UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

JOÃO PEDRO DE MORAES MADRUGA

AGENTCOMPILER: COMPILADOR LLM PARA ORQUESTRAÇÃO PARALELA DE MULTIAGENTES

Recife,
2025

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

AGENTCOMPILER: COMPILADOR LLM PARA ORQUESTRAÇÃO PARALELA DE MULTIAGENTES

Trabalho apresentado ao programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador(a): Filipe Carlos de Albuquerque Calegario

Aprovado em: 09/04/2025

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Madruga, João Pedro de Moraes.
AgentCompiler: compilador LLM para orquestração paralela de multiagentes / João Pedro de Moraes Madruga. - Recife, 2025.
33 p., tab.

Orientador(a): Filipe Carlos de Albuquerque Calegario
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Sistemas de Informação - Bacharelado, 2025.
Inclui referências.

1. Sistemas MultiAgentds. 2. Large Language Models (LLMs). 3. Inteligência Artificial. 4. Frameworks Multiagentes. 5. Agents. 6. Python. I. Calegario, Filipe Carlos de Albuquerque . (Orientação). II. Título.

000 CDD (22.ed.)

AGENTCOMPILER: COMPILADOR LLM PARA ORQUESTRAÇÃO PARALELA DE MULTIAGENTES

Trabalho apresentado ao programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em: 09/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Filipe Carlos de Albuquerque Calegario (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Geber Lisboa Ramalho (Examinador Interno)

Universidade Federal de Pernambuco

RESUMO

Nos últimos anos, a adoção de modelos de linguagem (Large Language Models, ou LLMs) cresceu exponencialmente, impulsionando aplicações em áreas como processamento de linguagem natural, análise de dados e suporte à tomada de decisão. Em paralelo, sistemas multiagentes — compostos por entidades autônomas que interagem para alcançar objetivos comuns ou complementares — vêm sendo empregados em problemas complexos e distribuídos, como logística e simulações de cenários de negócios. No entanto, a coordenação de vários agentes pode agravar desafios de latência, custo computacional e manutenção de acurácia. Para atenuar esses problemas, este trabalho propõe um sistema multiagentes inspirado nas técnicas de compiladores para otimização de chamadas de funções, adaptando o conceito de LLMCompiler por meio de uma abordagem dinâmica baseada em *Plan and Solve*. Um agente orquestrador decide quais agentes ativar e a forma de comunicação — hierárquica, centralizada, descentralizada ou em pool compartilhado — em um pipeline flexível, capaz de definir topologias e sequências de execução. A avaliação contempla métricas como latência, quantidade de agentes envolvidos, buscando equilibrar eficiência computacional e qualidade das respostas.

Palavras-chave: multiagentes; LLM; orquestração; compiladores; Plan and Solve; otimização.

ABSTRACT

In recent years, the adoption of language models (Large Language Models, or LLMs) has grown exponentially, driving applications in areas such as natural language processing, data analysis, and decision support. In parallel, multi-agent systems—composed of autonomous entities that interact to achieve common or complementary goals—have been employed in complex and distributed problems, such as logistics and business scenario simulations. However, coordinating multiple agents can exacerbate challenges related to latency, computational costs, and maintaining accuracy. To address these issues, this work proposes a multi-agent system inspired by compiler techniques for optimizing function calls, adapting the concept of LLMCompiler through a dynamic approach based on Plan and Solve. An orchestrator agent decides which agents to activate and how they will communicate—whether hierarchical, centralized, decentralized, or via a shared pool—in a flexible pipeline capable of defining topologies and execution sequences. The evaluation considers metrics such as latency, number of agents involved, aiming to balance computational efficiency and the quality of responses.

Keywords: multiagents; LLM; orchestration; compilers; Plan and Solve; optimization.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

1.1. Contexto e Problema

Nos últimos anos, os modelos de linguagem de grande escala (LLMs) revolucionaram a forma como interagimos com a tecnologia, impulsionando aplicações em áreas como atendimento ao cliente, criação de conteúdo e desenvolvimento de software. Estudos recentes mostram que esses modelos vêm sendo aplicados em chatbots, assistentes virtuais e ferramentas automatizadas de geração de código (Jin et al., 2024; Vaillant et al., 2024). Além disso, a popularização de sistemas como o ChatGPT e o GPT-4 evidencia o impacto que a produção de linguagem exerce sobre a cognição humana, influenciando a maneira como percebemos e interpretamos o mundo.

Com a evolução dessas LLMs, surgiram também frameworks multiagentes voltados a coordenar tarefas complexas, aproveitando o potencial desses modelos para dividir e gerenciar processos entre múltiplos agentes. Porém, esses frameworks ainda enfrentam desafios consideráveis, sobretudo em relação à latência e aos custos computacionais. Muitas vezes, a execução sequencial e o grande número de ativações de agents necessários para gerar respostas robustas ocasionam um volume excessivo de chamadas, conforme apontado por McRae (2024, “Rethinking AI Agents”), que descreve esse fenômeno como “excessiva quantidade de chamadas para gerar respostas completas”.

Buscando soluções para otimizar esse cenário, é possível inspirar-se em técnicas de compiladores tradicionais, capazes de planejar e paralelizar a execução de programas. Dada a semelhança na necessidade de coordenar diversas instruções para obter resultados rápidos e eficientes, este trabalho propõe adaptar princípios de compilação para a orquestração de LLMs em ambientes multiagentes. Conforme destacado por Kim et al. (2024, Abstract, L53–L63), estratégias de execução paralela inspiradas em compiladores têm se mostrado eficazes para reduzir latência e custos computacionais em sistemas que demandam múltiplas chamadas de função, indicando a viabilidade de aplicar essa abordagem em frameworks multiagentes.

1.2. Objetivos

1.2.1. Objetivo Geral

Propor o desenvolvimento de um sistema multiagente para LLMs que empregue princípios de compilação, com o intuito de otimizar a execução e a comunicação entre os agentes, reduzindo latência e custos computacionais sem prejudicar a precisão das respostas.

1.2.2. Objetivos Específicos

- Desenvolver um compilador LLM para orquestração paralela de multiagentes.
- Comparar com frameworks alternativos (Agent Compiler, LangGraph com Supervisor e CrewAI), avaliando seu desempenho e identificando ganhos em performance e escalabilidade.
- Promover avanços na eficiência operacional, reduzindo o tempo de resposta e otimizando o consumo de recursos computacionais.
- Assegurar confiabilidade e precisão, mantendo equilíbrio entre eficiência operacional e qualidade das respostas, bem como garantir adaptabilidade em sistemas multiagentes para LLMs.

2. CONCEITOS BÁSICOS

A seguir, são descritos os conceitos essenciais para o desenvolvimento do estudo em questão. Cada tópico busca evidenciar sua relevância dentro da proposta de orquestração multiagente aqui investigada, além de justificar o papel que desempenha na arquitetura do sistema.

