UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

ENGENHARIA OPERACIONAL DE AGENTES BASEADOS EM LLM

Observabilidade, Avaliação e Otimização de Sistemas Multiagentes

Carlos Henrique R. de Jesus

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

CARLOS HENRIQUE RODRIGUES DE JESUS

Engenharia Operacional de Agentes Baseados em LLM

Observabilidade, Avaliação e Otimização de Sistemas Multiagentes

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): CARLOS HENRIQUE RODRIGUES DE JESUS

Título do trabalho: Engenharia Operacional de Agentes Baseados em LLM

Observabilidade, Avaliação e Otimização de Sistemas Multiagentes

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Carlos Henrique Rodrigues De Jesus**, **Discente**, em 04/02/2026, às 17:24, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 13/03/2026, às 11:23, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5956320** e o código CRC **956DE109**.

Referência: Processo nº 23070.005482/2026-11 SEI nº 5956320

CARLOS HENRIQUE RODRIGUES DE JESUS

Engenharia Operacional de Agentes Baseados em LLM

Observabilidade, Avaliação e Otimização de Sistemas Multiagentes

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Orientador: Prof. Dr. Fernando Marques Federson

Goiânia

2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UFG.

JESUS, CARLOS HENRIQUE RODRIGUES DE
Engenharia Operacional de Agentes Baseados em LLM [manuscrito]: Observabilidade, Avaliação e Otimização de Sistemas Multiagentes / CARLOS HENRIQUE RODRIGUES DE JESUS. - 2025.
161 f.: 2025

Orientador: Prof. Dr. Fernando Marques Federson
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2025.

1. Inteligência Artificial. 2. Agentes Inteligentes. 3. Agentops.

I. Federson, Fernando Marques , orient. II. Título.

CDU 004

CARLOS HENRIQUE RODRIGUES DE JESUS

Engenharia Operacional de Agentes Baseados em LLM

Observabilidade, Avaliação e Otimização de Sistemas Multiagentes

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 09 de dezembro de 2025.

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

Prof. Dr. Anderson da Silva Soares
Coordenador do BIA (INF-UFG)

Prof. Dr. Sávio Salvarino Teles de Oliveira
(INF-UFG)

CARLOS HENRIQUE RODRIGUES DE JESUS

Engenharia Operacional de Agentes Baseados em LLM

Observabilidade, Avaliação e Otimização de Sistemas Multiagentes

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Engenharia Operacional de Agentes Inteligentes**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: Inteligência artificial; Agentes inteligentes; Agentops.

ABSTRACT

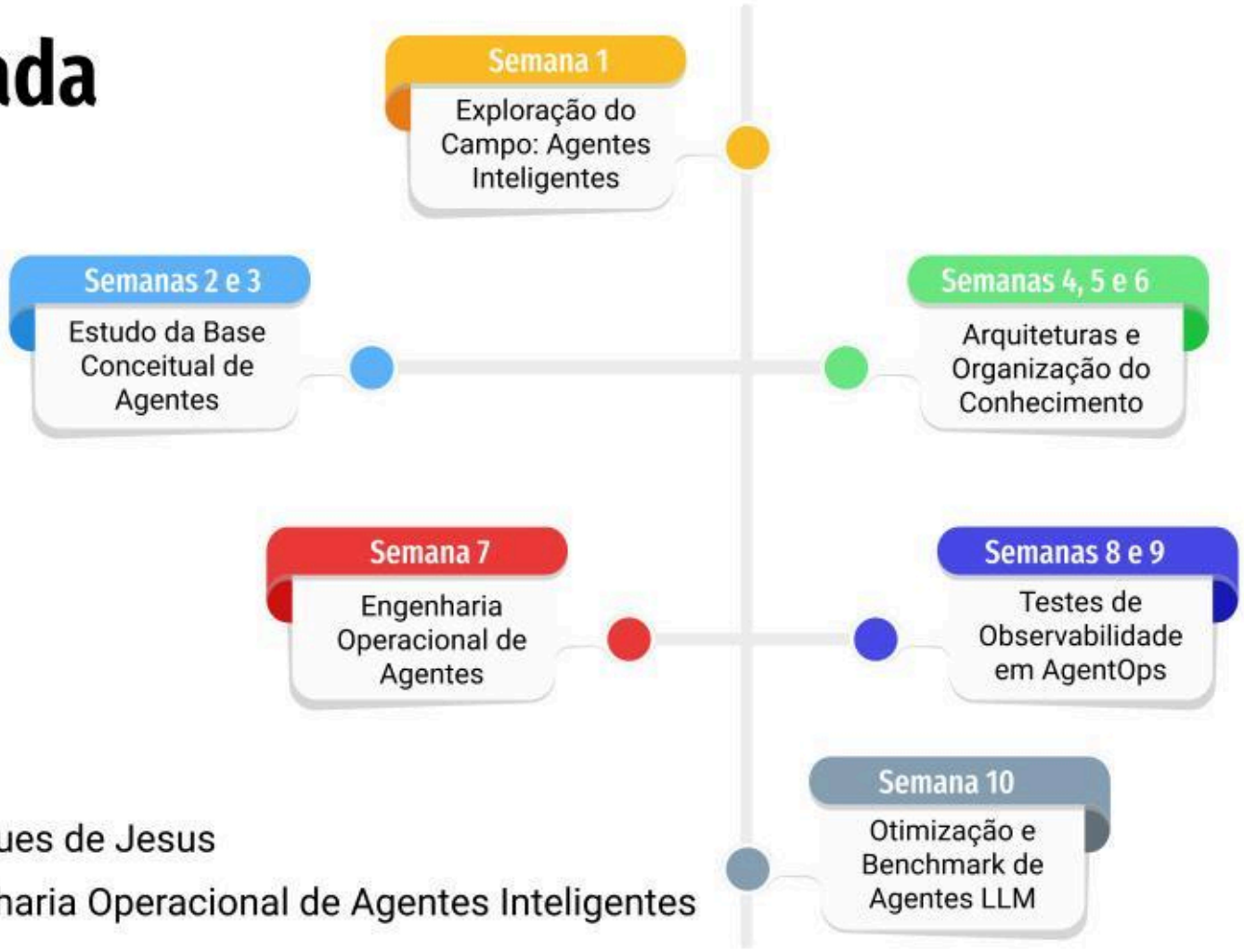
This Course Completion Report aims to bring together the results of my journey to become an expert in **Operational Engineering of Intelligent Agents**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

Keywords: Artificial intelligence; Intelligent agents; Agentops.

Goiânia

2025

Minha Jornada



Carlos Henrique Rodrigues de Jesus

Especialista em: Engenharia Operacional de Agentes Inteligentes

MINHA JORNADA

Nome: Carlos Henrique Rodrigues de Jesus

Especialidade: Engenharia Operacional de Agentes Inteligentes

Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

Minha Jornada

Minha jornada começou, na **Semana 1**, ainda com muitas dúvidas sobre qual recorte seguir dentro do universo da Inteligência Artificial. A partir da leitura da chamada do CASI'2025, fui explorando as trilhas do congresso, me concentrando em, *Intelligent Agents, Natural Language Processing, Agent Technologies for Data Mining, Agent-Based Modeling in Machine Learning Systems, Cognitive Modeling, Visualization and Big Data Analytics, Multimodal Learning Systems*, quase como quem passeia por um "cardápio" de possibilidades. Foi nesse percurso que **Intelligent Agents** começou a se destacar. Percebi que essa trilha dialogava diretamente com a minha curiosidade sobre sistemas que não apenas "respondem", mas percebem, decidem e agem. Além dessa curiosidade, sempre fiquei muito fascinado com a ideia do processo de automatização e comunicação de agentes, principalmente vendo possíveis "mundos" de trabalho com sistemas únicos e inteligentes, como em jogos e em outras séries da ficção. Em paralelo, organizei uma planilha de pesquisa científica para registrar cada artigo, autor e tipo de contribuição, e iniciei um documento de estudo de tópicos para anotar conceitos e insights que iam

[1] Dez Semanas, entre setembro de 2025 e dezembro de 2025.

surgindo. Ao final da **Semana 1**, eu já não tinha só um tema escolhido, mas um pequeno "kit de navegação": tema definido, primeiras referências mapeadas e um espaço estruturado para ir construindo conhecimento ao longo da jornada. Todos esses materiais (planilha de referências e estudo inicial de tópicos) estão reunidos no **Apêndice 1**.

