Chapter 2

Desenvolvendo um Sistema Multi-Agentes para Pesquisa Científica com LangGraph

Larissa Souza do Nascimento e Ricardo Moura Sekeff Budaruiche

Abstract

In recent years, Artificial Intelligence (AI) has evolved rapidly, becoming a strategic technology across many areas of knowledge. In the field of scientific research, AI plays a crucial role in automating tasks, processing large volumes of data, and generating relevant insights. This chapter presents a practical approach to building a multi-agent system for supporting scientific research using the ***LangGraph*** *framework. The system enables the orchestration of autonomous agents specialized in search, validation, and data analysis, integrating tools such as Tavily AI, arXiv API, and ChromaDB. We discuss the architecture, implementation process, and ethical considerations related to AI usage in science. The goal is to equip researchers and developers with both conceptual understanding and practical skills to apply multi-agent systems in their research workflows, promoting efficiency, transparency, and reproducibility in scientific investigation.*

Resumo

Nos últimos anos, a Inteligência Artificial (IA) evoluiu rapidamente, consolidando-se como uma tecnologia estratégica em diversas áreas do conhecimento. No campo da pesquisa científica, a IA exerce papel essencial na automação de tarefas, no processamento de grandes volumes de dados e na geração de insights relevantes. Este capítulo apresenta uma abordagem prática para a construção de um sistema multi-agentes de apoio à pesquisa científica utilizando o framework LangGraph. O sistema permite a orquestração de agentes autônomos especializados em busca, validação e análise de dados, integrando ferramentas como Tavily AI, arXiv API e ChromaDB. São discutidos a arquitetura, o processo de implementação e as considerações éticas envolvidas no uso da IA na ciência. O objetivo é capacitar pesquisadores e desenvolvedores com conhecimentos conceituais e práticos para empregar sistemas multi-agentes em fluxos de pesquisa, promovendo eficiência, transparência e reprodutibilidade na investigação científica.

2.1. Introdução

Nas últimas décadas, o avanço dos modelos de linguagem de larga escala (*Large Language Models* – LLMs) tem transformado de maneira profunda a forma como sistemas computacionais interagem com o conhecimento humano. Esses modelos demonstram capacidade crescente de compreender, sintetizar e gerar linguagem natural com um nível de fluidez sem precedentes, o que tem impulsionado novas aplicações em diversas áreas, da educação à pesquisa científica (Jordan; Mitchell, 2015).

Com o crescimento exponencial do volume de informações disponíveis em bases acadêmicas e na *web*, tornou-se inviável para pesquisadores acompanhar manualmente todas as publicações e inovações relevantes em seus respectivos campos de estudo. Nesse contexto, a automação de processos cognitivos, como busca, filtragem e análise de dados científicos, emerge como uma necessidade estratégica.

A Inteligência Artificial (IA), especialmente em sua vertente de Aprendizagem de Máquina e dos LLMs, vem desempenhando papel central nesse movimento de transformação. Ferramentas baseadas em IA são atualmente capazes de coletar, validar e integrar conhecimento proveniente de múltiplas fontes de forma contextualizada, apoiando o pesquisador não apenas como instrumento de busca, mas como um parceiro cognitivo (Meloni *et al.*, 2023). Essa nova geração de sistemas inteligentes está alinhada com as tendências discutidas em (Bender *et al.*, 2021), que ressaltam a importância de equilibrar o poder dos modelos de linguagem com mecanismos de controle, explicabilidade e responsabilidade ética, aspectos fundamentais para sua aplicação em ambientes científicos.

Inserido nesse cenário, o presente capítulo apresenta o desenvolvimento do ***SAPIEN – Sistema de Agentes para Pesquisa e Ensino***, uma solução computacional baseada em um sistema **multiagente** e construída sobre o *framework LangGraph*. O *SAPIEN* foi concebido para automatizar etapas do ciclo de pesquisa científica, incluindo a coleta de artigos em fontes como *arXiv* e *Tavily AI*, o processamento linguístico e semântico dos textos, a validação contextual por meio de *embeddings* e o armazenamento vetorial otimizado com *ChromaDB*. O sistema é resultado de esforços acadêmicos originalmente apresentados em (Nascimento; Budaruiche, 2025), sendo aqui expandido com aprofundamento teórico, descrição técnica e análise arquitetural.

A escolha do **LangGraph** como núcleo do sistema fundamenta-se em suas propriedades de modularidade e persistência de estado. Diferentemente de *frameworks* lineares, como o *LangChain*, o *LangGraph* permite representar fluxos de execução em forma de grafo direcionado, no qual cada nó corresponde a um agente e as arestas controlam o fluxo condicional de dados. Essa característica possibilita a criação de ciclos de retroalimentação, em que os agentes podem reavaliar resultados anteriores e aprimorar suas decisões (Chase, 2024).

Além disso, a natureza *stateful* do *LangGraph* permite que o sistema mantenha memória contextual ao longo das interações, assegurando continuidade no raciocínio e coerência nas respostas, aspecto essencial em domínios científicos complexos. Essa arquitetura mostra-se particularmente adequada a cenários nos quais múltiplos agentes, como buscadores, validadores e analisadores semânticos, devem cooperar para atingir um objetivo comum: transformar grandes volumes de dados em conhecimento estruturado e

relevante.

Dessa forma, este capítulo tem como objetivo apresentar, de maneira detalhada, os fundamentos teóricos, metodológicos e tecnológicos do **SAPIEN**. São discutidos o processo de configuração e desenvolvimento do sistema, sua arquitetura modular, a integração entre agentes e ferramentas de IA, bem como as implicações éticas e técnicas associadas à utilização de modelos de linguagem na ciência contemporânea. Ao final, busca-se demonstrar que sistemas multiagentes baseados em LLMs, como o *SAPIEN*, constituem uma abordagem promissora para a modernização dos métodos de pesquisa científica, promovendo maior eficiência, reprodutibilidade e transparência no processo de descoberta do conhecimento.

2.2. Fundamentação Teórica

2.2.1. Agentes Inteligentes

O conceito de agente inteligente constitui um dos alicerces da pesquisa em IA e em sistemas multiagente. Tradicionalmente, define-se um agente como uma entidade capaz de perceber o ambiente por meio de sensores, processar essas percepções e agir sobre o ambiente por meio de atuadores, de modo a perseguir objetivos explícitos ou implícitos de forma autônoma. Essa definição enfatiza três aspectos fundamentais, percepção, raciocínio/decisão e ação, que conferem ao agente um comportamento orientado por propósito.

Em uma perspectiva contemporânea, os agentes inteligentes, ou *AI agents*, são sistemas de *software* altamente autônomos, capazes de perseguir metas em nome de usuários ou outras entidades, demonstrando raciocínio, planejamento e memória ao longo da execução de tarefas compostas. Essas capacidades incluem: (i) a decomposição de objetivos em subtarefas; (ii) o uso de modelos de raciocínio simbólico, baseados em regras ou estatísticos; (iii) a interação com ferramentas externas, como APIs, bases de dados ou modelos de linguagem; e (iv) a adaptação contínua mediante aprendizagem ou *feedback*. Em suma, tais agentes não apenas reagem ao ambiente, mas também são capazes de antecipar estados futuros, planejar ações e cooperar com outros agentes ou sistemas.