2.1. LLMs (Large Language Models)

No contexto do presente trabalho, os LLMs (Large Language Models) constituem o alicerce computacional que possibilita a geração de linguagem natural com alto grau de fluência e coerência, recurso essencial para integrar diversos agentes em um ambiente multiagente. Esses modelos contam com bilhões de parâmetros, treinados em extensos conjuntos de dados textuais, o que lhes confere a habilidade de compreender e produzir texto em variados domínios. Kim et al. (2024, p. 3) indicam que, durante o processo de inferência e de treinamento, as informações são decompostas em tokens – unidades mínimas de análise do texto – e esse tratamento facilita o escalonamento das aplicações. Wang et al. (2023, p. 2) reforçam que, apesar dos desafios inerentes ao custo computacional, a escalabilidade dos LLMs propicia ganhos consideráveis em tarefas de raciocínio, justificando sua adoção como tecnologia base em arquiteturas que lidam com múltiplos agentes.

2.2. **Tokens**

A relevância dos tokens ganha destaque na configuração deste trabalho em razão da necessidade de gerenciar custos e latências decorrentes das chamadas aos LLMs. Cada interação ativa o processamento de um número específico de tokens, fator diretamente associado ao desempenho do sistema. Wang et al. (2023, p. 3) observam que a quantidade de tokens processados impacta não apenas o tempo de resposta, mas também a viabilidade econômica das aplicações, sobretudo em ambientes multiagentes que demandam múltiplas ativações. A redução de consultas redundantes e a delimitação cuidadosa dos prompts revelam-se estratégias fundamentais para equilibrar desempenho e custo.

2.3. **Agents**

Na composição do modelo proposto, os agentes atuam como unidades autônomas responsáveis por interpretar, decidir e executar ações no ecossistema multiagente. Essa autonomia permite que cada agente se concentre em tarefas específicas, como filtragem de dados ou interação direta com o usuário, o que favorece a modularidade e a escalabilidade do sistema. De acordo com as diretrizes descritas em “Building Effective Agents” (Anthropic, disponível em https://www.anthropic.com/engineering/building-effective-agents) e conforme destacado em An Introduction to MultiAgent Systems (Wooldridge, 2009), esses módulos podem operar de forma independente ou cooperativa, comunicando-se para solucionar problemas complexos. A flexibilidade oferecida pelos agentes e a capacidade de colaboração reforçam a viabilidade de empregar LLMs de maneira orquestrada, uma vez que cada agente interage seletivamente com o modelo, segundo sua função no fluxo geral.

2.4. **Orquestração de Agents**

A noção de orquestração de agents é introduzida como o mecanismo que possibilita a integração lógica das ações de cada agente, assegurando que a execução das tarefas ocorra de forma coordenada. Kim et al. (2024, p. 6) salientam que essa coordenação promove a sinergia e, em muitos casos, a paralelização de subprocessos, resultando em menor tempo de resposta total. Em aplicações de grande escala, a orquestração de agentes torna-se imprescindível para racionalizar o uso de recursos computacionais, incluindo os próprios LLMs, dado que cada chamada envolve um custo tanto de latência quanto de processamento de tokens. No presente estudo, a definição de quando e como cada agente é acionado integra o esquema maior de otimização inspirado nas práticas de compilação, explicado adiante.

2.5. Plan and Solve

A estratégia Plan and Solve aparece como uma técnica de organização mental e operacional para guiar os LLMs na resolução de problemas, dividindo tarefas complexas em subtarefas menores. Essa forma de conduzir o raciocínio fornece um roteiro que minimiza falhas ou lacunas ao longo do processo, conforme ilustrado por Wang et al. (2023, p. 4). Estudos recentes, como o de Wang et al. (Plan-and-Solve Prompting, 2023), demonstram que essa abordagem pode reduzir erros de cálculo e aumentar a clareza dos passos de raciocínio, contribuindo para uma performance aprimorada dos modelos. No escopo deste trabalho, o modelo elabora um “plano” antes de iniciar a execução, evitando consultas redundantes ao LLM e assegurando que cada agente receba instruções específicas. Em comparação com abordagens que disparam chamadas ao modelo de forma não estruturada, a adoção de um esquema Plan and Solve demonstra maior eficiência, visto que reduz o número de iterações desnecessárias e aumenta a precisão dos resultados obtidos.

2.6. Frameworks Multiagentes: CrewAI e LangGraph

A análise dos frameworks CrewAI e LangGraph contribui para situar os desafios enfrentados no desenvolvimento de arquiteturas multiagentes que empregam LLMs. No CrewAI, a ênfase recai sobre a colaboração entre agentes em tarefas interativas, embora a estrutura hierárquica possa limitar a flexibilidade de adaptação a cenários não previstos. Já o LangGraph adota uma representação em forma de grafos para organizar tarefas e dependências, o que viabiliza a execução paralela de subtarefas. Ambas as abordagens evidenciam a relevância do gerenciamento de dependências e do aproveitamento de paralelismo, ao mesmo tempo em que apontam para oportunidades de otimização adicionais. Esses insights corroboram a necessidade de um método que reconheça pontos de paralelização e evite operações redundantes, resultando em reduções de latência e custo (CREW AI, 2023; disponível em: https://crew.ai; LANGGRAPH, 2023; disponível em: https://langgraph.com).

2.7. Compiladores no Contexto de LLMs

A analogia com compiladores tradicionais é introduzida para ilustrar como a otimização global de chamadas ao LLM pode ser implementada. Assim como um compilador analisa e traduz código-fonte em instruções de máquina eficientes, um sistema baseado em LLMs é capaz de “compilar” a entrada textual em um plano de execução otimizado, antecipando necessidades de paralelização e minimizando repetições de consultas. Kim et al. (2024, p. 2) discutem a importância da identificação de padrões paralelizáveis e da gestão de dependências para reduzir o overhead computacional e acelerar o processamento de grandes quantidades de dados. Ao incorporar esses princípios de compilação, o presente trabalho propõe um framework no qual agentes coordenados exploram ao máximo a capacidade dos

LLMs, conservando recursos e ampliando a escalabilidade do sistema. Estudos como o de Kim et al. (LLM Compiler for Parallel Function Calling, 2024) evidenciam que a execução paralela de chamadas pode reduzir significativamente a latência e os custos operacionais, reforçando a validade desta abordagem.

3. TRABALHOS RELACIONADOS

Frameworks multiagentes baseados em LLMs surgiram como abordagem para coordenar múltiplos agentes de IA em tarefas complexas, organizando agentes autônomos especializados que se comunicam e dividem tarefas para alcançar um objetivo comum, potencialmente aumentando a escalabilidade e a eficácia em comparação com sistemas monolíticos. Além disso, estudos indicam que a aplicação de técnicas de prompt engineering pode otimizar a comunicação entre os agentes, mitigando parte da sobrecarga computacional inerente a esses sistemas (SHIN et al., 2023).

Entretanto, a literatura destaca desafios significativos nesses frameworks. A coordenação entre agentes impõe sobrecarga de comunicação e processamento, o que pode aumentar a latência e os custos computacionais. Por exemplo, interações adicionais, degradam a experiência do usuário e estudos reportam que frameworks multiagentes podem gerar gastos elevados – chegando a milhares de dólares em poucos dias de execução contínua (Kim et al., 2024). Além disso, a orquestração de agentes exige mecanismos de controle elaborados para sincronizar decisões e evitar conflitos, com riscos de desalinhamento e alucinações em fluxos complexos.