Nas **Semanas 2 e 3**, o foco esteve em transformar esse recorte inicial em fundamentação teórica consistente. Busquei novos materiais para ampliar a bibliografia, com destaque para textos e artigos recentes, publicados em **março de 2025**, que discutem a evolução histórica dos agentes e aprofundam a distinção entre agentes baseados em regras e abordagens mais autônomas e contextuais, aproximando-me da noção de IA agêntica. Em paralelo, aprofundei o estudo clássico da PUCRS sobre **Arquitetura de Agentes**, publicado em **abril de 2001**, que ajudou a organizar a evolução das principais abordagens arquiteturais, desde modelos mais simples até estruturas cognitivas e híbridas. A partir desse conjunto de leituras, comecei a diferenciar, com mais precisão, agentes que apenas reagem a estímulos daqueles que percebem o contexto, raciocinam e adaptam seu comportamento. Fui consolidando esse conhecimento em documentos próprios: elaborei um guia sobre a evolução dos agentes (que depois serviu de base para a construção de uma *timeline*), outro sobre conceitos fundamentais e glossário de termos recorrentes na área de agentes inteligentes, e um terceiro sobre "aplicação lógica vs. generativa", comparando agentes apoiados em lógica simbólica com aqueles que utilizam LLMs para tomada de decisão. Essas produções estão no registro detalhado das leituras na planilha de pesquisa científica e estão organizadas no **Apêndice 2**.

As **Semanas 4, 5 e 6** marcaram um ponto de virada. Deixei de apenas "consumir" referências para começar a reorganizar o campo à minha maneira. Na **Semana 4**, concluí as leituras sobre arquiteturas de agentes e produzi um estudo de tópico mais robusto, em que mapeei desde arquiteturas clássicas – como Agentes Reflexivos Simples, Agentes Reflexivos com Estado, Agentes Baseados em Objetivos, Agentes Baseados em Utilidade, arquiteturas baseadas em estados mentais e arquiteturas de referência como M-DRAP e ADEPT. A partir dessa base, construí uma classificação em camadas (arquiteturas de controle, por nível social, por tipo de tarefa ou domínio e por tipo de programa de agente) e também uma classificação hierárquica (reativas, deliberativas/cognitivas, híbridas, sociais,

baseadas em estados mentais). Para visualizar tudo isso, desenhei um diagrama hierárquico que funcionou quase como um “mapa de metrô” das arquiteturas, permitindo enxergar conexões e fronteiras entre os modelos.

Nas **Semanas 5 e 6**, comecei a conectar arquitetura com prática. Listei frameworks de agentes de IA usados na indústria, organizando-os por relevância e tipo de aplicação (orquestração, automação, integração com LLMs, etc.), entrei de vez no universo de **AgentOps** a partir do *Survey on AgentOps* e de materiais introdutórios da IBM. Da combinação de tudo isso, nasceu um roadmap conceitual em 11 níveis, que começa em programação e prompting, passa por noções básicas de agentes, LLMs & APIs, uso de ferramentas, frameworks de agentes, orquestração, memória, RAG, deployment e monitoramento, até chegar em segurança e governança. Esse *pipeline* passou a ser meu “roteiro mental” para pensar soluções com agentes baseados em LLMs. Sempre que avaliava uma nova ferramenta ou arquitetura, eu me perguntava em qual nível daquele roadmap ela se encaixava. Os estudos de tópicos sobre arquiteturas, a classificação, os diagramas e o roadmap das **Semanas 4, 5 e 6** estão descritos no **Apêndice 3**.

A **Semana 7** foi o momento em que o tema da jornada ganhou um nome próprio: “Engenharia Operacional de Agentes (AgentOps)”. Até então, eu vinha estudando agentes, arquiteturas e ferramentas. Nessa **Semana**, passei a olhar para a “vida real” dos agentes em produção – como projetar, orquestrar, observar, avaliar e governar esses sistemas de ponta a ponta. Estruturei meu estudo em cinco pilares: (1) Projeto/Criação de Agentes, (2) Orquestração, (3) Operação/Observabilidade, (4) Avaliação e (5) Governança. Para sair do plano abstrato, decidi aplicar os padrões do *AgentOps Pattern Catalogue* em um cenário concreto. Usei o SDK do AgentOps para instrumentar o framework multiagente MetaGPT, que simula uma empresa de software com diferentes “papéis” de agentes. Foi como conectar um monitor cardíaco a um organismo vivo (mas não tinha “vida” de fato). Comecei a registrar chamadas de LLM, custos, uso de tokens e métricas de execução, visualizando estes dados em um dashboard próprio de observabilidade. Em paralelo, assisti a uma apresentação da Google que “costura” DevOps, MLOps e AgentOps, e fiz um estudo de tópicos extraindo as arquiteturas e padrões que poderiam inspirar sistemas de agentes mais robustos. Ao final da **Semana 7**, eu já não via AgentOps apenas como um “assunto interessante”, mas como o eixo central da minha especialização, agora apoiado em

experimentos reais com o MetaGPT e em dados concretos exibidos no painel do AgentOps. Os registros das execuções, as capturas de dashboard e o estudo das arquiteturas inspiradas pela palestra da Google estão consolidados no **Apêndice 4**.

Nas **Semanas 8 e 9**, a jornada ganhou duas camadas importantes: observabilidade aprofundada e avaliação sistemática de agentes LLM. Na **Semana 8**, mergulhei no artigo "AgentOps: Enabling Observability of LLM Agents (2024)", que discute sistemas autônomos baseados em LLMs capazes de perceber contexto, raciocinar, planejar e executar fluxos de trabalho usando ferramentas externas e bases de conhecimento. A partir dessa leitura, revisitei meus experimentos com o MetaGPT e o AgentOps, buscando entender melhor o que significava "observabilidade" na prática, ou seja, não apenas ver logs, mas construir uma visão coerente da trajetória do agente: quais passos deu, quais ferramentas usou, onde errou, quanto custou. Em paralelo, fiz um levantamento de aproximadamente 17 ferramentas de observabilidade para agentes e LLMs, organizando em um estudo de tópico que passou a funcionar como um "catálogo pessoal" de soluções para monitorar sistemas inteligentes.

Na **Semana 9**, o foco se deslocou para *evaluation & benchmarking*. A partir da leitura do survey "Evaluation and Benchmarking of LLM Agents (2025)", que propõe uma taxonomia bidimensional (objetivos de avaliação e processo de avaliação), construí um plano de ação experimental em quatro camadas para testar agentes LLM com o MetaGPT: (1) instrumentação base do ambiente com MetaGPT, OpenTelemetry e AgentOps e criação de um dataset de cerca de 100 perguntas; (2) implementação de um agente com RAG, executando consultas sobre um conjunto específico de conhecimento; (3) análise e otimização, usando dashboards do AgentOps para enxergar latência, custo e taxa de sucesso, e explorando técnicas de otimização de prompts como DSPy e AutoPDL; (4) avaliação e *benchmarking*, comparando versões do agente (baseline, instrumentada e otimizada) com métricas de trajetória (como *exact match* e *any-order match*) e de resposta final (taxa de acerto, uso correto de ferramentas, eficiência). Nesse período, também iniciei a reprodução do plano de ação em outro framework, o TheAgentCompany, que mede desempenho de agentes LLM em tarefas profissionais do mundo real. Apesar de avanços na fase de instrumentação, limitações práticas (imagens Docker muito grandes, execução lenta e problemas de compatibilidade) levaram à decisão consciente de concentrar os esforços no MetaGPT, que se mostrou mais viável para experimentação interativa. O mapeamento das

ferramentas, o plano de ação detalhado e os registros comparativos dessas **Semanas 8 e 9** estão organizados no **Apêndice 5**.

