probabilidades sobre o vocabulário, determinando os próximos *tokens*.

Figura 1 - Modelo de Arquitetura – Transformer.



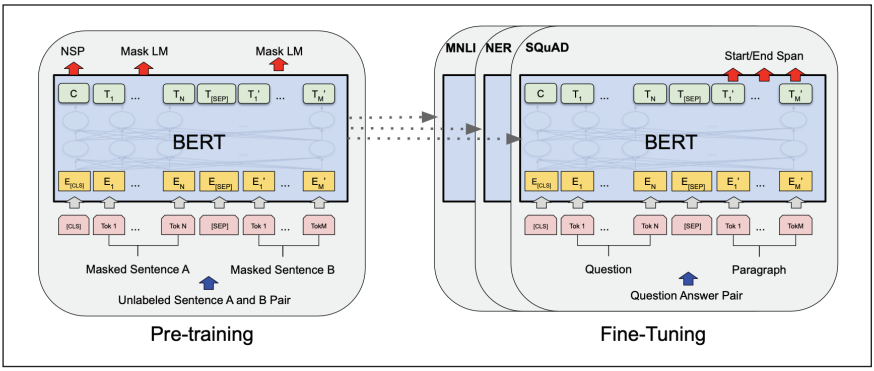
Fonte: Vaswani *et al.* (2017, p. 3).

Recebido em 26-02-2025 • Aceito em 08-05-2025

Já o modelo BERT trouxe impacto significativo ao introduzir aprendizado bidirecional para capturar o contexto de palavras em ambas as direções de uma frase (Devlin *et al*., 2019). Esse mecanismo foi um divisor de águas, permitindo avanços em tarefas como resposta a perguntas e reconhecimento de entidades nomeadas. Posteriormente, o RoBERTa refinou o BERT ao eliminar o mascaramento dinâmico e treinar o modelo em um conjunto de dados muito maior e por mais tempo, resultando em um aumento significativo no desempenho em *benchmarks* padronizados (Liu *et al*., 2021).

A **Figura 2** - Arquitetura do BERT: pré-treinamento com MLM e NSP e ajuste fino para tarefas específicas de NLP ilustra o processo de pré-treinamento e ajuste fino (*fine-tuning*) do modelo BERT, destacando sua arquitetura compartilhada para ambas as etapas. Durante o pré-treinamento, o modelo recebe pares de sentenças não rotuladas, compostas pelos *tokens* [CLS] (representando a sequência como um todo), [SEP] (separador entre sentenças), e os tokens das sentenças A e B, sendo treinado para duas tarefas principais: *Masked Language Modeling* (MLM), onde alguns tokens são mascarados e o modelo prevê seus valores com base no contexto bidirecional, e *Next Sentence Prediction* (NSP), que avalia se a segunda sentença é sequencialmente coerente com a primeira. No ajuste fino, o modelo pré-treinado é adaptado para tarefas específicas, como classificação de sentenças (usando o vetor [CLS]), reconhecimento de entidades nomeadas (NER) ou perguntas e respostas (SQuAD), ajustando as camadas de saída para prever spans, classes ou entidades, mantendo a arquitetura subjacente inalterada, o que destaca a flexibilidade e eficiência do BERT em diferentes aplicações de NLP.

Figura 2 - Arquitetura do BERT: pré-treinamento com MLM e NSP e ajuste fino para tarefas específicas de NLP.