3.1. CrewAI

CrewAI é um framework open-source que organiza agentes em “crews”, onde cada agente assume um papel especializado dentro de uma estrutura hierárquica. Nesta abordagem, um agente gerente supervisiona e delega tarefas aos agentes subordinados, facilitando a especialização e a integração com ferramentas externas. Contudo, essa estrutura hierárquica rígida limita a capacidade de redistribuição dinâmica das tarefas, uma vez que os papéis e as rotas de comunicação são predefinidos, dificultando adaptações a cenários imprevistos. Além disso, a coleta de dados de uso anonimizados, embora útil para análises, pode levantar questões de privacidade em contextos corporativos. Tais limitações evidenciam a necessidade de soluções que combinem especialização com maior flexibilidade na orquestração dos agentes (CREWAI, 2023; disponível em: https://crew.ai).

3.2. LangGraph

O LangGraph, parte do ecossistema LangChain, adota uma abordagem baseada em grafos para modelar os fluxos de trabalho. Nesse framework, as tarefas são representadas como nós, e as dependências entre elas como arestas, permitindo uma visualização clara do fluxo e um comportamento mais determinístico

durante a execução. Essa representação facilita a identificação das relações e a execução paralela de subtarefas independentes. No entanto, a necessidade de definir previamente o grafo de tarefas impõe uma configuração rígida, que pode dificultar iterações rápidas e a adaptação a cenários com requisitos dinâmicos ou menos estruturados. Além disso, em fluxos complexos com muitos nós, a limitação da janela de contexto dos LLMs pode comprometer a manutenção das informações relevantes, resultando em alucinações e desvios durante a execução (LANGGRAPH, 2023; disponível em: https://langgraph.com).

3.3. **Desafios Comuns em Frameworks Multiagentea**

A análise dos trabalhos relacionados revela desafios comuns em frameworks multiagentes, dentre os quais se destacam:

- Latência: O aumento do tempo de resposta decorre das múltiplas interações mediadas por modelos de linguagem, especialmente quando as chamadas de funções são realizadas de forma sequencial. Métodos tradicionais para invocar funções exigem um encadeamento de raciocínio e ações que resulta em alta latência, maior custo e, por vezes, resultados imprecisos (Kim et al., 2024).

- Custo Computacional: O uso intensivo dos modelos de linguagem implica custos elevados, uma vez que cada chamada processa um grande número de tokens. Estudos demonstram que a adoção de uma orquestração otimizada pode reduzir esses custos de forma significativa, atingindo economia de até 6,7 vezes, o que ressalta a importância de técnicas como armazenamento em cache, compressão e o uso de modelos especializados (Kim et al., 2024).

- Escalabilidade: Com o aumento do número de agentes, surgem desafios na coordenação e na manutenção de um estado compartilhado, tanto na capacidade de suportar mais agentes quanto em interações prolongadas. Esses problemas evidenciam a dificuldade de gerenciar dependências entre múltiplas chamadas e de manter a consistência das informações durante a execução (Kim et al., 2024).

- Flexibilidade na Orquestração: A rigidez imposta por estruturas hierárquicas fixas ou por grafos predefinidos limita a capacidade dos frameworks de se adaptarem a fluxos de trabalho dinâmicos. O replanejamento dinâmico, por sua vez, consiste na habilidade de ajustar a sequência de tarefas durante a execução, com base nos resultados intermediários, permitindo que o sistema se adeque a variações nas condições operacionais e otimize a resolução de problemas complexos (Kim et al., 2024).

4. METODOLOGIA

Para realizar este trabalho, adota-se a estratégia “Plan and Solve”, na qual um agente orquestrador analisa o problema e, de forma dinâmica, determina quais agentes serão acionados e como se dará o fluxo de informações. A orquestração pode ocorrer de modo hierárquico, descentralizado, centralizado ou com base em um pool de mensagens, adaptando-se às necessidades de coordenação. Esta pesquisa foi estruturada em cinco etapas principais, subdivididas em sub-tópicos, com o objetivo de construir um ambiente experimental robusto para a avaliação de sistemas multiagentes baseados em LLMs.

4.1. Construção do Agent Compiler

O desenvolvimento do Agent Compiler foi estruturado como um sistema modular, projetado para orquestrar eficientemente a interação entre agentes baseados em Modelos de Linguagem Grandes (LLMs). Esse sistema utiliza diferentes estratégias de comunicação, permitindo maior flexibilidade na realização das tarefas solicitadas pelos usuários.

A arquitetura principal do Agent Compiler é fundamentada no plano de execução, oferecendo múltiplas opções para coordenar o trabalho dos agentes de forma adaptável. As possíveis abordagens implementadas são:

- Execução em camadas: os agentes são organizados de forma hierárquica, onde cada nível depende dos resultados obtidos no nível anterior; [3]
- Execução descentralizada: agentes comunicam-se diretamente entre si, em uma rede distribuída, sem a necessidade de um coordenador central; [3]
- Execução centralizada: um agente central é responsável por coordenar todas as interações entre os demais agentes, atuando como intermediário principal; [3]
- Execução em grupo compartilhado: os agentes possuem acesso a um repositório comum, onde trocam e acessam informações necessárias para executar suas tarefas. [3]

Figura 1: Pipeline do Agent Compiler.

Conforme ilustrado na figura acima, o fluxo de operação inicia-se com o agente orquestrador, que analisa a tarefa proposta pelo usuário, dividindo-a em etapas claras e objetivas. Em seguida, este agente gera um plano detalhado, indicando quais agentes devem ser utilizados, suas funções específicas e o modelo de comunicação mais adequado para o contexto identificado. Posteriormente, o

Compilador de Agentes seleciona a estratégia apropriada com base nesse plano, coordenando a execução dos agentes conforme definido inicialmente. Nesse momento, o sistema gerencia o fluxo de informações, garantindo que os agentes recebam e compartilhem adequadamente os dados necessários ao cumprimento das etapas da tarefa. Após concluírem suas respectivas funções, os agentes retornam os resultados obtidos ao agente orquestrador, que realiza a síntese dessas informações em uma única resposta coesa e compreensível, atendendo plenamente à solicitação inicial do usuário. Para garantir precisão e qualidade nas respostas, os agentes utilizam modelos avançados de linguagem, tais como o GPT-4o-mini, capazes de interpretar contextos complexos e fornecer respostas especializadas. Além disso, o sistema como um todo é construído com técnicas rigorosas de validação de dados, assegurando robustez e confiabilidade nas informações entregues ao usuário final.

4.2. **Construção dos Conjuntos de Dados**

Nesta etapa, foram desenvolvidos dois conjuntos de dados específicos:

4.2.1. **Dataset de Atendimento ao Cliente**

Este dataset foi elaborado com casos de teste inspirados em situações reais de atendimento, contendo, para cada registro, os seguintes elementos:

- **Tema:** Exemplos como "Rastreamento de Pedido", "Consulta de Assinatura", "Solicitação de Reembolso", entre outros.
- **Pergunta:** A questão formulada pelo cliente.
- **Resposta Esperada:** O texto que representa a resposta ideal para cada caso.
- **Métricas:** Registros de latência, número de ativações realizadas e os detalhes dos agentes acionados.

4.2.2. **Dataset de Artigos Científicos**

Neste dataset, foram selecionados trechos extraídos de artigos acadêmicos, com o intuito de avaliar a capacidade do sistema em processar, extrair e resumir informações científicas. Cada registro contém:

- **Caso de Teste:** Informações como título, fonte, URL e conteúdo textual extraído do artigo.
- **Pergunta:** A questão formulada pelo cliente.
- **Fluxo de Processamento:** As etapas vão desde a descoberta do artigo até a geração de um relatório estruturado.
- **Métricas:** São mensuradas latência, número de ativações, contagem de tokens e custo estimado.