Com o surgimento dos LLMs e das arquiteturas *agentic* (multiagente), observa-se o aparecimento de uma nova categoria de agentes, que incorporam funcionalidades como memória persistente, colaboração entre agentes e decomposição dinâmica de tarefas complexas. Esse avanço amplia a definição clássica ao introduzir um nível superior de autonomia, característico dos chamados *agentic AI systems*, nos quais os agentes são capazes de gerar problemas, formular subobjetivos, delegar atividades a outros agentes e evoluir ao longo do tempo.

No contexto da pesquisa científica, os agentes inteligentes emergem como componentes críticos em sistemas voltados à automação de tarefas de alto volume, como rastreamento de literatura, extração de metadados, análise semântica e validação de resultados. Por meio de arquiteturas multiagente, cada agente assume uma especialização funcional, por exemplo, busca, processamento de linguagem natural (NLP) ou armazenamento vetorial, enquanto um agente supervisor orquestra o fluxo de execução e a comunicação entre eles. Esse tipo de organização favorece a escalabilidade, a modularidade e a manutenção

de um estado contextualizado entre interações.

Além disso, os agentes inteligentes possuem características centrais que os distinguem de simples sistemas automatizados:

- **Autonomia**: capacidade de agir sem intervenção humana contínua, controlando suas próprias ações e decisões;
- **Reatividade**: habilidade de perceber mudanças no ambiente em tempo real e responder de maneira apropriada;
- **Proatividade**: competência para tomar a iniciativa e buscar objetivos futuros, em vez de apenas reagir a estímulos;
- **Sociabilidade**: capacidade de interagir e comunicar-se com outros agentes, humanos ou sistemas, cooperando em prol de metas comuns.

Esses atributos são especialmente relevantes para o sistema aqui descrito, uma vez que os agentes devem operar de forma colaborativa, hierarquizada e persistente, mantendo o contexto, compartilhando conhecimento e refinando sucessivamente suas respostas. Em particular, ao integrar LLMs como componentes de raciocínio, por exemplo, na interpretação de *prompts* ou na análise semântica, é necessário que o agente não apenas execute etapas isoladas, mas também encadeie múltiplas operações, como planejamento, chamada de ferramentas, verificação de consistência e arquivamento de resultados. Assim, configura-se um ciclo contínuo de percepção, decisão e ação.

Por fim, é importante observar que o uso de agentes inteligentes em domínios críticos, como a pesquisa científica automatizada, envolve desafios técnicos e éticos significativos, entre eles a manutenção de memória útil e relevante, a mitigação de alucinações cognitivas de modelos de linguagem, a rastreabilidade das decisões e a modularização da arquitetura para fins de auditoria e supervisão. Esses aspectos reforçam a necessidade de projetar agentes inteligentes não apenas sob a ótica da funcionalidade, mas também da confiabilidade, explicabilidade e adaptabilidade ao longo do tempo.

2.2.2. Sistemas Multiagente e LLMs

Os sistemas multiagente (SMAs) consistem em coleções de agentes autônomos que interagem, cooperam ou competem entre si para resolver problemas distribuídos e, frequentemente, complexos. Em sua formulação clássica, cada agente possui seus próprios sensores, atuadores e objetivos, podendo coordenar suas ações com os demais a fim de alcançar metas compartilhadas (Luo *et al.*, 2019).

Com o advento dos LLMs, esse paradigma foi revitalizado: agentes capazes de empregar raciocínio em linguagem natural, acionar ferramentas externas, comunicar-se entre si e manter estado ao longo do tempo passaram a compor sistemas de apoio em domínios como pesquisa científica, revisão de literatura e descoberta de conhecimento.

No contexto da biblioteca *LangGraph*, os SMAs são formalizados como grafos de execução, nos quais cada nó representa um agente (ou ferramenta) e as arestas definem o fluxo de controle e a atualização de estado. A arquitetura denominada *Supervisor (tool-calling)* foi adotada pelo sistema proposto, o *SAPIEN*, por diversos motivos:

- **Modularidade e especialização**: cada agente, por exemplo, busca *web*, extração de metadados ou validação vetorial, é implementado como uma ferramenta especializada, o que facilita testes, manutenção e evolução individual.
- **Orquestração centralizada**: o agente supervisor atua como ponto de controle lógico, invocando as ferramentas apropriadas conforme o contexto, levando em conta a complexidade da tarefa e a necessidade de conhecimento específico de domínio.
- **Persistência de estado e memória contextual**: por meio do grafo de execução do *LangGraph*, o sistema mantém e transfere estado, como histórico de buscas, *embeddings* gerados e resultados validados, reforçando a continuidade das interações e aprimorando a coerência das respostas.

A adoção dessa arquitetura permite que o *SAPIEN* realize *handoffs* explícitos entre agentes, isto é, que o supervisor delegue a execução a outro agente (ou ferramenta) e receba o resultado por meio de um controle estruturado de grafo, conforme exemplificado na documentação oficial. Essa decomposição do *workflow* em agentes-ferramenta sob controle de um supervisor favorece a escalabilidade do sistema, especialmente quando novos domínios ou fontes de dados são incorporados.

Além disso, a literatura destaca benefícios significativos dos sistemas multiagente em comparação a *pipelines* lineares compostos por agentes isolados:

- **Separação de responsabilidades**: possibilita a criação de agentes especialistas em tarefas específicas, resultando em desempenho superior em relação a agentes genéricos que precisam selecionar entre múltiplas ferramentas ou estratégias;
- **Modularidade e extensibilidade**: facilita a experimentação e a melhoria incremental de cada agente, permitindo atualizações isoladas sem comprometer a integridade do sistema como um todo;
- **Controle de fluxo baseado em grafos**: substitui sequências rígidas de execução por arquiteturas flexíveis, que viabilizam lógica condicional, laços de retroalimentação e interações dinâmicas entre agentes.

Em síntese, ao combinar um agente supervisor controlador com agentes especializados implementados como ferramentas, o *SAPIEN* adota uma arquitetura robusta e escalável para lidar com tarefas típicas da pesquisa científica: coleta de dados provenientes da *web* e de repositórios, processamento semântico, validação cruzada e armazenamento vetorial, todos integrados em um ciclo contínuo de raciocínio e execução coordenada.

2.2.3. Frameworks para Agentes Baseados em LLMs

Entre as principais ferramentas para o desenvolvimento de agentes baseados em modelos de linguagem destacam-se o **LangChain**, o **AutoGen**, o **CrewAI** e o **LangGraph**. Enquanto o *LangChain* se sobressai na definição de fluxos lineares de tarefas, o *LangGraph* possibilita a construção de fluxos complexos e cíclicos, com controle detalhado de

estado e persistência (Barbarroxa; Gomes; Vale, 2024). Essa característica o torna particularmente adequado para aplicações científicas que demandam rastreabilidade, coerência contextual e continuidade de processamento.