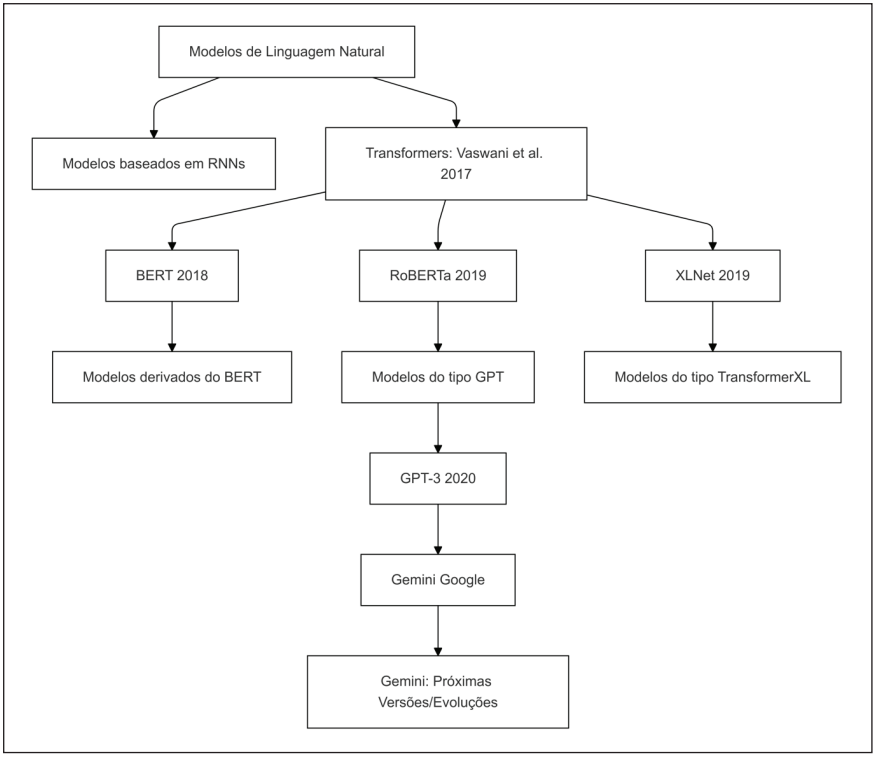
Fonte: Devlin *et al.* (2019, p.3).

Recebido em 26-02-2025 • Aceito em 08-05-2025

O XLNet combinou as vantagens do aprendizado autorregressivo com o bidirecional, utilizando uma abordagem de permutação de palavras durante o treinamento. Essa técnica permitiu que o modelo superasse algumas limitações do BERT, como a dificuldade em prever a probabilidade conjunta de sequências (Yang *et al*., 2021). Por outro lado, o GPT-3 destacou-se pela geração de texto coesa e pela capacidade de realizar tarefas complexas com pouca ou nenhuma instrução prévia, embora enfrentasse desafios em termos de custo computacional (Brown *et al*., 2020).

O Gemini 2.0 emergiu como um modelo multimodal de grande escala, integrando processamento de texto, imagens e áudio. Ele foi projetado especificamente para a era dos agentes de IA, com foco na capacidade de executar tarefas de forma autônoma e interagir de maneira eficiente com os usuários, como demonstrado na plataforma CogniLens (*Google*, 2024).

Figura 3 - Diagrama hierárquico que organiza a evolução e as categorias de Modelos de Linguagem Natural (NLP).



Fonte: Baseado em Devlin *et al.* (2019).

Recebido em 26-02-2025 • Aceito em 08-05-2025

A **Figura 3** apresenta um diagrama hierárquico que organiza a evolução e as categorias de Modelos de Linguagem Natural (NLP), destacando a progressão cronológica e as ramificações desses modelos. Ela evidencia como as inovações ao longo do tempo se integraram e se aprimoraram mutuamente, impulsionando significativamente o avanço no campo da Inteligência Artificial (IA).

3. METODOLOGIA

Esta seção descreve as etapas do desenvolvimento, os testes realizados bem como os métodos empregados.

O desenvolvimento do CogniLens baseia-se em uma arquitetura projetada para integrar modelos de linguagem avançados, como o Gemini, com ferramentas de automação para coordenação de tarefas complexas. Esta seção explora a arquitetura do sistema, suas funcionalidades principais e os métodos utilizados para sua implementação. Ao combinar uma interface intuitiva, suportada pelo *framework* Flask, e um núcleo analítico robusto, o CogniLens oferece uma solução escalável e eficiente para cenários colaborativos que demandam alta capacidade analítica e flexibilidade.