4.3. Definição dos Agentes

Nesta etapa, foi desenvolvido um conjunto de agentes com papéis bem definidos, cuja configuração visa simular o comportamento de um sistema multiagente. Os agentes foram organizados de acordo com dois contextos distintos:

4.3.1. Conjunto de agentes para atendimento ao cliente

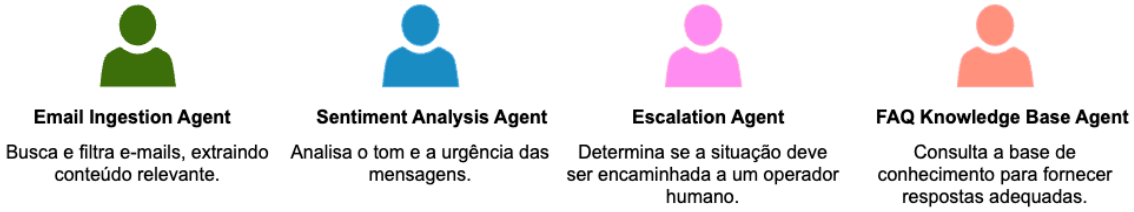

Figura 2: Agentes para atendimento ao cliente.

4.3.2. Agentes para busca de artigos científicos

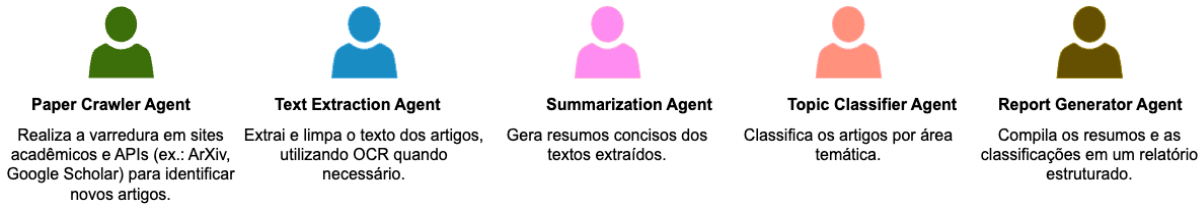

Figura 3: Agentes para busca de artigos científicos.

4.4. Execução dos experimentos

Nesta etapa, foi implementada a mesma estrutura com os mesmos prompts em três frameworks distintos: o **Agent Compiler** (o framework proposto neste artigo), o **Supervisor LangGraph** e o **CrewAI**. Essa abordagem permitiu uma comparação direta entre as diferentes arquiteturas multiagentes. Foram executados os casos de teste dos dois conjuntos de dados, e durante cada execução foram coletadas as seguintes métricas:

- **Latência:** Tempo total de processamento para cada caso de teste.
- **Número de Ativações:** Contabilização das chamadas realizadas pelos agentes.
- **Resposta Final:** Comparação entre a resposta gerada e a resposta esperada.
- **Registro de Agentes e Ferramentas:** Identificação dos agentes e das ferramentas acionadas.

4.5. Configuração do Ambiente

- **Execução:** Os três frameworks foram testados em ambiente controlado, utilizando os mesmos conjuntos de prompts e dados.
- **Coleta de Métricas:** Foram monitoradas variáveis como latência, quantidade de tokens utilizados e custo estimado de cada operação.

4.6. Integração dos Agentes com os Conjuntos de Dados

- **Fluxo de Trabalho:** Para o dataset de atendimento, os agentes interagiram para processar e responder as solicitações dos clientes; para o dataset de artigos científicos, os agentes colaboraram para identificar, extrair, resumir e compilar informações em relatórios.
- **Registro de Ativações:** Cada interação dos agentes foi registrada, permitindo a análise do comportamento do sistema e a verificação da eficiência na orquestração dos agentes.

4.7. Análise dos Resultados

Após a execução dos experimentos, foi realizada uma análise detalhada dos resultados para avaliar a performance dos sistemas multiagentes. Essa análise foi dividida em duas frentes:

4.7.1. Avaliação Quantitativa

- **Latência:** Medição do tempo total de resposta para cada caso de teste.
- **Número de Ativações:** Contabilização das chamadas realizadas pelos agentes.

4.7.2. Avaliação Qualitativa

- **Precisão das Respostas:** Comparação entre as respostas geradas e as respostas esperadas definidas nos datasets.
- **Registro de Agentes e Ferramentas:** Verificação dos agentes e ferramentas acionados para identificar padrões, gargalos e eventuais falhas na comunicação.

5. RESULTADOS E ANÁLISE

A seguir, apresentam-se os resultados e a análise dos experimentos realizados utilizando dois conjuntos de dados distintos, um voltado para o atendimento ao cliente e outro para a busca de pesquisas científicas.

5.1. Dataset de atendimento ao cliente

Para a avaliação do desempenho dos frameworks multiagentes no atendimento ao cliente, foi utilizado uma base de dados dedicada a testes, composta por diversos casos de teste com temas como 'Rastreamento de Pedido', 'Consulta de Assinatura', 'Solicitação de Reembolso', entre outros. Os resultados foram analisados em termos de latência, número de ativações, consumo de tokens e custo estimado, permitindo uma comparação detalhada entre os frameworks Agent Compiler, LangGraph e CrewAI.

5.1.1. Análise quantitativa

5.1.1.1. Latência

Para a avaliação do desempenho dos frameworks multiagentes no atendimento ao cliente, foi utilizada uma base de dados dedicada a testes, composta por diversos casos de teste. Cada caso foi avaliado em termos de latência, com os dados provenientes dos três frameworks (Agent Compiler, LangGraph e CrewAI).