Por fim, a **Semana 10** foi dedicada a transformar esse planejamento em um cenário experimental completo, agora com tarefas de desenvolvimento de software de ponta a ponta. Se até ali, eu avaliava agentes em tarefas mais pontuais, nessa **Semana** passei a olhar para projetos inteiros. Selecionei cinco aplicações como dataset de teste. No modelo original, eram 100 tarefas utilizando o MetaGPT original, mas justamente para evitar o alto custo, optei por seguir com apenas 5 tarefas robustas de criação de ferramentas: (1) uma *Todo List* em CLI, (2) uma API de encurtamento de URL, (3) uma *Weather CLI* que consulta uma API de clima, (4) um gerador de senhas seguras e (5) um conversor de Markdown para HTML. Esses projetos foram pensados para ter escopos claros, mas variados o suficiente para exigir análise de requisitos, arquitetura, implementação, testes e documentação. Em seguida, implementei uma equipe multiagente com CrewAI, composta por: um *Product Manager* (responsável pelo PRD), um *Software Architect* (desenho da arquitetura), um *Software Engineer* (implementação do código), um *QA Engineer* (planos e casos de teste) e um *Technical Writer* (documentação técnica e guias de uso). Para isso, utilizei a mesma estrutura do MetaGPT. Uma observação que considero importante: aqui, a ideia, como no fluxo das **Semanas** anteriores, era ainda avaliar o MetaGPT na prática, mas com um interesse em entender todo processo de orquestração. Alguns empecilhos na hora de executar o repositório original, como versões descontinuadas, me impediram de continuar esse processo na versão original. Optei, então, por refazer toda a estrutura do MetaGPT utilizando o framework CrewAI, tornando então o MetaGPT em uma nova versão, reconstruída utilizando o CrewAI com adição de algumas ferramentas. Conectei essa arquitetura a um sistema de RAG para enriquecer o contexto dos agentes com exemplos e boas práticas, e também utilizei o DSPy para experimentar otimizações de prompts, comparando versões manuais e otimizadas. Na parte da avaliação, consolidei um conjunto de métricas inspirado em boas práticas de plataformas como o Vertex AI: taxa de conclusão das tarefas, completude dos artefatos (por exemplo, se o projeto foi entregue com código, testes e documentação), qualidade do output final, correção e eficiência no uso de

ferramentas, além de métricas operacionais como latência por projeto, uso de tokens e custo estimado. Todas as métricas podem ser observadas pelo dashboard do AgentOps.

Uma das descobertas foi perceber como cada configuração do sistema ocupou um "lugar" diferente no equilíbrio entre custo, tempo e qualidade. Ao comparar os três cenários (**sem RAG**, **com RAG** e **com RAG + DSPy**), ficou claro que o baseline **sem RAG** cumpriu o papel de referência "bruta". O sistema funcionava e entregava os projetos, mas exigia mais iterações, consumia mais tokens e produzia artefatos menos completos, sobretudo em termos de arquitetura, testes e documentação. A versão **com RAG** se mostrou um meio-termo bastante eficiente: ao enriquecer o contexto antes da geração, o sistema passou a responder mais rápido, com menos chamadas ao modelo e às ferramentas, mantendo 100% de sucesso nas tarefas e entregando resultados tecnicamente melhores que o baseline inicial, o que a torna especialmente atraente para cenários em que tempo de resposta e simplicidade operacional são prioridade. Já a versão **com RAG + DSPy,** se comportou quase como uma equipe que "prepara o terreno" antes de agir, ou seja, prompts mais estruturados, instruções mais detalhadas e uso intenso de contexto recuperado levaram a PRDs mais completos, arquiteturas mais bem justificadas, códigos mais alinhados com os requisitos, planos de teste mais ricos e documentações mais claras. Mesmo com mais chamadas ao LLM e maior uso de tokens, o uso de um modelo mais econômico permitiu reduzir significativamente o custo por projeto, sobretudo quando se projeta o experimento para dezenas ou centenas de tarefas, tornando essa configuração a melhor em termos de custo-benefício em larga escala. Em contrapartida, essa sofisticação trouxe maior tempo de execução e mais camadas de orquestração, reforçando a ideia de que ganhos de qualidade e eficiência econômica podem exigir maior complexidade operacional. No fim, o experimento da **Semana 10** não apenas comparou três configurações técnicas, mas contou uma história sobre escolhas de projeto. Observei que quando a prioridade é velocidade, a abordagem **com RAG** tende a ser a mais interessante; quando o objetivo é maximizar qualidade e aproveitar melhor cada dólar em tokens, a combinação **RAG + DSPy** se destaca, especialmente em cenários de *batch processing* e alto volume. Um ponto importante é que, em todos os casos, o sistema multiagente se manteve estável, com 100% de sucesso nas tarefas, o que reforça a robustez da arquitetura construída. As definições

desses cenários de teste, os registros das execuções e a tabela de ferramentas por camada usados na **Semana 10** compõem o **Apêndice 6**.

Em função de tudo que vivi nesta **Jornada**, reconheço que o começo foi difícil, mas observar o comportamento dos agentes, seus fluxos, decisões, custos e rastros de execução, despertou uma curiosidade que me levou de uma simples dúvida inicial a uma proposta estruturada de **Engenharia Operacional de Agentes (AgentOps)**. Sou grato aos meus pais, à minha avó, aos docentes e aos amigos da faculdade André, Dayane, Edward, Letícia, Michael e Lisandra pelo apoio em cada **Semana** desse processo. Este documento registra o caminho percorrido, mas também marca um ponto de partida. Ainda há muito a aprender e a transformar em impacto real, e sinto que estou apenas no começo.

APÊNDICE 1

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 3 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Durante a primeira Semana, foram realizadas as seguintes atividades:

1. Estudo de interesse sobre áreas de aplicação e pesquisa, utilizando como referência o Congresso CSCE'25, foram utilizados três conferências sendo elas ICAI'25, ICDATA'25 e ACC'25, dentro delas foram selecionados os seguintes tópicos:
   - **Intelligent Agents**
   - **Natural Language Processing**
   - **Agent Technologies for Data Mining**
   - **Agent-Based Modeling in Machine Learning Systems**
   - **Cognitive Modeling, Visualization, and Big Data Analytics**
   - **Multimodal Learning Systems**
2. Definição do tema a ser trabalhado durante o processo:
   - **Intelligent Agents**
3. Pesquisa de artigos, trabalhos científicos e papers sobre o assunto, foi realizado uma pesquisa inicial sobre o tema abordado, com o foco em me aprofundar na base teórica:
   - Pesquisa Científica
4. Através dos materiais de pesquisa, foi então gerado um estudo de tópicos relacionados a fundamentos e história do tema:
   - Estudo de tópico - Agentes de IA

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Aprofundar conhecimento em **Agentes de IA**, sobre:

1. Evolução dos agentes;
2. Arquiteturas de agentes;

3. Aplicações lógicas e generativas de agentes;

O foco é acrescentar uma base de conhecimento ao nosso estudo de tópico.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Pesquisa e Construção de Referência:

Ao iniciar esta pesquisa, parte de uma curiosidade prática: como projetar e operar agentes de IA que sejam, ao mesmo tempo, úteis e confiáveis em cenários reais. A partir dessa pergunta inicial, a jornada começou com um mergulho na literatura clássica sobre agentes inteligentes e sistemas multi-agente, passando por modelos reativos, deliberativos, arquiteturas BDI e diferentes formas de coordenação entre agentes. Em seguida, o foco se deslocou para trabalhos recentes sobre *AI* Agents e frameworks modernos de construção de agentes baseados em LLM, que introduzem novos desafios de observabilidade, avaliação e operação contínua. Esse percurso bibliográfico não apenas mapeia o estado da arte, mas também revela lacunas e oportunidades, especialmente no que diz respeito à chamada *AgentOps*: que orientam as próximas etapas desta investigação.

Artigo: "Intelligent Agents: Theory and Practice”

A primeira menção amplamente reconhecida do termo em artigo científico é atribuída ao final da década de 1980/início de 1990;

- Década de 1980, o conceito de “agentes de software” aparece em alguns artigos de sistemas distribuídos e eng. de software, usando o termo “Agentes” como "processo autônomo”.

Mas a **definição formal** e aceita veio mesmo com:

- Jennings, N. R., & Wooldridge, M. J. (1995). "Intelligent Agents: Theory and Practice". Knowledge Engineering Review, 10(2), 115–152.
- Michael Wooldridge e Nicholas R. Jennings são os principais responsáveis por consolidar a definição, tornando se referência clássica.

Pós, o surgimento da definição formal, Russel & Norving (1995), definem no livro *Artificial Intelligence: A Modern Approach,* o'que é um agente inteligente - Sistema que percebe o ambiente através de sensores e age sobre ele com atuadores de forma racional). Passando a ser uma definição padrão.

Segundo o paper, o termo “Agente” possui as seguintes características:

- Autonomia: Agentes operam sem a intervenção humana direta, tendo alguns tipos de controle sobre suas ações e estado interno.
- Habilidade Social (Interação): Agentes tem a capacidade de interagir com outros agentes (ou seres humanos) via linguagem de comunicação entre agentes, como os LLMs.