3.1 Arquitetura do Sistema

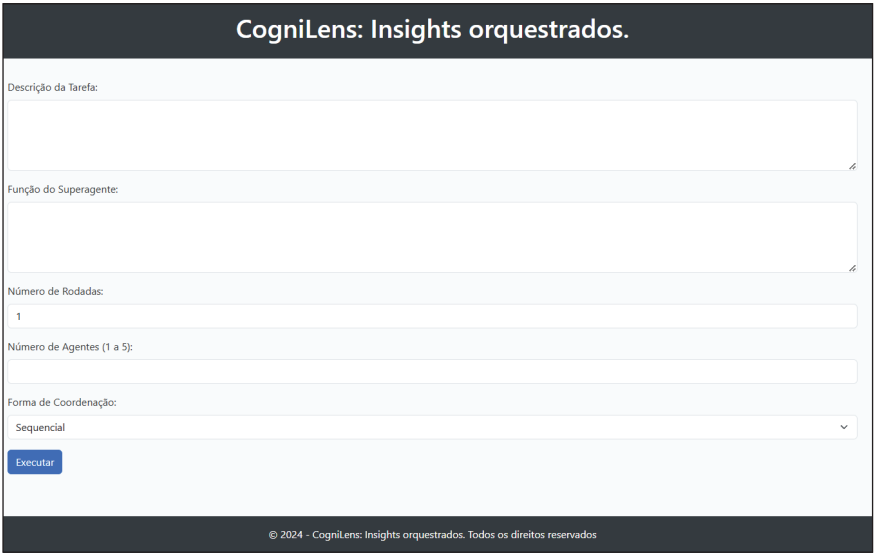
O CogniLens utiliza Flask como *framework web* para sua interface, integrando o modelo Gemini para processar as interações dos agentes e gerar resultados analíticos. A plataforma está configurada para receber descrições de tarefas, informações dos agentes e o método de coordenação desejado (sequencial ou paralelo) e, crucialmente, o número de rodadas de interação entre os agentes, como ilustrado na **Figura 4**.

O *framework* Flask foi escolhido devido à sua simplicidade, o que permite uma integração direta com o modelo Gemini, utilizado como o núcleo analítico do sistema. O Gemini, por sua vez, processa a entrada fornecida pelo usuário e distribui as tarefas para os agentes configurados, analisando os resultados para produzir relatórios finais. A arquitetura também foi projetada com um *pipeline* flexível para adaptar-se a diferentes necessidades e tipos de tarefas.

Também foi definida a gestão do número de rodadas de interação entre os agentes. O superagente, orquestrado pelo Gemini, itera sobre um número definido de rodadas, solicitando a cada agente a execução de sua parte na tarefa em cada iteração. Os resultados de cada rodada são compilados e utilizados para guiar as interações nas rodadas subsequentes, permitindo um refinamento iterativo da solução.

Recebido em 26-02-2025 • Aceito em 08-05-2025

Figura 4 - Interface do CogniLens.



Para suportar a complexidade das tarefas, o CogniLens utiliza um mecanismo de registro de parâmetros detalhados que rastreiam cada etapa do processamento, desde a entrada do usuário até a geração de relatórios. Além disso, o sistema foi desenhado para suportar execução paralela de agentes no futuro, embora atualmente opere com uma abordagem sequencial.

Por fim, a arquitetura foi planejada para escalabilidade, permitindo que o sistema seja implementado em ambientes de produção usando contêineres Docker. Isso não apenas simplifica o *deploy*, mas também possibilita que o sistema seja executado em diferentes plataformas de nuvem, como AWS, *Google Cloud, Vercel, Netlify e Render*. Atualmente, a versão funcional deste modelo está hospedada no Render, e pode ser acessada pelo seguinte endereço: <https://superagente.onrender.com/>, garantindo maior acessibilidade e *performance*.

3.2 Funcionalidades e métodos

As funcionalidades do sistema foram projetadas para oferecer uma experiência integrada e eficiente, abrangendo desde a configuração de tarefas até a geração de relatórios detalhados. Com ferramentas que permitem

Recebido em 26-02-2025 • Aceito em 08-05-2025

a gestão de agentes, definição de sub-tarefas, coordenação com o modelo Gemini e a produção de análises completas, o sistema garante flexibilidade, organização e *insights* valiosos para o usuário.

• Configuração de Tarefas: Permite a definição de uma tarefa principal e suas sub-tarefas associadas. A Listagem 1 - Função executar_tarefa **(Figura 5)** apresenta a implementação de uma função em *Python* que coordena agentes para realizar etapas específicas de uma tarefa, retornando os resultados obtidos. A função é estruturada para aceitar parâmetros como a descrição da tarefa, agentes e o método de coordenação (sequencial ou paralelo) e o número de rodadas.