Estudos como o de (Khandare *et al.*, 2023) evidenciam o papel central das bibliotecas em *Python* e dos *frameworks* de Inteligência Artificial no ecossistema computacional contemporâneo, ressaltando a importância da modularidade, reusabilidade e extensibilidade na concepção de agentes inteligentes. Nesse contexto, ferramentas como o *LangGraph* representam um avanço significativo, ao proporcionar infraestrutura flexível para orquestração de agentes em ambientes que exigem alta integração entre raciocínio simbólico, aprendizado estatístico e persistência de memória.

2.2.4. Tecnologias Utilizadas

Para o desenvolvimento do sistema multiagente voltado à realização de pesquisas científicas e buscas na *web*, foram empregadas tecnologias modernas e adequadas ao contexto de aplicações baseadas em modelos de linguagem. A seguir, apresentam-se as principais ferramentas utilizadas:

- ***Python***: utilizada como linguagem de programação principal devido à sua ampla adoção na comunidade de inteligência artificial, simplicidade sintática, vasta gama de bibliotecas voltadas ao desenvolvimento de agentes e suporte nativo a paradigmas como orientação a objetos e programação funcional. Além disso, o *Python* apresenta excelente integração com *frameworks* de IA, como *TensorFlow*, *PyTorch*, *LangChain* e *LangGraph*, facilitando tanto a prototipagem quanto o escalonamento de soluções complexas (Khandare *et al.*, 2023).

- ***LangGraph***: *framework* de orquestração baseado em grafos direcionados, utilizado para a construção do sistema multiagente. Permite a criação de fluxos dinâmicos e cíclicos entre agentes, com controle refinado sobre os estados e transições, sendo ideal para aplicações que requerem tomada de decisão condicionada e persistência de estado ao longo de múltiplas interações.

- ***TavilySearch***: ferramenta de busca na *web* integrada via API, utilizada para fornecer respostas baseadas em fontes abertas e atualizadas. Sua principal função no sistema é atender às demandas informacionais gerais que extrapolam o escopo de bases científicas especializadas.

- ***ChromaDB***: banco de dados vetorial responsável pelo armazenamento de *embeddings* extraídos de documentos e artigos científicos. Distingue-se por sua eficiência, interface intuitiva, modelo de código aberto e desempenho otimizado para conjuntos de dados de pequena e média escala. Essas características tornam o *ChromaDB* particularmente apropriado para aplicações acadêmicas e sistemas multiagentes em fase de prototipagem. Além disso, apresenta elevada performance em tarefas de recuperação semântica, como demonstrado em estudos recentes (Mathur; Chhabra, 2024).

- **arXiv API**: fonte de dados especializada, utilizada para acesso estruturado a artigos científicos em formato XML. Permite consultas precisas por metadados (au-

tores, palavras-chave, datas e resumos), sendo essencial para a operação do agente responsável por pesquisas bibliográficas.

- ***Anthropic Claude 3.5***: LLM de última geração utilizado para geração e interpretação de respostas dentro do sistema. Sua alta capacidade de raciocínio, compreensão de contexto extenso e desempenho superior em tarefas de múltiplos turnos o tornam adequado para funções de supervisão e coordenação de subagentes.

2.2.5. Configuração do Ambiente

A configuração do ambiente de desenvolvimento do *SAPIEN* segue um fluxo padronizado, projetado para assegurar a reprodutibilidade e a consistência na execução do sistema. O processo utiliza o interpretador *Python 3.10+* e adota o gerenciamento de dependências por meio de ambiente virtual. Essa abordagem isola bibliotecas e versões, evitando conflitos e facilitando a portabilidade entre diferentes sistemas operacionais (Khandare *et al.*, 2023).

O primeiro passo consiste em clonar o repositório do projeto a partir do GitHub, conforme ilustrado no trecho de código a seguir:

```
git clone https://github.com/larissaNa/SapienAgent.git

cd SapienAgent
```

Em seguida, cria-se um ambiente virtual e ativa-se o contexto de execução:

```
python -m venv venv
#Linux/macOS
source venv/bin/activate
#Windows
.\venv\Scripts\activate
```

A instalação das dependências é realizada a partir do arquivo `requirements.txt`, o qual lista todas as bibliotecas necessárias ao funcionamento do sistema, incluindo **Flask**, **LangGraph**, **LangChain**, **ChromaDB**, **APScheduler**, **SentenceTransformers** e **arXiv API**. Essa etapa é executada com o comando:

```
pip install -r requirements.txt
```

Posteriormente, configuram-se as variáveis de ambiente responsáveis pela autenticação com APIs externas:

```
TAVILY_API_KEY="sua_chave"
ANTHROPIC_API_KEY="sua_chave"
```

Essas chaves são indispensáveis ao funcionamento dos agentes de busca e interpretação, garantindo acesso às ferramentas **Tavily AI** (Tavily, 2024) e **Claude (Anthropic)** (Anthropic, 2024).

Concluída a configuração, a execução do sistema é realizada por meio do comando:

```
python run.py
```

Após a inicialização, a interface web torna-se acessível em `http://127.0.0.1:5000/`, permitindo ao usuário realizar consultas científicas em linguagem natural. A Figura 2.1 apresenta, de forma esquemática, as etapas de preparação e execução do ambiente de desenvolvimento.

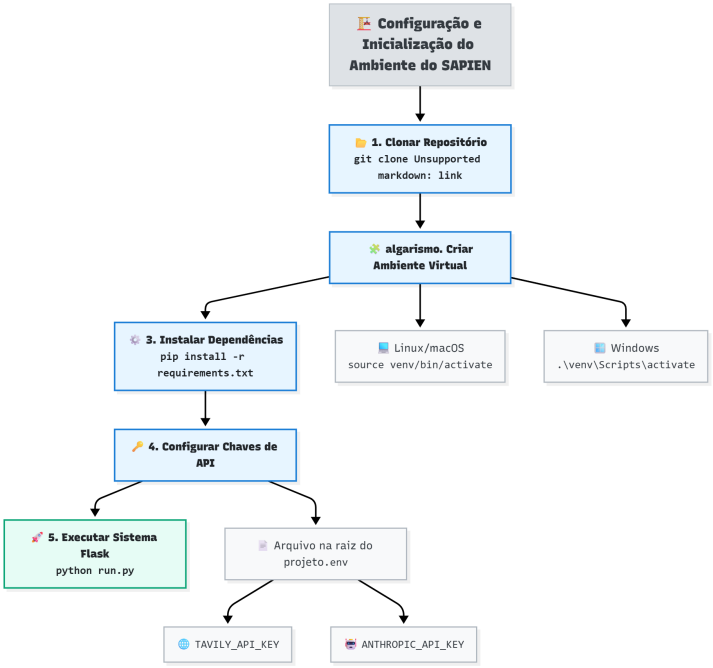


Figura 2.1: Etapas de preparação e execução do ambiente do *SAPIEN*

2.3. Estrutura e Componentes do Projeto

A arquitetura de *software* do *SAPIEN* foi concebida com base em princípios de modularidade e separação de responsabilidades, assegurando manutenção simplificada e extensibilidade para novos agentes ou serviços (Nascimento; Budaruiche, 2025). A estrutura de diretórios segue o padrão adotado em sistemas baseados no *Flask*, organizando os componentes principais conforme descrito na Tabela 2.1.