- Reatividade: Agentes percebem seu ambiente, respondendo de forma oportuna às mudanças que ocorrem nele.
    - O ambiente pode ser o mundo físico, interface gráfica, outros agentes, a Internet, ou a combinação de todos eles. Como vemos atualmente nos agentes generativos.
- Pró-atividade: Agentes não simplesmente agem em resposta ao seu ambiente, eles são capazes de exibir comportamento dirigido por objetos tomando iniciativa.

O termo ‘agente’ para aqueles que trabalham com IA, tem um significado mais forte e específico. Entendem como:

- Sistema computacional que, além de ter as propriedades identificadas (Autonomia, Habilidade Social, Reatividade e Pró-atividade), é conceituado ou implementado usando conceitos que são mais usualmente aplicados a humanos.
- Por exemplo, é comum em IA caracterizar um agente usando noções mentalísticas, como conhecimento, crença, intenção e obrigação.

O paper menciona atributos adicionais que são discutidos no contexto de agentes:

- Mobilidade: A habilidade de se mover através de uma rede eletrónica. Podendo se classificado como mobilidade de acesso a informações na rede.
- Veracidade: A suposição de que um agente não comunicara conscientemente informações falsas.
- Benevolência: A suposição de que agentes não tem objetivos conflitantes, todo agente sempre tentará fazer oque foi pedido.
- Racionalidade: A suposição de que um agente agirá para alcançar seus objetivos, e não agirá de forma a impedir que seus objetivos sejam alcançados.

Livro: Artificial Intelligence - A Modern Approach

No capítulo 2 do livro “Artificial Intelligence - A Modern Approach” de Stuart Russel e Peter Norving, agentes de IA são definidos como qualquer sistema que percebe seu ambiente por meio de sensores e age através de atuadores. Os autores explicam que os agentes podem ser Humanos, Robôs ou programas de software; um agente de software, por exemplo, recebe entradas com teclas ou pacotes de rede e age escrevendo arquivos ou enviando pacotes. Podemos trazer isso hoje para os Agentes inteligentes que tem motores generativos, que recebem prompts de entrada e extraem conhecimento através de ferramentas/tools e Inferência em APIs.

Cada percepção instantânea é chamada de percepto, e a sequência completa de perceptos forma o histórico do agente. O comportamento desse agente é formalizado por uma função

de agente, que mapeia qualquer sequência de perceptos para uma ação; Essa função é implementada por um programa de agente, conceito distinto do modelo abstrato.

- Racionalidade: Agente racional é aquele que escolhe ação esperada para maximizar um medidor de desempenho. A racionalidade depende do que o agente sabe, das percepções disponíveis e das consequência de suas ações.
- Natureza dos ambientes, são apresentadas várias classificações:
    - Totalmente observável versus parcialmente observável;
    - Único agente versus multiagente;
    - Determinístico versus estocástico;
    - Episódico versus sequencial;
    - Estático versus dinâmico;
    - Discreto versus contínuo;
- Um ambiente observável fornece ao agente toda informação relevante para tomada de decisão, enquanto em um ambiente parcialmente observável o agente deve manter um estado interno para lidar com a incerteza.
- Ambientes multiagentes podem ser cooperativos ou competitivos; ambientes estocásticos tratam a incerteza de resultados em termos probabilísticos.

Estrutura de agentes, apresentam-se diferentes arquiteturas:

- Agentes reflexos simples: escolhem ações com base no percepto atual, ignorando todo o histórico. Por exemplo, o aspirador de pó que suga se a sala está suja e se move caso contrário. Arquitetura simples funciona bem em ambientes totalmente observáveis.
- Agentes reflexivos baseados em modelo: mantém um estado interno para lidar com ambientes parcialmente observáveis; Precisa de um modelo de como evolui e de como suas ações o afetam.
- Agentes baseados em objetivos: além do estado, consideram metas desejadas para escolher ações.
- Agentes baseados em utilidade: Função de utilidade, que quantifica o quanto cada estado é desejável; sendo assim o agente escolhe a ação que maximiza a utilidade esperada. Permitindo lidar com ambientes incertos e conflitos entre metas, pois considera trade-offs entre agilidade, segurança e outros critérios.

Dessa forma, prosseguiremos para as próximas etapas de pesquisa do tema central, com essa base consolidada.

APÊNDICE 2

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 10 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Durante a segunda Semana, foram realizadas as seguintes atividades com o foco em agregar conhecimento teórico e fundamental sobre a área de Agentes de IA.

- Coleta de novos materiais para construção de uma bibliografia mais ampla, entre eles os principais estudados foram:
  - Blog: A evolução histórica dos agentes de inteligência artificial: Do conceito à aplicação - Escrito por Roberto Dias Duarte.
    - Desde a fundamentação teórica, dada do nascimento da IA, até a era dos LLMs e IA Agêntica, publicado em março de 2025.
    - Novo termo aprendido: IA Agêntica.
  - Blog: Agentes de IA vs. IA agente/agêntica compreendendo diferenças, publicado em março de 2025 - Escrito por Lori Macvittie.
    - Focado principalmente em diferenciar os termos existentes.
    - Agentes de IA são sistemas orientados por regras projetados para executar tarefas específicas com base em entradas e objetivos predefinidos.
    - A Agentic AI introduz a autonomia e adaptação contextual. Projetada para perceber, raciocinar e agir de forma independente, determinando dinamicamente o melhor curso de ação com base no ambiente.
  - Arquitetura de Agentes, publicado em abril de 2001, revisão bibliográfica feito pela PUCRS - Escrito por Murilo Juchem e Ricardo Melo Bastos.
    - Estudo que mostra e reflete a arquitetura dos primeiros agentes, desde arquiteturas cognitivas e deliberativas, até arquiteturas híbridas.
- Esses novos materiais foram adicionados à planilha, visando principalmente coletar uma boa base de conhecimento.
  - Pesquisa Científica
- Foi então realizado a agregação de materiais nos documentos já criados, criando guias novas, para cada estudo:
  - Evolução dos agentes, repercutindo toda sua fase histórica.

- Estudo de tópico - Agentes de IA
- Estudo sobre conceitos fundamentais, lista de termos e palavras-chaves comuns na área de agentes de IA.
- Estudo de tópico - Agentes de IA
- Estudo sobre aplicação lógica vs generativa.
- Estudo de tópico - Agentes de IA

- Timeline sobre a evolução dos agentes
  - Evolução dos Agentes de IA.pdf

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Terminar de ler a revisão bibliográfica da PUC sobre arquitetura de agentes.
Construir diagrama de arquiteturas existentes.
Explorar o conceito de simulação de agentes em larga escala.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Fundamentação Teórica em Agentes Inteligentes:

Definição básica: Um **agente inteligente** percebe seu ambiente, toma decisões e age de forma autônoma para alcançar objetivos específicos e aprimora seu desempenho através de **aprendizado ou aquisição de conhecimento**.

Definição de Jennings e Wooldridge (1995): Estes autores introduziram uma definição formal de agente inteligente como *"um sistema computacional situado em algum ambiente, capaz de ação autônoma e flexível nesse ambiente para cumprir seus objetivos de projeto. Sistema que percebe o ambiente através de sensores e age sobre ele com atuadores de forma racional "*. Flexibilidade significa que o agente pode **reagir a mudanças**, agir **proativamente** e **interagir socialmente** com outros agentes ou humanos. Além disso, uma síntese posterior da mesma definição ressalta que agentes inteligentes também possuem **atitudes mentais** como intenções, crenças e objetivos.

- **Características essenciais**: Autonomia, habilidade social (comunicação com outros agentes e humanos), reatividade (resposta em tempo real ao ambiente) e proatividade (comportamento orientado a objetivos) diferenciam agentes de programas tradicionais.
- **Função da IA:** A IA é descrita como o estudo e o projeto de agentes inteligentes; portanto, compreender agentes é fundamental para entender IA.

Agentes versus IA Agêntica (agentic AI)

AI agents (agentes de IA) – São softwares contextuais que tomam decisões e executam tarefas sem intervenção constante. Utilizam memória e lógica para atingir **metas definidas por humanos**. Exemplos no mundo corporativo incluem agentes de suporte (Moveworks), agentes de vendas e operações (triagem de emails, agendamento etc.).