Figura 5 - Listagem 1 - Função executar_tarefa.

```
def executar_tarefa(tarefa, agentes, coordenação, num_rodadas,
funcao_superagente):
    """
    Executa uma tarefa passo a passo, pedindo para cada agente executar sua
parte e retorna os resultados.
    :param tarefa: tarefa a ser executada
    :param agentes: lista de agentes a serem utilizados
    :param coordenação: como os agentes devem ser coordenados
    :param num_rodadas: número de rodadas de conversas entre agentes
    :param funcao_superagente: descrição detalhada da função do superagente
    :return: um dicionário com os resultados, onde a chave é o nome do agente
e o valor é o resultado da execução
    """
    resultados = {}
    prompt_base = f"""Você é um superagente. Sua função é:
{funcao_superagente}. Sua tarefa é: {tarefa}. Você deve coordenar os
seguintes agentes de forma {coordenação}. Os agentes são:"""

    for agent in agentes:
        prompt_base += f"""
            - **Agente**: {agent['nome']}.
            - **Especialidade**: {agent['especialidade']}.
            - **Responsabilidades**: {agent['responsabilidades']}.
            - **Habilidades**: {agent['habilidades']}.
            """

    prompt_base += "Agora, execute a tarefa passo a passo, pedindo para cada
agente executar sua parte e me retorne os resultados no seguinte formato:
```nome_agente: resultado_agente```"

    for rodada in range(num_rodadas):
        prompt_inicial = f"""{prompt_base} \n Esta é a rodada {rodada + 1}.
"""
        resultados_str = model.generate_content(prompt_inicial).text
        linhas = resultados_str.strip().split("\n")
        for linha in linhas:
            if ":" in linha:
                nome_agente, resultado_agente = linha.split(":",1)
                resultados[nome_agente.strip()] = resultado_agente.strip()
    return resultados
```

Recebido em 26-02-2025 • Aceito em 08-05-2025
```

• Gestão de Agentes: Possibilita aos usuários configurar agentes, definindo especialidades, responsabilidades e habilidades. A função criar_agente vista na Listagem 2 **(Figura 6)** é responsável por estruturar e encapsular as informações de um agente em um dicionário, facilitando sua manipulação em tarefas posteriores. Essa função recebe como parâmetros o nome, a especialidade, as responsabilidades e as habilidades do agente, permitindo configurar agentes de forma flexível e personalizada. O retorno é um dicionário que organiza essas informações de maneira clara, garantindo que cada agente possua uma descrição detalhada e padronizada para ser utilizado em processos como coordenação e execução de tarefas no sistema.

Figura 6 - Listagem 2 - Função criar_agente.

```
def criar_agente(nome, especialidade, responsabilidades, habilidades):
    """
    Cria um agente a partir dos parâmetros fornecidos.

    :param nome: nome do agente
    :param especialidade: especialidade do agente
    :param responsabilidades: responsabilidades do agente
    :param habilidades: habilidades do agente
    :return: um dicionário com as informações do agente
    """

    return {
        "nome": nome,
        "especialidade": especialidade,
        "responsabilidades": responsabilidades,
        "habilidades": habilidades,
    }
```

• Coordenação e Execução: Utiliza o modelo Gemini para orquestrar a execução das tarefas com base nos agentes configurados. A configuração da API do Gemini é uma etapa fundamental para habilitar a integração do sistema com os recursos de geração de conteúdo fornecidos pelo modelo. O código visto na Listagem 3 **(Figura 7)** verifica a presença da chave de API necessária, armazenada como uma variável de ambiente, garantindo que o acesso seja seguro e protegido. Caso a chave não esteja definida, é levantada uma exceção para alertar sobre o problema. Em seguida, a chave é usada para configurar a API através da biblioteca *genai*, e o modelo Gemini 2.0 *Flash Exp* é inicializado, permitindo sua utilização para processar e gerar conteúdo em tarefas específicas.

Recebido em 26-02-2025 • Aceito em 08-05-2025

Figura 7 - Listagem 3 - Configuração da API do Gemini.

```
# Configuração da API do Gemini
google_api_key = os.getenv("GOOGLE_API_KEY")
if not google_api_key:
    raise EnvironmentError("A variável de ambiente GOOGLE_API_KEY não
está definida.")

genai.configure(api_key=google_api_key)
model = genai.GenerativeModel('gemini-2.0-flash-exp')
```

• Geração de Relatórios: A função gerar_relatorio tem como objetivo consolidar e apresentar, de forma organizada e detalhada, todas as informações relacionadas à execução de uma tarefa no sistema. Utilizando os dados fornecidos, como a descrição da tarefa, os agentes envolvidos, a estratégia de coordenação utilizada, os resultados obtidos e a análise gerada pelo superagente, a função constrói um relatório final em formato *Markdown*. Esse relatório é estruturado com seções claras, incluindo detalhes sobre cada agente, os resultados individuais e uma visão geral analítica, permitindo fácil interpretação e uso posterior. Abaixo, encontra-se o código completo da função. Veja o código completo na Listagem 4 (**Figura 8**) .