Além da estrutura de diretórios, o sistema é organizado em módulos funcionais interdependentes. A camada **core** concentra o processamento lógico dos agentes e define o grafo *LangGraph*, responsável por conectar as operações em sequência e ciclos de retroalimentação. O módulo **services.py** gerencia a comunicação assíncrona entre agentes, utilizando *threads* para execução paralela e controle de fluxo de mensagens.

A Figura 2.2 ilustra a estrutura em grafo utilizada para orquestrar os agentes, permitindo controle flexível do fluxo de dados e das decisões dentro do sistema.

Os agentes especializados foram projetados com foco em eficiência e especialização de tarefas.

Tabela 2.1: Estrutura de diretórios e descrição dos principais componentes do *SAPIEN*.

Diretório / Arquivo	**Descrição**
`app/`	Contém o código principal da aplicação Flask e o núcleo multiagente.
`app/core/`	Núcleo do sistema. Implementa o grafo de execução, os agentes e a configuração central.
`app/core/agents.py`	Define os agentes especializados: *tavily_agent*, *arxiv_agent*, *nlp_agent*, *validation_agent*, *chromadb_agent* e *sched_agent*.
`app/core/services.py`	Implementa o fluxo de processamento via LangGraph, gerenciando a troca de mensagens e o raciocínio dos agentes.
`app/core/config.py`	Arquivo de configuração global, responsável pela inicialização de modelos de linguagem, embeddings e banco vetorial.
`app/static/`	Diretório de arquivos estáticos da interface web, incluindo folhas de estilo, scripts JavaScript e imagens.
`app/templates/`	Armazena os templates HTML utilizados na renderização da interface interativa de chat.
`chroma_db/`	Base local de dados vetoriais utilizada para armazenamento semântico de documentos.
`requirements.txt`	Lista de dependências necessárias à execução do projeto.
`run.py`	Ponto de entrada principal da aplicação. Inicializa o servidor Flask e o grafo de agentes.

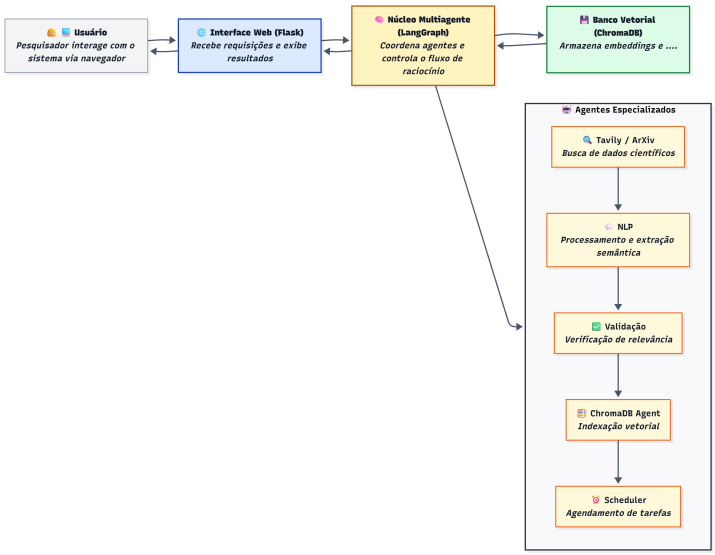


Figura 2.2: Estrutura em grafo utilizada para a orquestração dos agentes no *SAPIEN*

Por exemplo, o **tavily_agent** realiza buscas em fontes abertas, o **nlp_agent** aplica técnicas de processamento linguístico e o **validation_agent** emprega embeddings do

modelo *all-MiniLM-L6-v2* para avaliar similaridade semântica. A Figura 2.3 apresenta o fluxo interno de processamento de um agente, evidenciando as etapas de entrada, transformação e resposta.

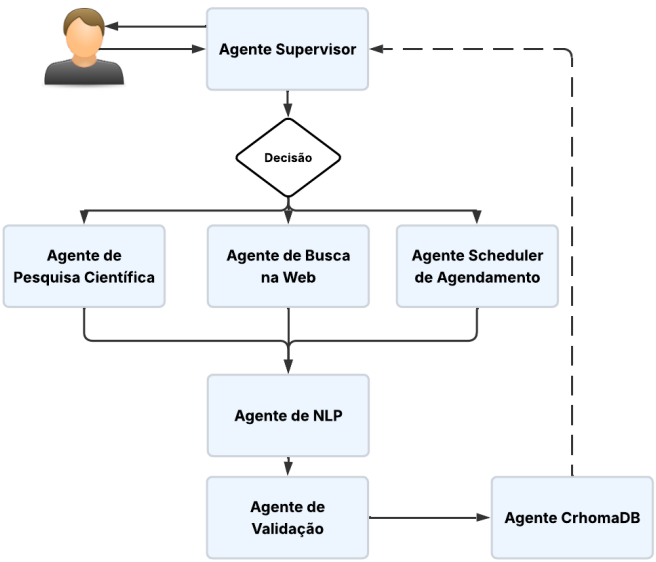


Figura 2.3: Fluxo interno de processamento de um agente especializado no *SAPIEN*

O design modular e desacoplado permite que novos agentes ou ferramentas sejam adicionados sem comprometer a arquitetura existente, promovendo escalabilidade e reutilização de código. Essa organização não apenas aprimora a manutenibilidade e a rastreabilidade do sistema, mas também o torna um recurso didático eficaz para o ensino de arquiteturas baseadas em agentes. A divisão lógica entre camadas possibilita que estudantes e pesquisadores explorem, modifiquem e estendam o *SAPIEN*, favorecendo experimentação e aprendizado em Inteligência Artificial aplicada à pesquisa científica.

2.3.1. Implementação do Sistema

A implementação do *SAPIEN* foi realizada utilizando o *framework* **LangGraph** para modelar o ciclo de execução e interação entre agentes. Cada nó do grafo representa um agente especializado, enquanto as arestas definem a ordem, as dependências e as condições de transição entre etapas de processamento.

O uso de grafos cíclicos no *LangGraph* permite a reavaliação contínua das respostas e o refinamento iterativo dos resultados, promovendo maior consistência semântica e precisão contextual. Esse mecanismo de retroalimentação possibilita que o sistema aprimore progressivamente a qualidade das inferências, ajustando suas decisões com base em verificações sucessivas e validações semânticas realizadas pelos agentes especializados.

2.4. Demonstração Prática e Casos de Uso

A fase de demonstração prática teve como objetivo validar o funcionamento integrado dos agentes que compõem o sistema *SAPIEN*, bem como avaliar sua aplicabilidade em tarefas reais de pesquisa científica. O experimento consistiu na execução de consultas automatizadas a bases de conhecimento acadêmico, com destaque para o repositório *arXiv* e para o mecanismo de busca científica na *web* fornecido pela ferramenta **Tavily AI**.

Durante a demonstração, o sistema foi executado em ambiente local, configurado conforme as etapas descritas na Seção 2.1. O acesso se deu por meio da interface *web* desenvolvida com o *framework* **Flask**, que possibilita a interação direta entre o usuário e o agente supervisor. As consultas foram formuladas em linguagem natural, e o agente supervisor interpretou os comandos, determinando o fluxo de execução e acionando os agentes especializados de acordo com o tipo, o domínio e a complexidade da solicitação.