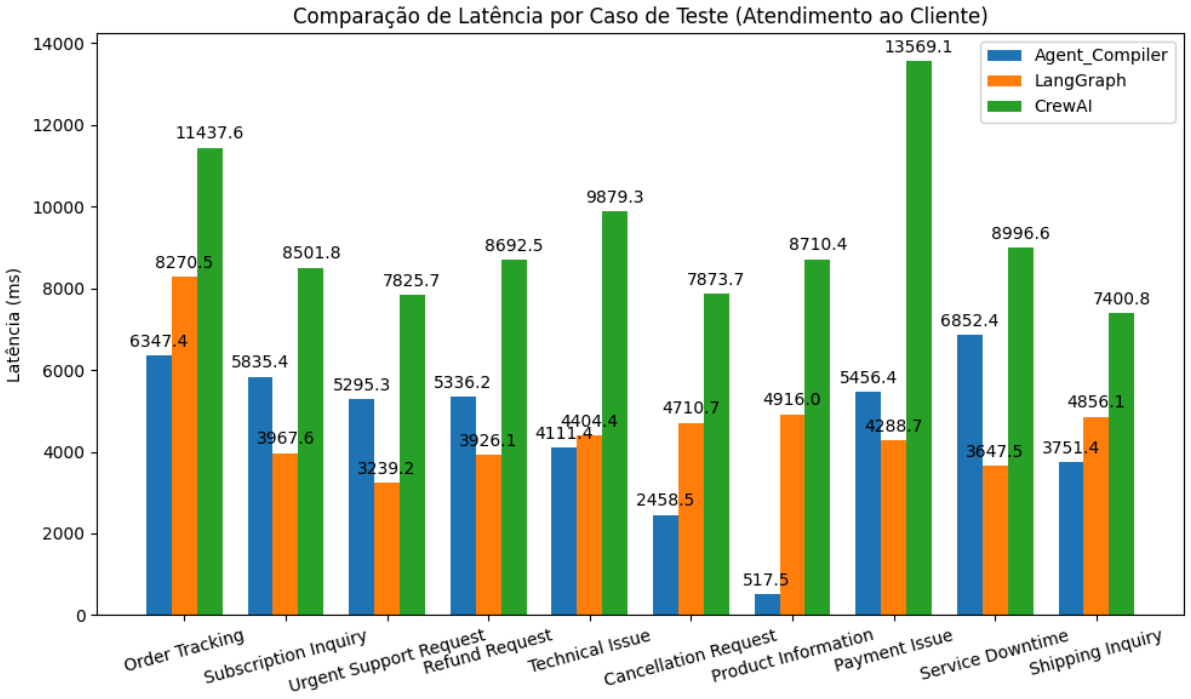

Figura 4: Comparação de Latência por Caso de Teste (Atendimento ao Cliente)

Conforme ilustrado na Figura 4, o Agent Compiler apresentou uma latência média de 4322,10 ms, enquanto o LangGraph obteve 4512,21 ms e o CrewAI atingiu 8974,46 ms. Esses valores demonstram que o Agent Compiler e o LangGraph exibiram tempos de resposta relativamente próximos, indicando uma eficiência temporal semelhante. Em contrapartida, o CrewAI apresentou um tempo de resposta significativamente mais elevado, sugerindo maior complexidade operacional e possíveis gargalos na orquestração de agentes. Desse modo, embora o LangGraph e o Agent Compiler possam ser considerados eficientes para a maioria dos cenários de atendimento, o CrewAI se mostrou menos competitivo em termos de latência para o dataset explorado.

5.1.1.2. Eficiência Operacional

Para analisar a eficiência operacional dos diferentes frameworks no atendimento ao cliente, foi registrado o número de ativações de cada agente em cada caso de teste. Essa métrica possibilita identificar a complexidade do processo interno de cada sistema, bem como o potencial impacto na latência e na escalabilidade.

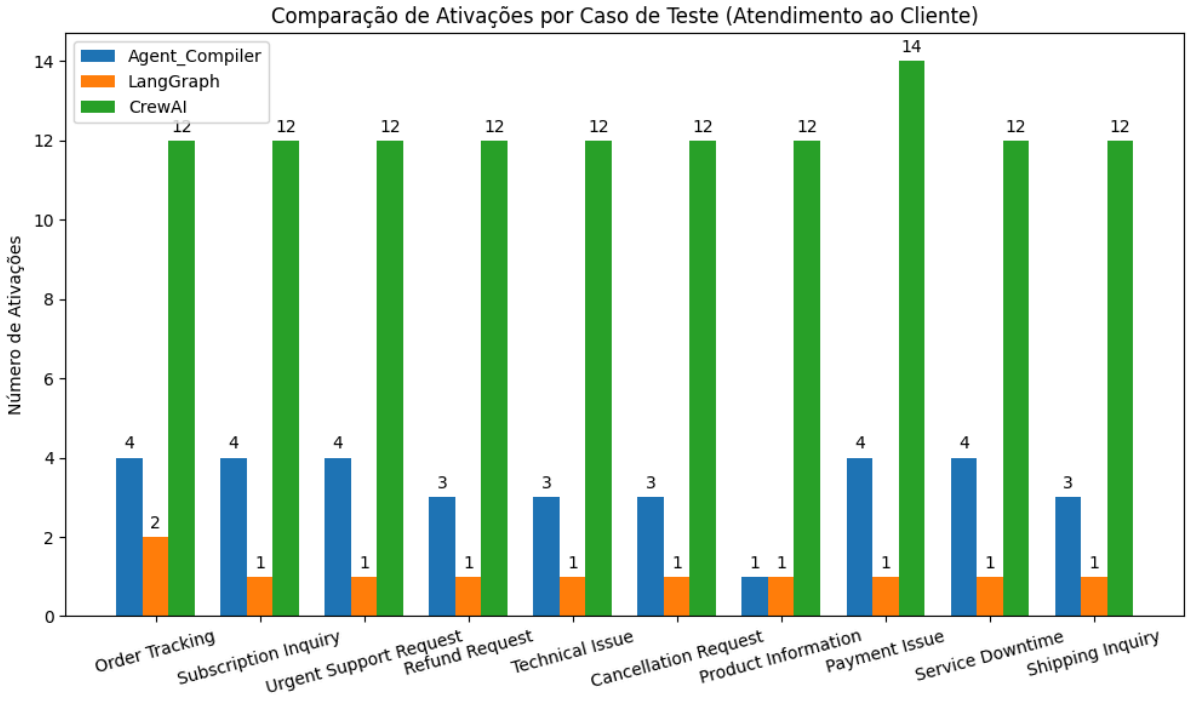

Figura 5: Comparação de Ativações por Caso de Teste (Atendimento ao Cliente)

Conforme ilustrado na Figura 5, o CrewAI apresenta o maior número de ativações, variando entre 12 e 14 chamadas por caso de teste. Em contraste, o Agent Compiler registra entre 3 e 4 ativações, enquanto o LangGraph se mantém próximo de 1.

Esses resultados sugerem que o CrewAI utiliza uma abordagem mais complexa e possivelmente redundante, o que pode explicar sua latência mais elevada. Por outro lado, o LangGraph e o Agent Compiler exibem um processo de execução mais enxuto, potencialmente contribuindo para uma melhor eficiência operacional.

5.1.1.3. Ativações

As Figuras 3 e 4 apresentam, respectivamente, a distribuição do número de ativações e a distribuição das latências registradas para cada framework no contexto de atendimento ao cliente.

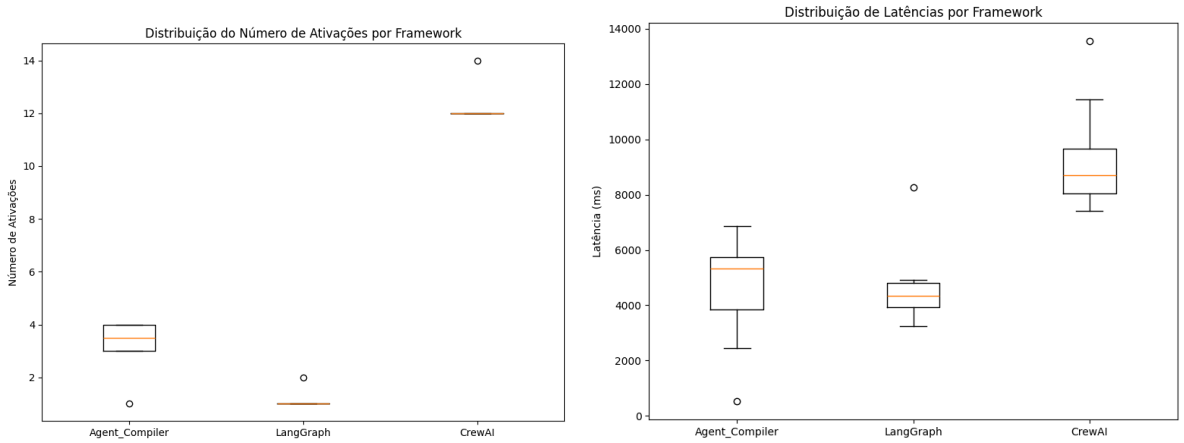

Figura 6: Distribuição do número de ativações e latências por framework.