Figura 8 - Listagem 4 - Função gerar_relatorio.

```
def    gerar_relatorio(tarefa,    agentes,    coordenação,    resultados,
analise_superagente, funcao_superagente):
    """
    Gera um relatório final com todos os dados da tarefa, agentes,
coordenação, resultados e análise do superagente.
    :param tarefa: tarefa executada
    :param agentes: lista de agentes utilizados
    :param coordenação: como os agentes foram coordenados
    :param resultados: resultados da execução da tarefa por cada agente
    :param analise_superagente: análise do superagente sobre a execução
da tarefa
    :param funcao_superagente: função do superagente
    :return: um texto no formato markdown com o relatório final
    """
    relatorio_texto = f"""
    # Relatório Final #
    ## Função do Superagente ##
    {funcao_superagente}

    ## Tarefa ##
    {tarefa}
    ## Agentes ##
    """
```

Recebido em 26-02-2025 • Aceito em 08-05-2025

```
    for agent in agentes:
        relatorio_texto += f"""
        ### {agent['nome']} ###
        - **Especialidade:** {agent['especialidade']}
        - **Responsabilidades:** {agent['responsabilidades']}
        - **Habilidades:** {agent['habilidades']}
        """
    relatorio_texto+= f"""
    ## Coordenação ##
    {coordenação}

    ## Resultados ##
    """
    for nome_agente, resultado_agente in resultados.items():
        relatorio_texto += f"""
        ### {nome_agente} ###
        {resultado_agente}
        """
    relatorio_texto +=f"""
    ## Análise do Superagente ##
    {analise_superagente}
    """
    return relatorio_texto
```

3.3 Implementação e implantação

O sistema foi projetado para ser executado em um ambiente conteinerizado utilizando *Docker,* uma tecnologia que simplifica o processo de empacotamento, distribuição e execução de aplicações. Essa abordagem permite que o sistema seja facilmente implantado em diversas plataformas de nuvem, como por exemplo o *Render,* que garante a portabilidade e a consistência no ambiente de execução, independentemente da infraestrutura subjacente. Além disso, o uso de contêineres reduz conflitos de dependências, tornando o processo de manutenção e atualização mais eficiente.

Para reforçar a segurança, o sistema utiliza variáveis de ambiente para armazenar credenciais sensíveis, como a chave de *Application Programming Interface* (API) necessária para acessar o modelo Gemini. Essa prática elimina a necessidade de expor informações críticas diretamente no código fonte, minimizando riscos de vazamentos e fortalecendo a proteção contra acessos não autorizados. Com essa combinação de conteinerização e segurança baseada em variáveis de ambiente, o sistema garante um processo de implantação (*deploy*) robusto, seguro e adequado para ambientes de produção.

Recebido em 26-02-2025 • Aceito em 08-05-2025

3.4 Procedimentos de Teste

Nesta subseção são detalhados os testes realizados no CogniLens, envolvendo diferentes abordagens metodológicas com foco em validar a funcionalidade, a usabilidade, o desempenho e a robustez da plataforma em um cenário realista.

3.4.1 Tipos de Testes Realizados

Foram realizados três tipos principais de testes:

- Testes Funcionais: Avaliaram a consistência e correção das funcionalidades centrais, como a criação de agentes, execução de tarefas, geração de relatórios e coordenação por múltiplas rodadas. As funções executar_tarefa, criar_agente e gerar_relatorio foram testadas individualmente e em conjunto, com o objetivo de garantir a integridade lógica e a interoperabilidade dos módulos.
- Testes de Desempenho: Monitoraram o tempo de resposta do modelo Gemini e da aplicação Flask, bem como o uso de memória e CPU em tarefas mais exigentes. Embora o sistema ainda opere de forma sequencial, as simulações consideraram diferentes cargas e variações na quantidade de agentes e tarefas.
- Testes de Usabilidade: Avaliaram a experiência do usuário, especialmente no preenchimento dos parâmetros da tarefa, visualização dos resultados e compreensão do relatório final. Foram utilizadas entrevistas e simulações com usuários simulados para coletar feedback qualitativo.
- Ferramentas Utilizadas
- Plataforma CogniLens: Executada em ambiente de testes via Render, com observação direta da resposta do modelo Gemini 2.0 Flash Exp.
- Docker: Utilizado para conteinerização do ambiente de testes, garantindo reprodutibilidade e isolamento de dependências.
- Logs e registros internos: Permitem rastrear cada etapa da execução, possibilitando uma análise detalhada do comportamento do sistema em tempo real.