2.4.1. Execução das Consultas

Foram realizadas diferentes categorias de consultas, com o intuito de demonstrar o potencial do sistema *SAPIEN* no apoio a atividades de pesquisa científica. A seguir, apresentam-se alguns exemplos de comandos executados durante a fase experimental:

- "Busque artigos sobre *transformers* em IA nos últimos dois anos."
- "Agende buscas semanais sobre IA aplicada à educação."
- "Cancele todas as buscas agendadas."

Ao receber uma solicitação, o **Agente Supervisor** inicia o processo iterativo de orquestração das tarefas. Inicialmente, o **Tavily Agent** realiza a busca em fontes abertas de pesquisa, enquanto o **arXiv Agent** coleta publicações recentes em repositórios científicos. Os resultados obtidos são encaminhados ao **NLP Agent**, responsável por extrair informações relevantes, normalizar metadados e executar análises linguísticas sobre os textos recuperados.

Na sequência, o **Validator Agent** aplica filtros semânticos baseados em *embeddings*, assegurando a relevância dos resultados e eliminando duplicidades ou publicações fora do escopo da consulta. Por fim, o **ChromaDB Agent** armazena os documentos processados no banco vetorial local, permitindo consultas futuras baseadas em similaridade semântica e facilitando a reutilização do conhecimento previamente adquirido.

A Figura 2.4 ilustra o protótipo prático da interface entre o usuário e o agente de pesquisa, evidenciando o fluxo de consulta, análise e resposta conduzido pelo sistema.

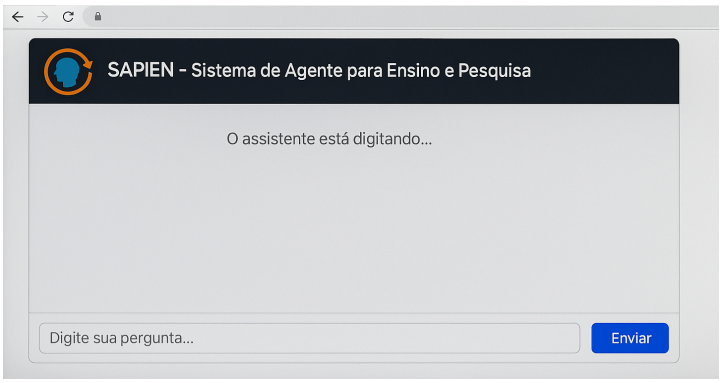


Figura 2.4: Interface de usuário

2.4.2. Casos de Uso Representativos

Entre os cenários avaliados, destacam-se três casos de uso práticos que ilustram o potencial do *SAPIEN* em apoiar pesquisadores e instituições acadêmicas:

1. **Monitoramento temático contínuo:** o sistema possibilita o agendamento de buscas recorrentes sobre tópicos específicos, como "redes neurais convolucionais" ou "aprendizagem federada", executadas em intervalos regulares. Os resultados são atualizados automaticamente, reduzindo a necessidade de acompanhamento manual e garantindo atualizações constantes da base de conhecimento.

2. **Exploração semântica de artigos:** graças à integração com o **ChromaDB**, o *SAPIEN* permite consultas baseadas em similaridade vetorial, possibilitando identificar publicações conceitualmente relacionadas, mesmo quando empregam terminologias distintas. Essa funcionalidade contribui para revisões de literatura mais amplas e interdisciplinares.

3. **Validação automatizada de relevância:** o agente de validação semântica compara os resumos recuperados com o contexto da consulta original, atribuindo pesos de relevância de acordo com a proximidade semântica. Essa etapa reduz significativamente ruídos e elimina resultados pouco pertinentes ao tema em análise.

2.5. Descrição do Código e Implementação dos Módulos

O sistema *SAPIEN* foi desenvolvido segundo princípios de modularidade, clareza estrutural e extensibilidade, favorecendo a integração eficiente entre agentes especializados. Sua implementação segue os fundamentos de engenharia de *software* aplicados a sistemas multiagente, em conformidade com o paradigma de orquestração proposto pelo *framework* **LangGraph**.

Nas subseções seguintes, são apresentados os principais arquivos e trechos de código que compõem o núcleo funcional do sistema, destacando-se o papel de cada módulo no fluxo geral de execução e as interações entre os componentes que formam a arquitetura multiagente do *SAPIEN*.

2.5.1. Roteamento e Inicialização da Aplicação

O ponto de entrada do sistema é o arquivo `app/routes.py`, responsável por inicializar o servidor *Flask* e definir as rotas de comunicação entre a interface web e o núcleo multiagente. A aplicação é instanciada por meio da função `create_app()`, que encapsula a configuração e segue o padrão de projeto *factory*, permitindo modularidade e escalabilidade na implantação em ambientes de produção.

```
from app import create_app
app = create_app()

if name == "main":
app.run(debug=True)
```

Esse design orientado à fábrica (*factory pattern*) favorece a separação de dependências e simplifica o processo de manutenção e expansão do sistema. Durante o desenvolvimento, o servidor é executado em modo de depuração (`debug=True`), possibilitando inspeção em tempo real. Para ambientes de produção, a configuração pode ser facilmente adaptada para execução em contêineres *Docker* ou plataformas em nuvem, garantindo portabilidade e segurança operacional.

2.5.2. Camada de Configuração e Recursos Compartilhados

O módulo `app/core/config.py` centraliza a configuração global do sistema, são carregadas as chaves de autenticação das APIs externas (**Anthropic** e **Tavily**) e instanciados os modelos correspondentes:

```
load_dotenv()

def _set_env(var: str):
    value = os.getenv(var)
    if not value:
        raise EnvironmentError(f"Vari vel de ambiente {var} n o
            definida.")
    return value

_set_env("TAVILY_API_KEY")
_set_env("ANTHROPIC_API_KEY")

....

# caches globais
adicionados_arxiv_links = set()
processed_content_hashes = set()
current_processed_data = None

# scheduler compartilhado
from apscheduler.schedulers.background import BackgroundScheduler
scheduler = BackgroundScheduler()
scheduler.start()
active_jobs = {}
```

Essa camada também gerencia o **BackgroundScheduler**, proveniente da biblioteca *APScheduler*, responsável pela execução automatizada de tarefas em segundo

plano. Por meio desse agendador, o sistema realiza buscas periódicas e atualizações contínuas na base de dados científica, permitindo o monitoramento automatizado de temas de interesse.

2.5.3. Gerenciamento de Estado Compartilhado

O módulo `shared_state.py` implementa um mecanismo leve de controle e compartilhamento de estado entre os agentes durante o ciclo de execução do sistema. Esse estado é mantido em memória e registra informações sobre conteúdos processados, *hashes* de validação e resultados de tarefas agendadas.

```
_current_processed_data: Optional[Dict[str, Any]] = None
_processed_content_hashes: Set[str] = set()
_scheduler_results: list = []
```

Funções auxiliares são responsáveis por definir, recuperar e limpar esses dados de forma controlada, assegurando coerência entre os componentes do sistema e evitando condições de corrida em execuções concorrentes. Esse design fornece uma camada de persistência temporária entre chamadas assíncronas, reduz redundâncias durante execuções iterativas e mantém a consistência do fluxo de informações entre os agentes.