Observa-se que o Agent Compiler apresenta uma distribuição bastante concentrada, com o número de ativações variando entre 3 e 4 e latências médias situadas entre 4.000 ms e 6.000 ms, indicando consistência e eficiência operacional. Em contraste, o LangGraph demonstra uma média de ativações muito baixa (próxima de 1) e latências similares às do Agent Compiler, sugerindo uma abordagem enxuta, embora seu comportamento de resposta possa ter outras implicações funcionais. Já o CrewAI exibe uma dispersão maior tanto no número de ativações — com valores que chegam a 14 — quanto nas latências, que se estendem de aproximadamente 9.000 ms a 12.000 ms, o que aponta para um processamento mais complexo e potencialmente redundante, afetando sua escalabilidade e eficiência em cenários de alta demanda.

5.1.2. Análise Qualitativa

A análise qualitativa foi conduzida a partir de diferentes cenários de atendimento ao cliente, enfatizando a precisão das respostas fornecidas e o registro dos agentes e ferramentas utilizados. No cenário de “Rastreamento de Pedidos”, por exemplo, o Agent Compiler forneceu uma resposta condizente com o resultado esperado – “Olá! Localizei seu e-mail referente à consulta sobre seu pedido. Seguem as informações: – Situação do Pedido: Por favor, aguarde de 3 a 5 dias úteis para a chegada do seu pedido. Em caso de dúvidas adicionais ou necessidade

de ajuda, fique à vontade para perguntar!” – acionando com eficácia o agente de ingestão de e-mail e o agente de escalonamento. Em contraste, o CrewAI acionou um número significativamente maior de agentes, evidenciando um fluxo de execução mais complexo e possivelmente redundante, o que eleva a latência sem, necessariamente, aprimorar a acurácia da resposta.

Nos cenários de “Consulta de Assinatura” e “Solicitação de Reembolso”, verificou-se que tanto o Agent Compiler quanto o LangGraph recorreram de maneira consistente ao agente de base de conhecimento (FAQ) para obter informações confiáveis, enquanto o agente de análise de sentimento auxiliou na adequação do tom das mensagens e na decisão de escalonamento quando necessário. Em um exemplo de “Solicitação de Suporte Urgente”, a resposta final apontou o encaminhamento para um operador humano, demonstrando a capacidade dos agentes em reconhecer situações emergenciais. Sob uma perspectiva comparativa, observou-se que frameworks que otimizam o fluxo de acionamento – em especial o Agent Compiler e o LangGraph – geraram respostas de elevada qualidade, aliadas a uma estrutura de comunicação mais concisa e eficiente. Já no caso do CrewAI, embora apresente uma organização reconhecidamente estruturada, sua maior redundância nas ativações sugere impactos potenciais na eficiência operacional do sistema, reforçando, portanto, a importância de uma gestão criteriosa dos agentes em ambientes de atendimento ao cliente.

5.2. **Dataset de Pesquisa**

O dataset de pesquisa utilizado é composto por dez casos de teste padronizados, abrangendo cenários comuns em tarefas acadêmicas e técnicas relacionadas a recuperação e manipulação de informações científicas. Os cenários incluem operações de recuperação de artigos científicos, extração de textos não selecionáveis, sumarização de conteúdo extenso, classificação temática dos artigos, entre outros.

5.2.1. **Análise Quantitativa**

5.2.1.1. Latência

Para a avaliação comparativa de latência entre os três frameworks (Agent Compiler, LangGraph e CrewAI), utilizou-se o conjunto de dez casos de teste relacionado a operações como “Paper Retrieval”, “Text Extraction”, “Summarization”, entre outros. A Tabela 1 apresenta as latências registradas em cada caso de teste.

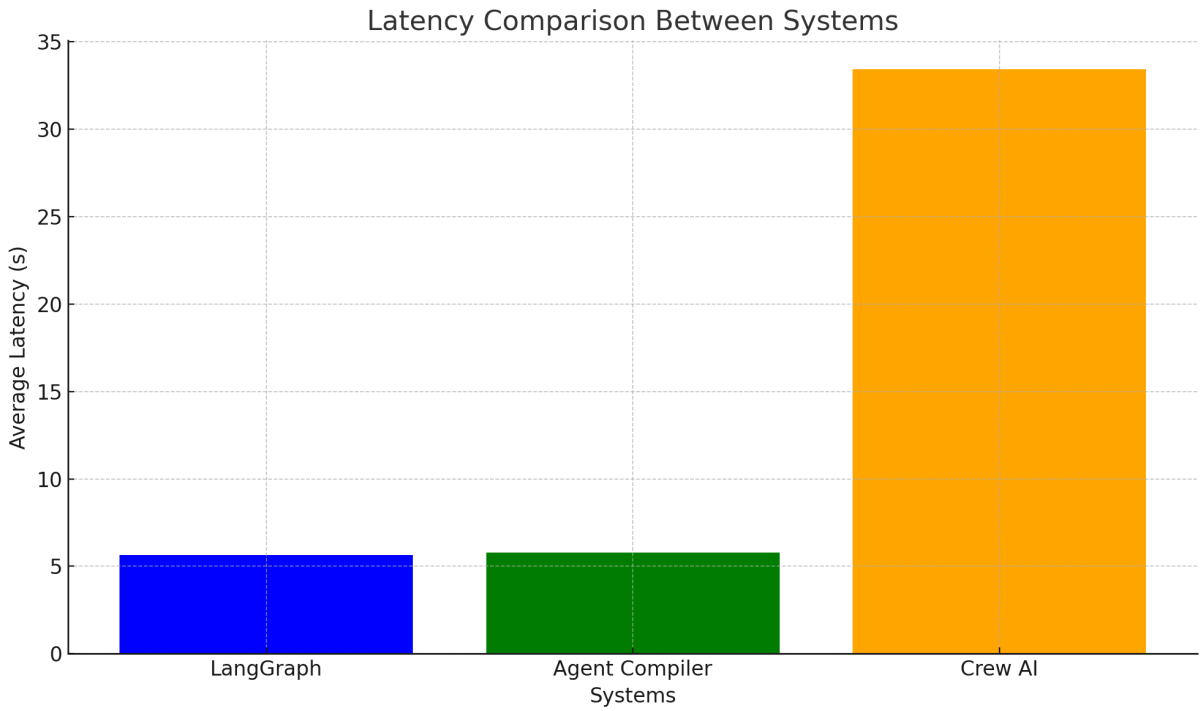

Figura 7: Comparação de Latência Média por Framework

Conforme observado no gráfico, o sistema LangGraph apresentou uma latência média de aproximadamente 5,64 segundos, enquanto o Agent Compiler registrou cerca de 5,78 segundos. Esses valores indicam que os dois sistemas têm tempos de resposta bastante próximos, com uma leve vantagem para o LangGraph. Em contrapartida, o Crew AI teve uma latência média significativamente mais elevada, de aproximadamente 33,43 segundos, o que pode ser indicativo de uma maior complexidade em seu processo interno de orquestração de agentes

5.2.1.2. Ativações

Outro parâmetro relevante para analisar a eficiência é a quantidade de ativações de agentes necessárias para atender às solicitações. Os resultados obtidos mostram que o Agent Compiler possui uma média de 3,7 ativações por teste, enquanto o LangGraph destaca-se com a menor média, apresentando apenas 0,9 ativações por teste. Em contrapartida, o Crew AI apresentou uma média muito

superior, com cerca de 28,9 ativações por teste.