3.4.2 Estudo de Caso Testado

O principal cenário testado envolveu a simulação de um Plano de Recuperação e Prevenção de Tragédias no Rio Grande do Sul, baseado em uma catástrofe real ocorrida em 2024, quando inundações afetaram mais de 80 mil pessoas.

Recebido em 26-02-2025 • Aceito em 08-05-2025

O superagente SynkroAI coordenou cinco agentes especializados:

- AnalytiQ – Estratégia e Gestão de Crise
- CodeMind – Desenvolvimento de Tecnologia
- DataSage – Análise de Dados e Insights
- FlowMaster – Automação e Logística
- SecureShield – Segurança e Resiliência

A tarefa envolvia a construção de um plano detalhado de recuperação, passando pelas fases de resposta imediata, reconstrução, prevenção e monitoramento contínuo. Cada agente atuou conforme suas responsabilidades, em seis rodadas de interação.

O estudo de caso utilizado como base para esses testes pode ser consultado de forma livre e completa no seguinte endereço eletrônico: <https://github.com/maromo71/superagente/blob/main/PLANO_REC_RS.docx>

A partir da configuração desse cenário, a plataforma CogniLens executou a tarefa em múltiplas rodadas, orquestrando as interações entre os agentes e gerando um relatório final com os resultados consolidados da simulação. Esse relatório apresenta, em formato Markdown, os objetivos da tarefa, as funções de cada agente, os resultados intermediários de cada rodada, a estratégia de coordenação adotada e uma análise final elaborada pelo superagente. O relatório final gerado automaticamente está disponível em: https://github.com/maromo71/superagente/blob/main/FINAL_GERADO.txt.

4. RESULTADOS E DISCUSSÃO

A implementação do CogniLens resultou em uma plataforma funcional, modular e escalável, capaz de simplificar processos colaborativos complexos. Os testes demonstraram a eficácia da arquitetura em ambientes que demandam alto grau de coordenação e análise estratégica, como demonstrado no cenário de recuperação do Rio Grande do Sul, segundo Magalhães Filho *et al.* (2024) e Brack (2024).

A coordenação sequencial de múltiplos agentes mostrou-se robusta, com atribuições distribuídas de forma clara e dinâmica. A geração de relatórios, ancorada nas análises do superagente SynkroAI, forneceu *insights* estruturados e detalhados, capazes de apoiar a tomada de decisão em situações críticas. A cada rodada, os resultados foram refinados, seguindo uma abordagem iterativa eficiente.

Em termos de desempenho, o modelo Gemini revelou-se adequado ao contexto aplicado, com tempo de resposta satisfatório e alto nível de

Recebido em 26-02-2025 • Aceito em 08-05-2025

contextualização nas respostas. Sua integração com o *framework* Flask e a capacidade de adaptação às entradas do usuário permitiram flexibilidade sem comprometer a coerência dos resultados.

Entretanto, desafios persistem para a futura implementação de execução paralela. Questões como a sincronização entre múltiplos agentes, uso otimizado de recursos computacionais e controle de concorrência devem ser tratados em versões posteriores da plataforma. A adoção de paradigmas como programação assíncrona (*asyncio*) ou *frameworks* de orquestração distribuída (como *Celery*) podem contribuir para esse avanço.

Do ponto de vista da interface, embora funcional, a plataforma ainda carece de recursos visuais que aprimorem a experiência do usuário. Recomenda-se o desenvolvimento de *dashboards* interativos com gráficos, indicadores e alertas em tempo real. Tais melhorias facilitariam a interpretação dos dados e tornariam a ferramenta mais atrativa e intuitiva, especialmente para usuários não técnicos.

Em consonância com autores como Russell e Norvig (2020), que destacam a importância da aplicabilidade prática da IA em sistemas colaborativos, o CogniLens representa um passo promissor rumo à automação de decisões estratégicas com base em múltiplos agentes especializados. Sua aplicabilidade extrapola o contexto acadêmico e pode beneficiar setores como gestão pública, saúde, logística e educação.