2.5.4. Definição dos Agentes e do Supervisor

O módulo `agents.py` define os agentes especializados, cada um associado a uma função específica no pipeline de pesquisa científica. A criação de cada agente é realizada por meio do método `create_react_agent()` do *framework* **LangGraph**, que associa um modelo de linguagem a ferramentas e comportamentos personalizados.

Os principais agentes definidos são:

- **Tavily Agent**: executa buscas em fontes abertas e bases científicas na web;
- **arXiv Agent**: coleta publicações diretamente do repositório *arXiv*;
- **NLP Agent**: realiza o processamento e a normalização de conteúdo textual;
- **Validation Agent**: aplica filtragem semântica baseada em *embeddings*;
- **ChromaDB Agent**: armazena conteúdos vetorizados para consultas futuras;
- **Scheduler Agent**: gerencia tarefas recorrentes de coleta e atualização.

A coordenação entre os agentes é realizada pelo **Agente Supervisor**, configurado com base na arquitetura *Supervisor Tool-Calling*, conforme descrita na documentação oficial do **LangGraph** (Chase, 2024). Essa abordagem permite que o supervisor atue como controlador dinâmico, decidindo, em tempo de execução, qual agente deve ser invocado e quais parâmetros devem ser transmitidos em cada etapa do processo.

```
supervisor_graph = create_supervisor(
model=llm,
agents=[tavily_agent, arxiv_agent, sched_agent,
```

```
nlp_agent, validation_agent, chromadb_agent],
prompt=("Voc      um supervisor que coordena um sistema "
"multiagente de pesquisas cient ficas.")
)
compiled_supervisor = supervisor_graph.compile()
```

2.5.5. Construção do Grafo e Execução do Pipeline

O módulo `service.py` é responsável pela compilação do grafo principal de execução e pela definição da lógica de fluxo de mensagens entre os agentes. Ele utiliza a classe `StateGraph` do **LangGraph**, na qual cada nó representa um agente e cada aresta define a sequência de comunicação e dependência entre tarefas.

```
graph = StateGraph(StateSchema)
graph.add_node("supervisor", compiled_supervisor)
graph.add_edge(START, "supervisor")
compiled = graph.compile(checkpointer=MemorySaver())
```

A função `run()` inicia o processo a partir da entrada textual fornecida pelo usuário, criando um identificador único de *thread* para rastrear o contexto da sessão. Durante a execução, as mensagens são processadas de forma iterativa, e as respostas são filtradas para capturar apenas as geradas pelos agentes de alto nível, eliminando ruídos intermediários de comunicação. Essa estrutura de grafo permite ciclos de retroalimentação entre agentes, possibilitando reavaliações sucessivas de resultados e refinamento dinâmico das respostas, um diferencial importante frente a *pipelines* lineares tradicionais.

2.5.6. Camada Vetorial e Persistência

O módulo `vectorstore.py` concentra a criação e o gerenciamento do banco vetorial local, implementado com a biblioteca **ChromaDB**. Essa camada é fundamental para o armazenamento semântico e a recuperação eficiente de textos processados, viabilizando consultas baseadas em similaridade vetorial.

```
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/
    all-MiniLM-L6-v2")
vectorstore = Chroma(
collection_name="artigos_cientificos",
embedding_function=embeddings,
persist_directory="./chroma_db"
)
```

A combinação entre *embeddings* densos e armazenamento vetorial permite ao *SAPIEN* executar buscas semânticas de alta precisão, superando as limitações de consultas puramente lexicais. Esse mecanismo garante que documentos semanticamente relacionados possam ser recuperados mesmo quando não compartilham termos idênticos, favorecendo a descoberta de conhecimento e a ampliação de revisões bibliográficas automatizadas.

2.5.7. Ferramentas e Fluxo de Processamento

A camada de ferramentas (`app/core/tools`) é responsável pela implementação das funcionalidades específicas utilizadas pelos agentes do sistema. Cada ferramenta é encap-

sulada em uma função decorada com o modificador `@tool`, fornecido pelo *LangChain*, o que permite sua integração direta ao grafo de execução do *LangGraph*. Essas ferramentas representam as "ações atômicas" do sistema multiagente, seguindo o fluxo lógico de coleta, processamento, validação e armazenamento de dados científicos.

2.5.7.1. Processamento Linguístico (NLP Agent)

O módulo `nlp_process.py` implementa o agente responsável pela normalização linguística, enriquecimento de metadados e preparação de conteúdo para indexação vetorial. Durante a execução, o texto bruto é higienizado, os contadores de palavras e caracteres são extraídos, e um *hash* MD5 é gerado para rastreamento e prevenção de duplicatas. Esses dados são posteriormente armazenados no estado compartilhado do sistema, tornando-se disponíveis para uso pelos demais agentes.

```
normalized_content = re.sub(r'\s+', ' ', raw_content.strip())
content_hash = hashlib.md5(normalized_content.encode()).hexdigest()
set_current_processed_data(processed_data)
```

O agente também realiza a detecção automática do idioma predominante e registra a data de processamento, assegurando a rastreabilidade e a integridade dos documentos analisados. Essa etapa de pré-processamento é essencial para garantir consistência linguística e otimizar o desempenho das consultas semânticas posteriores.

2.5.7.2. Busca e Coleta de Dados Científicos

A etapa de coleta de informação é implementada por dois módulos complementares: `web_search_with_flow.py` e `simple_arxiv_search.py`. O primeiro utiliza a API do **Tavily AI** para efetuar buscas em fontes científicas reconhecidas — como *Nature*, *Science* e *IEEE*, enquanto o segundo acessa diretamente o repositório *arXiv* por meio de requisições HTTP.

```
url = f"http://export.arxiv.org/api/query?search_query=all:{query}
"
r = requests.get(url)
```

Os resultados obtidos são encaminhados ao pipeline completo de processamento (**NLP → Validação → ChromaDB**), assegurando consistência semântica e eliminando duplicidades de artigos já indexados. O agente de coleta foi projetado para lidar com diferentes formatos de entrada, *string*, dicionário ou instância *Pydantic*, o que amplia sua robustez e compatibilidade com o sistema supervisor. Essa flexibilidade reforça a capacidade adaptativa do *SAPIEN*, permitindo sua integração com múltiplas fontes de dados e fluxos de pesquisa automatizada.