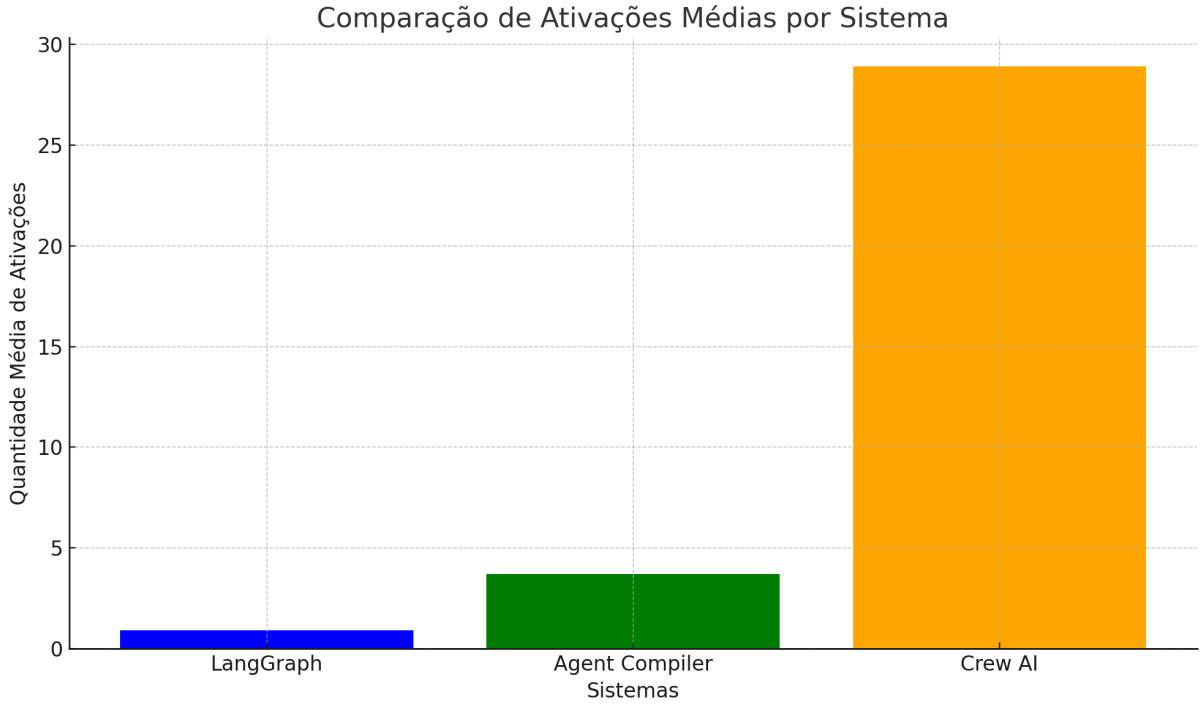

Figura 8: Comparação de Ativações Médias por Framework

Observa-se, portanto, que tanto o Agent Compiler quanto o LangGraph requerem relativamente poucas ativações para processar as solicitações, com destaque significativo para o LangGraph, que apresentou o menor índice médio. Já o Crew AI, devido à alta média de ativações, demonstra uma maior subdivisão das tarefas entre diversos agentes e módulos. Tal comportamento pode proporcionar maior flexibilidade na resolução de problemas complexos, mas também pode resultar em maior sobrecarga e complexidade operacional.

5.2.2. **Análise Qualitativa**

A análise qualitativa do dataset foi conduzida a partir de diversos cenários de testes, com ênfase na precisão das respostas fornecidas, na coerência dos fluxos de acionamento dos agentes e na consistência das ferramentas empregadas. No cenário do dataset de pesquisa, por exemplo, observou-se que o Agent Compiler produziu respostas claras e objetivas, listando de forma organizada os títulos, fontes e URLs dos artigos, o que demonstra uma efetiva integração entre os agentes de coleta e os mecanismos de apresentação dos dados. Em contraste, o CrewAI acionou um número maior de agentes durante o processo, resultando em um fluxo de execução mais complexo e redundante, o que, embora garanta uma cobertura ampla das etapas, pode impactar a clareza final da resposta.

Nos testes de “Text Extraction” e “Summarization”, tanto o LangGraph quanto o Agent Compiler apresentaram resultados consistentes, demonstrando uma capacidade de processamento integrada e uma comunicação eficiente entre os

módulos responsáveis pela extração e síntese do conteúdo dos artigos. Tal comportamento evidencia a adequação dos frameworks na execução de tarefas que demandam precisão textual e organização dos dados extraídos. Por outro lado, o CrewAI, apesar de oferecer uma estrutura robusta, evidencia, através do elevado número de ativações e do maior tempo de resposta, uma abordagem que pode carecer de uma otimização mais refinada, contribuindo para a complexidade operacional sem aprimorar, necessariamente, a acurácia ou a concisão da resposta final.

De forma geral, a análise qualitativa ressalta que frameworks que otimizam o fluxo de acionamento e evitam redundâncias tendem a gerar respostas de elevada qualidade, associadas a uma estrutura comunicacional mais direta e eficiente. Já o CrewAI, mesmo demonstrando uma arquitetura bem estruturada, sugere a necessidade de uma gestão mais criteriosa dos agentes, a fim de reduzir a sobrecarga de operações e melhorar a performance global do sistema.

6. CONCLUSÃO

Neste trabalho, endereçamos o problema de integração e eficiência em sistemas multi-agentes de tal forma a melhorar o planejamento e a orquestração dos fluxos de execução. Outras pesquisas já tinham tentado resolver este problema, porém não conseguiram implementar uma solução escalável e adaptável que atendesse de forma eficaz às demandas simultâneas das tarefas. Assim, nós propusemos a abordagem Plan and Solve que, usando o método de avaliação focado na eficiência dos fluxos de execução e na coerência na orquestração entre os agentes, se mostrou boa nos seguintes aspectos: organização, escalabilidade e adaptabilidade às necessidades específicas de cada tarefa.

6.1. **Considerações finais**

A implementação de agentes em sistemas multi-agentes demonstrou potencial para lidar com diversas tarefas de forma organizada e escalável. Ao longo deste trabalho, observou-se que a adoção de estratégias de planejamento (Plan and Solve) pode contribuir significativamente para a eficiência dos fluxos de execução, tornando a orquestração entre os agentes mais coerente e adaptável às necessidades de cada tarefa. Apesar de ainda existirem desafios, a perspectiva de integrar abordagens mais avançadas de planejamento e colaboração sugere um caminho promissor para o aperfeiçoamento dos sistemas multi-agentes.