Agentic AI (IA agêntica) – Refere-se a sistemas que **estabelecem e reavaliam suas próprias metas**, planejam como alcançá-las e adaptam suas ações conforme o contexto. Esses sistemas apresentam características de planejamento, adaptação e tomada de decisão semelhantes ao pensamento humano. Podem reagir a falhas de API, escolher adiar tarefas com base em prioridades ou mudar de ferramenta conforme o comportamento do usuário.

- **Diferença principal** – Enquanto **agentes de IA seguem scripts ou fluxos definidos**, **IA agêntica replana e reordena objetivos de forma autônoma**. Portanto, nem todo agente autônomo é agêntico; a distinção recai sobre a capacidade de *autodefinição de metas* e *replanejamento*.

Fundamentos: Evolução, Origens e Nascimento da IA (1940-1956)

Autômatos Cibernéticos Pioneiros

Elmer e Elsie (c. 1948-1950) – O neurofisiologista **W. Grey Walter** construiu duas "tartarugas cibernéticas" chamadas *Elmer* e *Elsie*. Esses robôs usavam **fotocélulas giratórias** para seguir luz e sensores de toque para desviar de obstáculos. Se a luz fosse muito forte, recuavam; se colidiam com algo, invertiam a direção. Esses autômatos mostravam comportamento exploratório e foram apelidados de *Machina speculatrix*.

- **Importância**: Demonstravam que **comportamentos "animais" básicos poderiam emergir de circuitos analógicos simples**, pré-figurando conceitos de reatividade e auto-recarregamento (conseguiram até localizar suas próprias bases de recarga). Apesar de exibir comportamentos complexos para a época, não possuíam memória ou capacidade de aprendizado.

O Johns Hopkins Beast (1960s)

- Criado no Laboratório de Física Aplicada da Universidade Johns Hopkins, o **Beast** era um automato móvel construído com dezenas de **transistores e circuitos analógicos** (não utilizava computador digital). Possuía **fotocélulas e sonar** para navegar, buscava tomadas pretas nos corredores para se recarregar e usava portas **NOR** implementadas com transistores para executar lógica booleana.
- Seu sistema de sensores incluía interruptores físicos para detectar tomadas, escadas e obstáculos, e transdutores ultrassônicos que permitiam uma espécie de orientação "à moda dos morcegos". Esses robôs demonstravam **autonomia de sobrevivência**, precursoras dos agentes reativos.

Dartmouth Workshop (1956) e os Primeiros Programas

Convergência de ideias, na década de 1940 e início de 1950, avanços em neurologia, cibernética (Norbert Wiener) e teoria da computação (Claude Shannon) sugeriam que máquinas poderiam imitar o cérebro humano. Estas influências culminaram no *Workshop de Dartmouth* de 1956, onde **John McCarthy, Marvin Minsky, Claude Shannon** e outros cunharam o termo "inteligência artificial".

- Pesquisadores como **H. A. Simon** e **Marvin Minsky** previram, na década de 1960, que máquinas totalmente inteligentes seriam construídas em 10 a 20 anos.
- **Primeiros sucessos**, com programas como o **General Problem Solver** (Newell & Simon), o **Geometry Theorem Prover** (Gelernter) e o **SAINT** (Slagle) demonstraram que computadores podiam resolver problemas de álgebra, provar teoremas

geométricos e até aprender inglês. O sistema **STUDENT** de Daniel Bobrow solucionava problemas de álgebra descritos em linguagem natural, e **ELIZA** (1966) simulava conversas de psicoterapia com respostas predefinidas, sendo o primeiro "chatbot".

Evolução: Fases Principais de Desenvolvimento e Arquiteturas

Fase 1 – IA Simbólica e Sistemas Baseados em Regras (1956-1974)

Abordagem de busca: Os primeiros programas tratavam o raciocínio como um **problema de busca em árvore**. Utilizavam heurísticas para reduzir o espaço de procura e backtracking para retornar quando ficavam sem saída.

- **Expertise simbólica** – Sistemas como *STRIPS* (Stanford) planeavam ações para o robô **Shakey** usando regras lógicas. Ao mesmo tempo, novas representações de conhecimento surgiram, como **redes semânticas** (Quillian) e **scripts/frames** (Minsky, Schank) para modelar conhecimentos do senso comum.
- **Limitações** – A promessa de resolver qualquer problema foi frustrada por problemas como **explosão combinatória**(muito caminhos possíveis) e **paradoxo de Moravec** (tarefas "fáceis" para humanos, como visão e movimento, eram difíceis para algoritmos). A escassez de poder computacional e de dados de conhecimento comum levou à **primeira "AI winter"**: críticas sobre promessa exagerada e cortes de financiamento na década de 1970.

Fase 2 – Especialistas, Lógica e Agentes Deliberativos (1974-2000)

Prolog e Lógica Programada: Nos anos 1970, Robert Kowalski e pesquisadores franceses criaram o **Prolog**, linguagem baseada em **Horn clauses** (regras) que permitia **raciocínio dedutivo eficiente**. Essas regras sustentaram os **sistemas especialistas** de Edward Feigenbaum (como **DENDRAL** e **MYCIN**), que diagnosticavam doenças ou identificavam compostos químicos.

- **Sistemas especialistas**: Popularizados na década de 1980, esses programas restringiam-se a domínios específicos e foram amplamente usados por empresas. O sistema **R1** economizava milhões em configuração de computadores na Digital Equipment Corp. e o mercado de AI cresceu de alguns milhões em 1980 para **bilhões de dólares em 1988**.
- **Revival dos neurônios artificiais**: Nos anos 1980, o **Hopfield net** provou que redes neurais simples podiam aprender e convergir. Geoffrey Hinton e David Rumelhart popularizaram o **algoritmo de retropropagação (backpropagation)**, revivendo o interesse por redes neurais e conectando-as à aprendizagem supervisionada.

- **Arquiteturas deliberativas (BDI e PRS)**: Inspiradas em modelos cognitivos humanos, as arquiteturas **Belief-Desire-Intention (BDI)** representam explicitamente as **crenças**, **desejos** (objetivos) e **intenções** (planos em execução) de um agente. O **Procedural Reasoning System (PRS)**, desenvolvido na SRI na década de 1980, usa esse modelo; o intérprete mantém crenças sobre o mundo, escolhe objetivos e seleciona planos (knowledge areas), intercalando planejamento e execução. Isso superava abordagens puramente reativas, pois os agentes podiam reconsiderar planos em tempo real.

Fase 3 – Aprendizado, Autonomia e Sistemas Multimodais (2000-2017)

Aprendizado de máquina: A explosão de dados e a queda dos custos computacionais nos anos 2000 permitiram que agentes incorporassem **aprendizado de máquina** para melhorar desempenho. Técnicas como redes neurais profundas e *reinforcement learning* geraram avanços em visão, tradução e robótica.

- **Sistemas multiagente (MAS)** – Problemas complexos passaram a ser resolvidos por **conjuntos de agentes cooperativos**. Um MAS é um sistema descentralizado em que vários agentes interagem; cada agente possui **autonomia, visão local e nenhum controle central**, e a cooperação pode levar a **auto-organização e emergências**. Tais sistemas são ideais para otimizar logística, simulações sociais e jogos, pois a combinação de agentes simples pode resultar em comportamentos sofisticados.
- **Agentes adaptativos** – Com dados e algoritmos mais robustos, surgiram agentes **aprendizes** capazes de ajustar estratégias e incorporar percepção multimodal (texto, imagem, áudio). Esses agentes não apenas reagiam, mas **aprendiam e se adaptavam continuamente**.

Fase 4 – Era dos LLMs e IA Agêntica (2017-Presente)

Grandes Modelos de Linguagem (LLMs): A introdução de transformadores e modelos como GPT-3, GPT-4 e similares permitiu que agentes compreendessem e gerassem linguagem natural de forma surpreendente. Pesquisas em **chain-of-thought** (COT) e **tree-of-thought** mostraram que incentivar o modelo a articular raciocínios intermediários melhora significativamente o desempenho em tarefas complexas.

- **Agentes ReAct e frameworks de memória**: Técnicas como **ReAct** alternam entre "pensar" e "agir", permitindo que o modelo planeje etapas, execute ações (por exemplo, chamada de API) e reflita sobre observações. Métodos como **Reflexion** adicionam mecanismos de memória de curto e longo prazo, enquanto sistemas de vetorização externa armazenam fatos para consultas futuras.