2.5.7.3. Validação Semântica

O módulo `validate_content.py` é responsável pela etapa de verificação de relevância semântica, na qual o conteúdo processado é comparado com o tópico de pesquisa orig-

inal. Para isso, utiliza-se o modelo de embeddings *SentenceTransformer all-MiniLM-L6-v2*, com cálculo da similaridade de cosseno entre os vetores de contexto correspondentes.

```
similarity = cosine_similarity(text_embedding, topic_embedding)[0][0]
is_relevant = similarity >= threshold
```

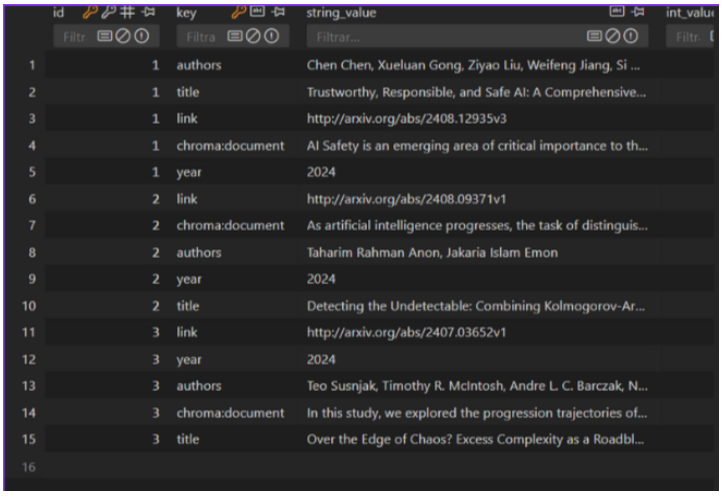
A função `validate_content()` também evita o reprocessamento de textos já avaliados, verificando previamente o estado global de *hashes* armazenados. Quando a similaridade ultrapassa o limiar configurado (geralmente 0.6), o documento é considerado relevante e encaminhado para o módulo de armazenamento vetorial. Esse procedimento assegura que apenas conteúdos semanticamente coerentes com o tema sejam indexados, reduzindo ruído e otimizando a precisão do sistema.

2.5.7.4. Armazenamento Vetorial no ChromaDB

O módulo `store_in_chromadb.py` finaliza o pipeline de processamento, transformando os documentos validados em representações vetoriais e persistindo-os na base *ChromaDB*. Para otimizar o desempenho das consultas por similaridade, o conteúdo textual é segmentado em blocos menores (*chunks*) por meio do *RecursiveCharacterTextSplitter*, o que garante granularidade adequada durante a vetorização.

```
splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
    chunk_overlap=200)
docs = splitter.split_documents([document])
vectorstore.add_documents(docs)
vectorstore.persist()
```

Após a persistência, o estado compartilhado é limpo para evitar redundâncias em execuções subsequentes. Esse mecanismo mantém a base vetorial organizada, consistente e preparada para consultas semânticas de alta eficiência, permitindo que o sistema realize buscas contextuais com grande precisão. A Figura 2.5 demonstra o armazenamento vetorial no banco de dados ChromaDB, sendo armazenado dados como autor, título, link, resumo e ano de publicação.



	id	key	string_value	int_valu
	Filtr	Filtra	Filtrar...	Filtr
1	1	authors	Chen Chen, Xueluan Gong, Ziyao Liu, Weifeng Jiang, Si ...	
2	1	title	Trustworthy, Responsible, and Safe AI: A Comprehensive...	
3	1	link	http://arxiv.org/abs/2408.12935v3	
4	1	chroma:document	AI Safety is an emerging area of critical importance to th...	
5	1	year	2024	
6	2	link	http://arxiv.org/abs/2408.09371v1	
7	2	chroma:document	As artificial intelligence progresses, the task of distinguis...	
8	2	authors	Taharim Rahman Anon, Jakaria Islam Emon	
9	2	year	2024	
10	2	title	Detecting the Undetectable: Combining Kolmogorov-Ar...	
11	3	link	http://arxiv.org/abs/2407.03652v1	
12	3	year	2024	
13	3	authors	Teo Susnjak, Timothy R. McIntosh, Andre L. C. Barczak, N...	
14	3	chroma:document	In this study, we explored the progression trajectories of...	
15	3	title	Over the Edge of Chaos? Excess Complexity as a Roadbl...	
16				

Figura 2.5: Armazenamento Semântico

2.5.7.5. Agendamento de Pesquisas Automáticas

O módulo `scheduler_tools.py` implementa as funções que permitem ao sistema executar pesquisas automáticas de forma periódica, a partir de instruções expressas em linguagem natural. Por exemplo, comandos como:

```
"pesquise sobre agentes inteligentes durante 2 minutos a cada 30
   segundos"
```

são interpretados por meio de expressões regulares, das quais são extraídos os parâmetros de tempo e de tema. As tarefas são então gerenciadas pelo *APScheduler*, que executa as rotinas em segundo plano sem interromper o fluxo principal da aplicação. O agente também oferece suporte para cancelamento de tarefas ativas e consulta de resultados concluídos.

```
scheduler.add_job(tarefa, 'interval', seconds=int_seg, id=job_id)
add_scheduler_result(f"[{tema}] {resultado}")
```

Essa funcionalidade permite que o *SAPIEN* opere de forma contínua e autônoma, realizando o monitoramento temático de publicações científicas sem necessidade de supervisão manual constante. Com isso, o sistema se aproxima de um modelo de *assistente de pesquisa persistente*, capaz de acompanhar tendências e atualizar periodicamente a base de conhecimento.

2.5.8. Síntese da Camada de Ferramentas

A camada de ferramentas constitui o elo entre a orquestração lógica, realizada pelo *LangGraph*, e a execução concreta das tarefas de pesquisa. Cada módulo é projetado para desempenhar uma função autônoma, porém cooperativa, permitindo que o **Agente Supervisor** componha fluxos de execução complexos de forma dinâmica e adaptativa.

Essa abordagem modular, baseada em *tool-calling*, segue o paradigma descrito na documentação do *LangGraph* (Chase, 2024), no qual o supervisor atua como um nó decisório que invoca agentes conforme a necessidade, utilizando chamadas de ferramenta parametrizadas. O resultado é uma arquitetura altamente extensível, na qual novos agentes ou ferramentas podem ser integrados ao pipeline com impacto mínimo na estrutura existente.

A integração entre esses módulos resulta em um sistema robusto e escalável, no qual cada componente desempenha um papel claramente definido dentro de uma arquitetura cooperativa de agentes. O uso do *LangGraph* viabiliza a modelagem declarativa do fluxo de execução, o *ChromaDB* assegura consultas semânticas de alta eficiência, e o *Flask* fornece uma interface web acessível para interação em linguagem natural. Essa combinação tecnológica oferece uma base sólida para expansões futuras, como a inclusão de agentes especializados em sumarização automática, tradução científica ou geração de relatórios temáticos.

2.5.9. Resultados e Benefícios Observados

Os resultados obtidos durante a fase de demonstração confirmaram que o sistema é capaz de executar fluxos multiagentes de forma coordenada, preservando o estado global e garantindo coerência entre as etapas do pipeline. Observou-se que, em consultas envolvendo múltiplas fontes de informação, o tempo médio de resposta foi significativamente reduzido em relação à pesquisa manual tradicional, especialmente em virtude da integração direta com APIs científicas, como *Tavily AI* e *arXiv*.

Além do ganho de eficiência, destaca-se a principal contribuição prática do sistema: a **redução da sobrecarga cognitiva** dos pesquisadores. O *SAPIEN* atua como um assistente inteligente capaz de compreender intenções em linguagem natural, sintetizar informações e apresentar resultados prontos para análise, diminuindo o esforço necessário para coleta e triagem de literatura científica.