6.2. **Limitações**

O estudo aqui desenvolvido não esgota todas as técnicas de compiladores e otimizações potenciais que poderiam ser aplicadas para acelerar ainda mais o processo de execução dos sistemas multi-agentes. Além disso, aspectos como tratamento de conflitos entre agentes ou análise de desempenho em cenários de larga escala não foram explorados em profundidade. Esses pontos podem

influenciar a adoção das soluções propostas em contextos mais complexos e demandam investigações adicionais.

6.3. **Trabalhos Futuros**

Há uma gama de oportunidades para expandir os resultados obtidos. A figura 9, que foi capturada durante a elaboração deste trabalho, remete à introdução de um novo modelo de Process Implementation na plataforma da CrewAI, incluindo a abordagem Consensual Process (Planned). Isso reforça que o paradigma de Plan and Solve vem sendo discutido e deve ser incorporado em futuros desenvolvimentos, sinalizando a evolução do Agent Compiler em direção a processos mais colaborativos e democráticos na tomada de decisões.

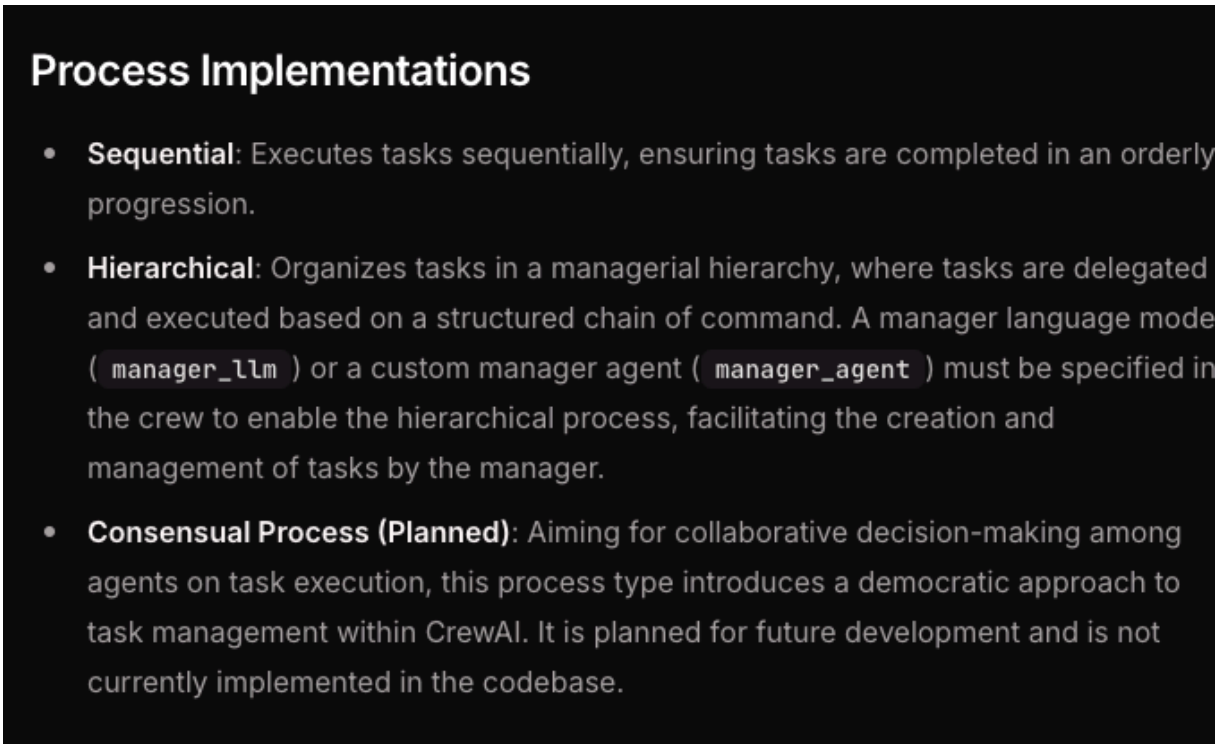

Figura 9: Pipeline de agentes disponíveis no site da CrewAI.

Embora não tenham sido exploradas todas as possibilidades das técnicas de compiladores nem toda a flexibilidade do plano do orquestrador, que poderiam agilizar ou aprimorar a execução dos agentes, foi possível identificar uma perspectiva interessante ao comparar essa abordagem de Agent Compiler com estruturas como o Supervisor do LangChain e o CrewAI. Investigar novas formas de compilação e orquestração, bem como avaliar o impacto dessas inovações em cenários de maior complexidade, constitui uma linha de pesquisa futura com potencial para expandir a aplicabilidade e a eficiência dos sistemas multi-agentes.

REFERÊNCIAS

[1] WANG, Lei; XU, Wanyu; LAN, Yihuai; HU, Zhiqiang; LAN, Yunshi; LEE, Roy Ka-Wei; LIM, Ee-Peng. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. Disponível em:https://arxiv.org/abs/2305.04091. Acesso em: 04 abr. 2025.

[2] KIM, Sehoon; MOON, Suhong; TABRIZI, Ryan; LEE, Nicholas; MAHONEY, Michael W.; KEUTZER, Kurt; GHOLAMI, Amir. An LLM Compiler for Parallel Function Calling. In: Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 2024. Disponível em:https://arxiv.org/abs/2312.04511. Acesso em: 04 abr. 2025.

[3] GUO, Taicheng; CHEN, Xiuying; WANG, Yaqi; CHANG, Ruidi; PEI, Shichao; CHAWLA, Nitesh V.; WIEST, Olaf; ZHANG, Xiangliang. Large Language Model based Multi-Agents: A Survey of Progress and Challenges. Disponível em:https://arxiv.org/abs/2402.01680. Acesso em: 04 abr. 2025.

[4] CREWAI. CrewAI. Disponível em:https://crew.ai. Acesso em: 04 abr. 2025.

[5] LANGGRAPH. LangGraph. Disponível em:https://langgraph.com. Acesso em: 04 abr. 2025.

[6] CREWAI. Process Class Detailed Overview. Disponível em:https://docs.crewai.com/concepts/processes#process-class-detailed-overview. Acesso em: 04 abr. 2025.

[7] WOOLDRIDGE, Michael. An Introduction to MultiAgent Systems. 2. ed. Indianapolis: John Wiley & Sons, 2009.

[8] ANTROPIC. Building Effective Agents. Disponível em:https://www.anthropic.com/engineering/building-effective-agents. Acesso em: 04 abr. 2025.

[9] LANGCHAIN-AI. LangGraph Supervisor Py. Disponível em:https://github.com/langchain-ai/langgraph-supervisor-py. Acesso em: 04 abr. 2025.

[10] JIN, et al. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. 2024. Disponível em:https://arxiv.org/abs/2408.02479. Acesso em: 04 abr. 2025.

[11] VAILLANT, et al. Developers' Perceptions on the Impact of ChatGPT in Software Development: A Survey. 2024. Disponível em:https://arxiv.org/abs/2405.12195. Acesso em: 04 abr. 2025.

[12] MCRAE, Tanner. Rethinking AI Agents: Why a Simple Router May Be All You Need. Medium, mar. 2024. Disponível em: https://medium.com/@tannermcrae/rethinking-ai-agents-why-a-simple-router-may-be-all-you-need-c95031c2d397. Acesso em: 04 abr. 2025.