- **AutoGPT e BabyAGI** – Exemplos de agentes autônomos de 2023/2024. O **AutoGPT** é um framework open-source que usa um grande modelo de linguagem para **dividir objetivos em subtarefas**, gerar prompts internos, executar ações via ferramentas e adaptar-se com base em feedback. Ele coordena o LLM, integrações modulares de ferramentas e laços de crítica interna, permitindo planejamento e execução autônomos, embora ainda exija supervisão humana para evitar erros ou loops infinitos. Possui módulos para **memória de curto e longo prazo**, **decomposição de tarefas**, mecanismo de auto-questionamento e integração com APIs e código.
- **CrewAI e multi-agentes modernos**: **CrewAI** é um framework de 2024/2025 que cria uma "tripulação" de agentes de IA, cada um com papéis, objetivos e habilidades específicos. Ele permite **planejamento dinâmico**, **delegação de tarefas**, **comunicação inter-agente**, estruturas hierárquicas, resolução de conflitos e simulação de cenários. As principais vantagens incluem especialização por papéis, escalabilidade, modularidade e aprendizado adaptativo, sendo aplicado em gestão de projetos, pesquisa científica, simulações urbanas e análise de mercado.

Comparação agentic AI vs AI agents: *Artigos recentes enfatizam que agentes de IA são componentes que executam tarefas específicas, enquanto a IA agêntica é um sistema mais amplo capaz de definir prioridades, manter memória persistente e reavaliar objetivos. Os agentes de IA são previsíveis; a IA agêntica reestrutura ações de acordo com o contexto.*

Essa diferença implica desafios de **governança e ética**, pois sistemas agênticos precisam de limites claros para evitar comportamentos inesperados.

Taxonomias de Agentes: Reativos, Deliberativos e Aprendizes

- **Agentes reativos:** Respondem diretamente aos estímulos sem memória interna; são rápidos e simples, baseados em regras de ação imediata (ex.: tartarugas de Grey Walter).
- **Agentes deliberativos (planejadores)**: Mantêm uma representação interna do mundo e elaboram planos (e.g., BDI, PRS). São capazes de replanejar frente a mudanças no ambiente.
- **Agentes utilitários e de aprendizado**: Alguns agentes maximizam uma função utilidade (balançando riscos e recompensas), enquanto **agentes de aprendizado** melhoram com experiências passadas; técnicas como **reinforcement learning** e redes profundas tornaram-nos predominantes nas últimas décadas.

Sistemas Multiagente e Colaboração

- **Definição** – Um **sistema multiagente (MAS)** é composto por vários agentes que interagem para resolver problemas que seriam difíceis para um único agente.

- **Características** – Autonomia parcial dos agentes, visão local limitada, descentralização (sem controlador central) e potencial para auto-organização.
- **Benefícios** – Os agentes podem cooperar ou competir, compartilhando conhecimento via protocolos de comunicação para alcançar sinergias que emergem de comportamentos individuais simples.
- **Perspectivas futuras** – Com a integração de LLMs e frameworks como CrewAI, sistemas multi-agente prometem coordenação sofisticada em escala, abrindo caminho para novas aplicações em logística, saúde, finanças e governança.

Transformação: IA Agêntica e Considerações Éticas

IA Agêntica: Expande o conceito de agente ao permitir que o sistema **defina seus próprios objetivos**, planeje passos intermediários e adapte-se em tempo real.

- **Desafios de controle**: Empresas devem ponderar entre **inteligência (adaptação)** e **previsibilidade (controle)**. Sistemas agênticos exigem limites e mecanismos de supervisão para evitar decisões erradas ou danos. Perguntas essenciais envolvem se o sistema pode reordenar prioridades, manter memória estratégica e modificar metas.
- **Aplicações**: IA agêntica pode otimizar atendimento ao cliente, orquestrar pipelines de desenvolvimento de software, gerenciar cadeias de suprimentos ou até coordenar equipes de agentes em jogos. Entretanto, para adoção empresarial, agentes convencionais (previsíveis) muitas vezes são preferíveis devido à necessidade de confiança e auditoria.

Termos e palavras chaves

Termos básicos

- **Agente de IA (AI Agent)**: sistema capaz de perceber, raciocinar e agir de forma autônoma para atingir objetivos, tanto em software (chatbots) quanto em hardware (robôs).
- **Agente autônomo (Autonomous Agent)**: opera sem supervisão constante; planeja e executa passos rumo a uma meta, como um agente que conclui tarefas com AutoGPT.
- **Ambiente (Environment):** mundo externo com o qual o agente interage; pode ser físico (ruas, fábricas) ou digital (bancos de dados, APIs).
- **Ação (Action)**: passo que o agente realiza para influenciar o ambiente, como enviar um e-mail ou mover um braço robótico.
- **Atuador (Actuator)**: mecanismo que executa a ação; em robôs são motores, em software são chamadas de função.
- **Percepto/Sensor (Percept):** entrada de sensores que permite ao agente perceber o ambiente em um momento; inclui imagens de câmeras, dados de GPS ou leituras de velocidade.
- **Objetivo (Goal)**: resultado que o agente busca, orientando o planejamento e a tomada de decisão.
- **Função utilidade (Utility Function):** mecanismo que avalia as ações e ajuda a escolher a que maximiza os resultados desejados.
- **Planejamento (Planning)**: processo de sequenciar ações para atingir um objetivo; é crucial em tarefas que exigem previsão e não apenas reação.
- **Modelo do mundo (World Model):** representação interna usada para simular e prever resultados, permitindo ao agente planejar com maior eficácia.
- **Estado de crença (Belief State):** estimativa do agente sobre a situação atual, especialmente quando as informações são incompletas.
- **Heurística (Heuristic):** atalho de raciocínio que simplifica a busca por soluções, usando “bom senso” para reduzir o esforço computacional.
- **Grafo de conhecimento (Knowledge Graph)**: rede de dados que mapeia relações entre conceitos, funcionando como um mapa mental para o agente.
- **Ontologia (Ontology):** dicionário estruturado de conceitos e relações que ajuda o agente a compreender e raciocinar sobre o ambiente.
- **Sistema baseado em regras (Rule-Based System)**: mecanismo em que decisões seguem regras pré-definidas (do tipo “se-então”), como um semáforo que muda de cor em determinados intervalos.

- **Deslocamento de modelo (Model drift)**: deterioração de desempenho quando o mundo muda; exige atualizar os modelos do agente.

Definição de agentes de IA

- **Autonomia (Autonomy)**: é a capacidade do agente de operar de forma independente, decidindo e atuando sem supervisão humana direta.
- **Habilidade social (Social ability)**: é a aptidão de interagir com outros agentes, sistemas ou humanos; necessária para cooperação, negociação e troca de informações.
- **Reatividade (Reactivity)**: define o talento do agente de perceber mudanças no ambiente e responder prontamente a elas, ajustando suas ações em tempo real.
- **Proatividade (Proactiveness)**: refere-se à capacidade de prever necessidades e agir com antecedência, sem esperar por estímulos externos.
- **Racionalidade (Rationality)**: descreve o comportamento de buscar a melhor ação possível, maximizando o desempenho esperado de acordo com os objetivos e a informação disponível.
- **Medida de desempenho (Performance measure)**: é o critério que avalia quão bem o agente atinge seus objetivos dentro de um determinado ambiente; integra o modelo PEAS (Performance measure, Environment, Actuators, Sensors).
- **Percepção /Sensor (Perception /Sensor)**: mecanismo pelo qual o agente obtém dados do ambiente; envolve sensores físicos ou virtuais que alimentam o estado de crença do agente.
- **Ação / Atuador (Action / Actuator)**: ação é qualquer intervenção do agente sobre o ambiente, enquanto o atuador é o mecanismo (motor, API, etc.) que realiza essa intervenção.
- **Ambiente (Environment)**: é o mundo físico ou digital no qual o agente está inserido e no qual suas percepções e ações têm efeito.
- **Base de conhecimento (Knowledge base)**: conjunto de informações que o agente possui sobre o ambiente e sobre si, usado como referência para tomar decisões.
- **Processo de decisão (Decision-making process)**: conjunto de algoritmos e regras que orientam o agente a escolher a próxima ação a partir da percepção, do conhecimento e dos objetivos.
- **Crenças (Beliefs – BDI)**: no modelo BDI, representam o que o agente sabe ou assume sobre o mundo, podendo incluir informações incompletas ou imprecisas.
- **Desejos (Desires – BDI)**: são os objetivos ou metas que o agente deseja alcançar, orientando sua atuação.
- **Intenções (Intentions – BDI)**: são os planos de ação que o agente escolhe e se compromete a seguir para concretizar seus desejos.