Essa abordagem mostra-se especialmente promissora em contextos educacionais e corporativos, nos quais agentes especializados podem ser configurados para coletar, resumir e validar informações em domínios específicos. O uso combinado de *LLMs* e arquiteturas multiagentes, como a implementada no *SAPIEN*, amplia o potencial da automação científica, promovendo maior produtividade, reprodutibilidade e confiabilidade no ciclo de pesquisa.

2.6. Mitigação de Alucinações e Otimização

Para mitigar o fenômeno das alucinações, respostas incorretas ou infundadas geradas por modelos de linguagem, foram aplicadas estratégias de *re-ranking* e *cross-checking*, nas quais os resultados produzidos por diferentes agentes são comparados entre si para validação de consistência (Almeida da Silva *et al.*, 2024). Esse mecanismo permite priorizar respostas convergentes e reduzir a influência de ruído semântico proveniente de fontes isoladas.

A abordagem adotada segue práticas recentes de mitigação propostas em (Perov; Perova, 2024), complementando recomendações de transparência e rastreabilidade em sistemas de IA científica. Além disso, o design modular do *SAPIEN* possibilita incorporar verificadores externos ou heurísticas de controle de confiança (*confidence scoring*), aprimorando o controle de qualidade das respostas e a confiabilidade global do sistema.

2.7. Discussão e Considerações Éticas

A incorporação de agentes autônomos em processos de pesquisa científica representa um avanço expressivo, mas também levanta desafios éticos e epistemológicos relevantes. Modelos de linguagem podem refletir vieses inerentes aos dados de treinamento ou gerar respostas inconsistentes com o estado da arte (Bender *et al.*, 2021). Nesse contexto, a transparência, a responsabilidade e a explicabilidade tornam-se princípios fundamentais para o uso responsável de sistemas de IA (Meloni *et al.*, 2023).

O *SAPIEN* adota práticas de auditoria de histórico e monitoramento contínuo das interações entre agentes, garantindo rastreabilidade das decisões e permitindo a revisão humana dos resultados. Esses mecanismos reduzem o risco de propagação de erros e aumentam a confiabilidade científica, alinhando-se a princípios de governança e ética em

IA.

2.8. Conclusão

Este capítulo apresentou o desenvolvimento do sistema multiagente **SAPIEN**, uma plataforma baseada no *LangGraph* voltada à automação e otimização do processo de pesquisa científica. A arquitetura proposta combina agentes especializados em busca, processamento e validação, integrando múltiplas fontes de informação e mantendo um controle persistente de estado e contexto.

Os experimentos realizados demonstraram o potencial do uso combinado de *LLMs* e sistemas multiagentes para ampliar a eficiência, a qualidade e a rastreabilidade da produção científica. A solução proposta contribui não apenas para reduzir o esforço cognitivo de pesquisadores, mas também para estabelecer um modelo de orquestração modular e auditável, compatível com os princípios de transparência e reprodutibilidade da ciência moderna.

Como trabalhos futuros, pretende-se incorporar suporte a dados multimodais (texto, imagem e código-fonte), explorar métricas de confiança baseadas em evidência cruzada e desenvolver mecanismos colaborativos que permitam interação entre múltiplos usuários ou equipes de pesquisa em tempo real.

References

ALMEIDA DA SILVA, Wildemarkes de *et al.* Mitigation of Hallucinations in Language Models in Education: A New Approach of Comparative and Cross-Verification. *In:* 2024 IEEE International Conference on Advanced Learning Technologies (ICALT). [*S. l.: s. n.*], 2024. p. 207–209. DOI: 10.1109/ICALT61570.2024.00066.

ANTHROPIC. **Anthropic key**. [*S. l.: s. n.*], 2024. Acessado em 20 de março de 2025. Available from: https://www.anthropic.com/.

BARBARROXA, Rafael; GOMES, Luis; VALE, Zita. Benchmarking Large Language Models for Multi-agent Systems: A Comparative Analysis of AutoGen, CrewAI, and TaskWeaver. *In:* MATHIEU, Philippe; DE LA PRIETA, Fernando (eds.). **Advances in Practical Applications of Agents, Multi-Agent Systems, and Digital Twins: The PAAMS Collection**. Cham: Springer Nature Switzerland, 2024. p. 39–48. ISBN 978-3-031-70415-4.

BENDER, Emily M. *et al.* On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *In:* PROCEEDINGS of the 2021 ACM Conference on Fairness, Accountability, and Transparency. Virtual Event, Canada: Association for Computing Machinery, 2021. (FAccT '21), p. 610–623. ISBN 9781450383097. DOI: 10.1145/3442188.3445922. Available from: https://doi.org/10.1145/3442188.3445922.

CHASE, Harrison. **LangGraph Documentation**. [*S. l.: s. n.*], 2024. Acessado em 14 de março de 2025. Available from: https://langchain-ai.github.io/langgraph/.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, v. 349, n. 6245, p. 255–260, 2015. DOI: 10.1126/science.aaa8415. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8415. Available from: https://www.science.org/doi/abs/10.1126/science.aaa8415.

KHANDARE, Anand *et al.* Analysis of Python Libraries for Artificial Intelligence. *In:* BALAS, Valentina Emilia; SEMWAL, Vijay Bhaskar; KHANDARE, Anand (eds.). **Intelligent Computing and Networking**. Singapore: Springer Nature Singapore, 2023. p. 157–177. ISBN 978-981-99-0071-8.

LUO, Tianze *et al.* Multi-Agent Collaborative Exploration through Graph-based Deep Reinforcement Learning. *In:* 2019 IEEE International Conference on Agents (ICA). [*S. l.: s. n.*], 2019. p. 2–7. DOI: 10.1109/AGENTS.2019.8929168.

MATHUR, Srushti; CHHABRA, Aayush. Vector Search Algorithms: A Brief Survey. *In:* 2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS). [*S. l.: s. n.*], 2024. p. 365–371. DOI: 10.1109/ICUIS64676.2024.10866377.

MELONI, Antonello *et al.* Integrating Conversational Agents and Knowledge Graphs Within the Scholarly Domain. **IEEE Access**, v. 11, p. 22468–22489, 2023. DOI: 10.1109/ACCESS.2023.3253388.

NASCIMENTO, Larissa Souza do; BUDARUICHE, Ricardo Moura Sekeff. Como Criar seu Próprio Assistente de Pesquisa Científica com LangGraph. *In:* ANAIS do ENUCOMPI 2025. [*S. l.*]: Sociedade Brasileira de Computação, 2025. DOI: 10.5753/sbc.16935.3.3. Available from: https://doi.org/10.5753/sbc.16935.3.3.

PEROV, Vadim; PEROVA, Nina. AI Hallucinations: Is “Artificial Evil” Possible? *In:* 2024 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT). [*S. l.: s. n.*], 2024. p. 114–117. DOI: 10.1109/USBEREIT61901.2024.10584048.

TAVILY. **Tavily key**. [*S. l.: s. n.*], 2024. Acessado em 20 de março de 2025. Available from: https://www.tavily.com/.