CONSIDERAÇÕES FINAIS

O CogniLens ocupa um espaço importante na aplicação de modelos de linguagem natural para coordenação de tarefas e análise colaborativa. A plataforma mostrou-se eficaz em ambientes que demandam automação, destacando-se pela habilidade de integrar agentes com diferentes especialidades em fluxos de trabalho otimizados. Testes realizados demonstraram não apenas a viabilidade técnica, mas também o impacto prático da plataforma em cenários reais, como a resposta a crises e a reconstrução de áreas afetadas, gerando *insights* valiosos para a tomada de decisões estratégicas.

Um dos maiores desafios é a implementação de execução paralela de agentes, o que poderia melhorar significativamente a escalabilidade e a eficiência do sistema em cenários mais complexos. Adicionalmente, a interface do usuário necessita de melhorias, como *dashboards* interativos e relatórios gráficos mais intuitivos, especialmente para usuários menos técnicos. Tais atualizações facilitariam a interpretação dos dados

Recebido em 26-02-2025 • Aceito em 08-05-2025

e aumentariam a eficácia geral do sistema.

No futuro, espera-se que o CogniLens evolua para integrar novos modelos de linguagem e expandir suas funcionalidades, permitindo maior flexibilidade em diferentes cenários de uso. A continuidade deste trabalho pode servir de base para outros sistemas que buscam otimizar processos colaborativos por meio de inteligência artificial, contribuindo para avanços tecnológicos, sociais e ambientais em diversas áreas.

REFERÊNCIAS

BRACK, P.. **Tragédia climática e ambiental no Rio Grande do Sul em 2024.** Instituto Humanitas Unisinos - IHU, 27 maio 2024. Disponível em: https://www.ihu.unisinos.br/categorias/639766-tragedia-climatica-e-ambiental-no-rio-grande-do-sul-em-2024-artigo-de-paulo-brack. Acesso em abril de 2025.

BROWN, T. B. *et al.*. ***In***: NeurIPS, 2020. **Language Models are Few-Shot Learners**. Disponível em: https://papers.nips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf Acesso em dezembro de 2024.

DEVLIN, J. *et al.*. ***In***: BURSTEIN, J.; DORAN, C.; SOLORIO, T. (Eds.). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics:** Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: https://aclanthology.org/N19-1423. DOI: 10.18653/v1/N19-1423. Acesso em dezembro de 2024.

GOOGLE. **Apresentamos o Gemini 2.0.** Disponível em: https://blog.google/intl/pt-br. Acesso em dezembro de 2024.

LIU, Y. *et al.*. **RoBERTa:** A Robustly Optimized BERT Pretraining Approach. CoRR, v. abs/1907.11692, 2019. Disponível em: http://arxiv.org/abs/1907.11692. Acesso em dezembro de 2024.

MAGALHÃES FILHO, F. J. C.; MENDES, A. T.; SANTOS, G. R. dos; BENETTI, A. D.; DORNELLES, F.. Enchentes e inundações no Rio Grande do Sul em 2024: impactos e desafios para a gestão integrada de políticas públicas no saneamento básico. **Boletim Regional,** Urbano e Ambiental, Brasília, nº 33, p. 23-32, dez. 2024. Disponível em: https://repositorio.ipea.gov.br/bitstream/11058/16385/1/BRUA_33_Artigo_1_Enchentes_e_inundacoes.pdf. Acesso em abril de 2025.

RUSSELL, S.; NORVIG, P.. **Artificial Intelligence**: A Modern Approach. 4. ed. Hoboken, NJ: Pearson, 2020.

VASWANI, A. *et al.*. **Attention Is All You Need**. CoRR, v. abs/1706.03762, 2017. Disponível em: http://arxiv.org/abs/1706.03762. Acesso em dezembro de 2024.

Recebido em 26-02-2025 • Aceito em 08-05-2025

YANG, Z. *et al*.. **XLNet:** Generalized Autoregressive Pretraining for Language Understanding. Disponível em: https://papers.nips.cc/paper_files/paper/2019/hash/dc6a7e655d7e5840e-66733e9ee67cc69-Abstract.htm. Acesso em dezembro de 2024. NeurIPS, 2021.

Recebido em 26-02-2025 • Aceito em 08-05-2025