- **Aprendizado e adaptação (Learning and adaptation)**: característica de agentes que ajustam seu comportamento com base no feedback recebido, melhorando seu desempenho ao longo do tempo.

Padrões e técnicas intermediárias

- **Padrões de design de IA agêntica**: arquiteturas reutilizáveis para construir agentes, desde agentes reativos que respondem imediatamente até deliberativos que planejam antes de agir.
- **ReAct (Reasoning + Acting)**: técnica em que o agente intercala raciocínio e ação; ele pesquisa, executa, reflete sobre o resultado e itera.
- **ReWOO**: abordagem de "Raciocínio sem Observações" que permite ao agente tomar decisões baseadas em dados externos sem perceptos diretos.
- **Reflexão**: capacidade do agente de avaliar decisões passadas e ajustar estratégias, promovendo auto-melhoria.
- **Agentic RAG**: variação de geração aumentada por recuperação em que o agente faz consultas iterativas para refinar e corrigir respostas.
- **Uso de ferramenta (Tool use)**: habilidade de aproveitar APIs, calculadoras ou bancos de dados para ampliar as capacidades do agente.
- **Chamada de função (Function calling)**: mecanismo que permite ao agente invocar serviços externos de forma estruturada para realizar ações no mundo real.
- **Planejamento deliberativo**: método em que o agente avalia sequências de ações em direção a um objetivo, comparando caminhos possíveis.

Termos multi-agente e humanos no ciclo

- **Multi-agente**: configuração em que vários agentes interagem colaborando ou competindo para alcançar resultados complexos.
- **Sistema multi-agente (MAS)**: estrutura onde agentes coordenam em um ambiente comum; um exemplo é um sistema de semáforos inteligentes.
- **Comportamento emergente**: padrões complexos e inesperados que surgem das interações entre agentes sob regras simples.
- **Aprendizado federado**: método de treinamento descentralizado em que agentes contribuem para um modelo global sem compartilhar dados brutos.
- **Humano-no-ciclo (HITL)**: sistema onde humanos revisam ou aprovam decisões do agente em etapas críticas, complementando a autonomia.
- **Colaboração humano-agente**: parceria em que humanos e agentes combinam forças para realizar tarefas, como chefes e sous-chefs numa cozinha.
- **Transferência de controle**: mudança de comando entre o agente e o humano; por exemplo, um piloto tomando o controle do piloto automático.

- **Mecanismo de fail-safe**: procedimentos integrados que colocam o agente em estado seguro quando algo dá errado.
- **IA ética**: abordagem que busca garantir equidade, transparência e responsabilidade nos agentes.
- **Justiça algorítmica**: princípio de que os resultados do agente não devem introduzir ou reforçar preconceitos.
- **Inteligência de enxame**: fenômeno em que diversos agentes simples colaboram para resolver problemas complexos, inspirado em abelhas ou aves.

Termos avançados e frameworks

- **Módulos de memória**: componentes que permitem ao agente armazenar e recuperar informações, ajudando-o a aprender e a tomar decisões melhores.
- **Hierarchical Multiagent**: sistema no qual agentes se organizam em níveis; agentes superiores coordenam e delegam tarefas a agentes inferiores.
- **Agente simples reflex**: agente que reage diretamente ao estado atual sem memória ou modelo interno; funciona com regras “se-então”.
- **Agentforce**: orquestrador que coordena vários agentes, comparável a um maestro que sincroniza uma orquestra.
- **LLM (Large Language Model)**: modelo de linguagem que serve de “cérebro” do agente, interpretando entradas, gerando respostas e tomando decisões.
- **Ações de referência**: exemplos de ações que orientam a tomada de decisão do agente.
- **Ações padrão**: movimentos pré-definidos que o agente pode executar, como ferramentas básicas de sua caixa de ferramentas.
- **Tópicos padrão**: assuntos que o agente foi treinado para tratar, como perguntas frequentes ou análises de dados.
- **Ações do sistema**: operações internas realizadas pelo agente, como alocação de recursos ou agendamento.
- **Representação de conhecimento**: forma como o agente estrutura e armazena informações, agindo como um mapa cognitivo.
- **Singularidade**: hipótese de que a inteligência das máquinas possa superar a humana, gerando avanços tecnológicos imprevisíveis.
- **LangChain**: framework modular amplamente usado para construir agentes com LLM; integra prompts, APIs e memória para fluxos complexos.
- **AutoGen**: estrutura open-source da Microsoft que permite a colaboração conversacional entre agentes; agentes delegam tarefas e refinam resultados em diálogo.
- **CrewAI**: framework que orquestra grupos (“crews”) de agentes com papéis distintos, facilitando projetos cooperativos como geração de conteúdo ou análise de dados.

- **SmolAgents**: framework minimalista da Hugging Face para criar agentes rapidamente, com integração de ferramentas e foco na prototipagem.
- **Model Context Protocol (MCP)**: protocolo que conecta agentes e modelos a ferramentas e dados externos de forma padronizada, ampliando capacidades sem integrações sob medida.
- **Small Language Model**: modelo de linguagem compacto que possibilita agentes leves e eficientes em dispositivos com poucos recursos.
- **Tokens**: menores unidades de texto que o agente processa; limitam a quantidade de contexto que ele pode analisar.
- **Prompts**: instruções ou perguntas que definem o que o agente deve fazer; boas instruções melhoram a precisão das respostas.
- **Janela de contexto**: quantidade de informações anteriores que o agente pode recordar durante a interação; janelas maiores permitem conversas mais longas.
- **Alucinações**: casos em que o agente inventa fatos ou fornece informações incorretas; precisam ser mitigadas para garantir confiabilidade.
- **Temperatura**: parâmetro que controla a aleatoriedade das respostas: valores baixos geram saídas mais previsíveis, valores altos criam respostas mais criativas.
- **Chain-of-thought prompting**: técnica que orienta o agente a explicar seu raciocínio passo a passo, melhorando a resolução de problemas.
- **Framework de agente**: infraestrutura que fornece ferramentas para construir agentes com raciocínio, decisão e ação; exemplos incluem LangChain e AutoGen.
- **Fluxo de trabalho agêntico (Agentic workflow)**: modo de operação em que o agente planeja, executa e refina tarefas de forma autônoma para automação responsiva.
- **Langgraph**: framework para construir aplicações LLM multi-agentes com gerenciamento explícito de estado e transições estruturadas.
- **Memória de longo prazo:** armazenamento persistente que mantém informações ao longo de múltiplas sessões, permitindo ao agente recordar contextos antigos.
- **Memória de curto prazo**: armazenamento temporário que retém o contexto recente da conversa, com capacidade limitada e duração curta.

Aplicações de Agentes de IA

Aplicações lógicas dos agentes

Raciocínio com base em lógica simbólica: Os agentes lógicos representam fatos sobre o mundo numa base de conhecimento e usam um mecanismo de inferência para derivar conclusões. Um agente deste tipo armazena sentenças numa linguagem formal e aplica regras de inferência para responder consultas e escolher ações. Em um agente baseado em conhecimento, a função do agente inclui “Tell” (inserir novos perceptos), “Ask” (consultar a base de conhecimento) e “Tell” novamente para registrar a ação; ele precisa representar

estados, incorporar novos perceptos, atualizar seu modelo interno, deduzir propriedades ocultas e determinar a ação apropriada. Esse método é usado em sistemas de prova de teoremas, planejamento simbólico e jogos lógicos como Wumpus World.

- **Diagnóstico e tomada de decisão**: Ao empregar lógica formal e dedução, os agentes lógicos conseguem diagnosticar falhas ou prever consequências. Eles foram adotados em robótica e na verificação de software; na saúde, regras lógicas auxiliam em diagnósticos e recomendações clínicas.
- **Sistemas especialistas e assistentes virtuais**: Mecanismos de busca, assistentes de voz (por exemplo, Siri e Alexa) e sistemas de recomendação utilizam raciocínio baseado em conhecimento para consultar fatos e inferir respostas. Na área jurídica, sistemas lógicos redigem contratos ou verificam a conformidade de documentos.
- **Teste e verificação de software**: As bases de conhecimento e inferências são utilizadas para gerar casos de teste, provar propriedades e detectar inconsistências, aumentando a confiabilidade de sistemas críticos.

Aplicações generativas dos agentes

Simulação e prototipagem social: Pesquisadores de Stanford construíram “agentes generativos” que se comportam como personagens humanos. Cada agente recebe uma biografia resumida e utiliza um grande modelo de linguagem para lembrar interações, construir relações e planejar eventos; o resultado são simulações em que os agentes acordam, conversam e até organizam festas. Esses agentes podem ser usados em jogos para criar personagens não-jogadores mais críveis e em prototipagem social para testar intervenções ou políticas em ambientes seguros.

- **Pesquisa em ciências sociais**: Uma arquitetura de agentes generativos foi usada para simular mais de mil indivíduos, combinando transcrições de entrevistas com um LLM. As respostas desses agentes replicaram as respostas de participantes humanos em pesquisas sociais com 85 % de precisão. Essa abordagem permite testar mensagens de saúde pública, políticas econômicas ou intervenções de desinformação sem envolver pessoas reais.
- **Criação de conteúdo**: Os agentes generativos produzem novos textos, códigos, imagens, músicas ou design aprendendo padrões a partir de dados existentes. Em marketing e comunicação empresarial, esses agentes redigem posts, anúncios, e-mails e descrições de produtos, otimizando-os para diferentes públicos; também geram respostas personalizadas em chatbots de atendimento ao cliente. No entretenimento, escrevem roteiros, narrativas interativas e poemas.
- **Personalização e recomendações**: Artigos sobre AI agents destacam que esses sistemas podem gerenciar agendas, reservar viagens e oferecer recomendações personalizadas que se adaptam ao histórico do usuário. Uma aplicação emergente é

a *hiper-personalização*: agentes generativos criam experiências altamente personalizadas, recomendando conteúdos e serviços alinhados a preferências individuais.

- **Sistemas "auto-curativos" e manutenção preditiva**: Agentes generativos podem monitorar infra-estruturas e sistemas de TI, identificar problemas iminentes e executar correções automaticamente, reduzindo o tempo de inatividade e aumentando a resiliência dos serviços.
- **Automação de desenvolvimento de software**: Estes agentes estão começando a automatizar tarefas de desenvolvimento (por exemplo, escrever funções, documentar códigos e propor arquiteturas), abrindo espaço para paradigmas como DevOps autônomos. Ao permitir que pessoas sem experiência em programação criem aplicações por meio de interfaces naturais, os agentes generativos democratizam a criação de software.
- **Finanças e saúde**: No setor financeiro, agentes generativos podem executar negociações de forma autônoma analisando o mercado em tempo real e gerando relatórios financeiros. Na saúde, exemplos como o IBM Watson Health demonstram como esses agentes analisam dados médicos, auxiliam diagnósticos e sugerem planos de tratamento personalizados.

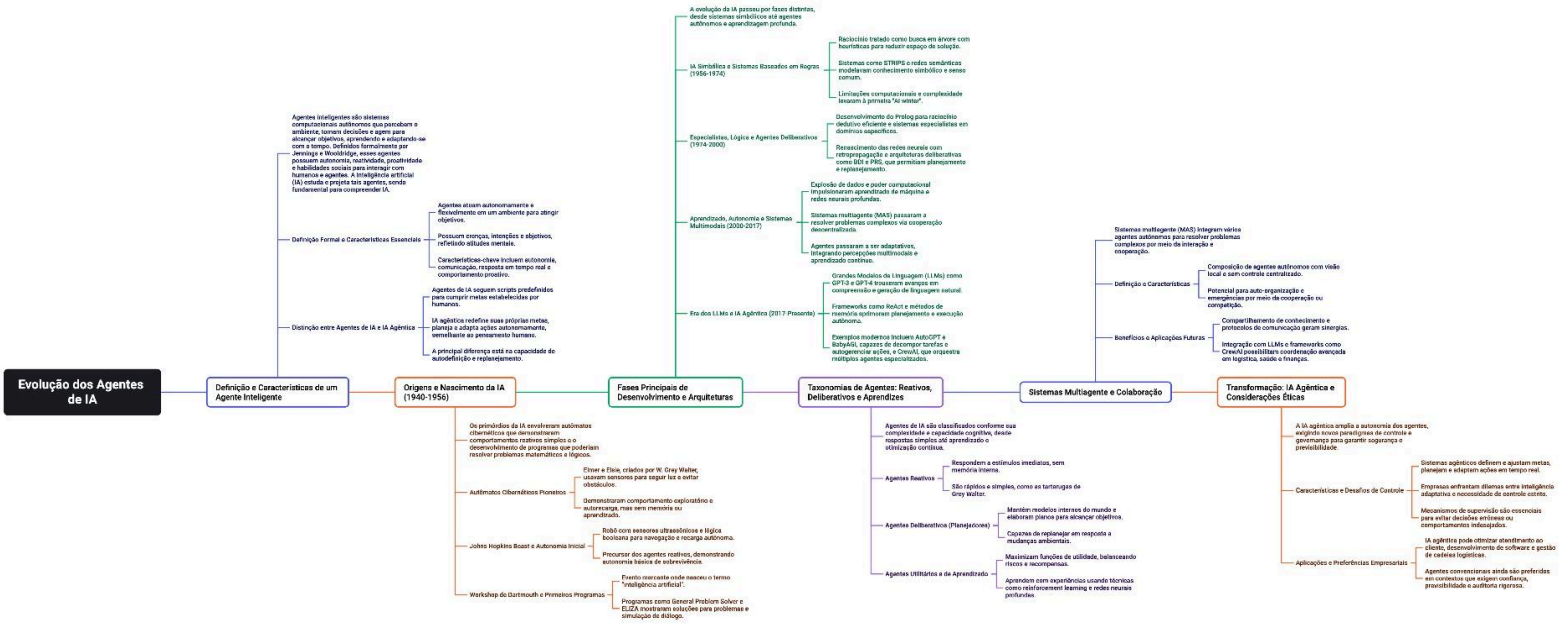


Imagem 1: *Diagrama da Evolução dos Agentes de IA*
Link: *Evolução dos Agentes de IA.pdf*

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 18 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Essa Semana com o intuito de me aprofundar mais sobre o tema de Agentes Inteligentes, concentrei principalmente o conhecimento sobre as arquiteturas de agentes inteligentes:

- Concentrei principalmente os estudos na referência bibliográfica de Arquiteturas de agentes de Murilo Juchem e Ricardo Melo Bastos da PUCRS.
- Referência Revisada.pdf
- Produzi uma guia nova no documento com as arquiteturas citadas no artigo, juntamente com a explicação dos modelos. (Revisitar depois)
- Estudo de tópico - Agentes de IA
- Encontrei dois surveys principais sendo eles:
  - Agentic AI: A Comprehensive Survey of Technologies, Applications, and Societal Implications
    - (PDF) Agentic AI: A Comprehensive Survey of Technologies, Applications, and Societal Implications
    - Comecei a ler o survey, com o foco de ter uma abrangência maior sobre o conhecimento na área.
  - A Survey on AgentOps: Categorization, Challenges, and Future Directions
    - A Survey on AgentOps
- Li o material que explicava bem por cima sobre AgentOps: What is AgentOps? | IBM

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Terminar estudo de arquiteturas de agentes;
Terminar de Ler o Survey sobre Agentic AI, produzir estudo de tópico em cima do survey;
Caso sinta que tenha base o suficiente, irei pesquisar mais sobre AgentOps.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Arquitetura de Agentes

Agente deliberativo é um agente que **raciocina antes de agir**. Ele mantém um **modelo interno do mundo** (crenças), persegue **metas explícitas** (desejos/objetivos) e escolhe **intenções/planos** por meio de busca e planejamento — clássico no paradigma **BDI (Beliefs-Desires-Intentions)**. Ele escolhe a ação que, com base na sequência de percepções recebidas e no conhecimento que tem, **maximiza o valor esperado de uma medida de desempenho**

- **Deliberativo:** orientado a metas de médio/longa duração, explicável, flexível; porém mais **custoso** e **lento** sob incerteza.
- **Reativo:** respostas rápidas por regras/estímulo-ação, mas sem planejamento nem justificativa global.

Modelo de Agente

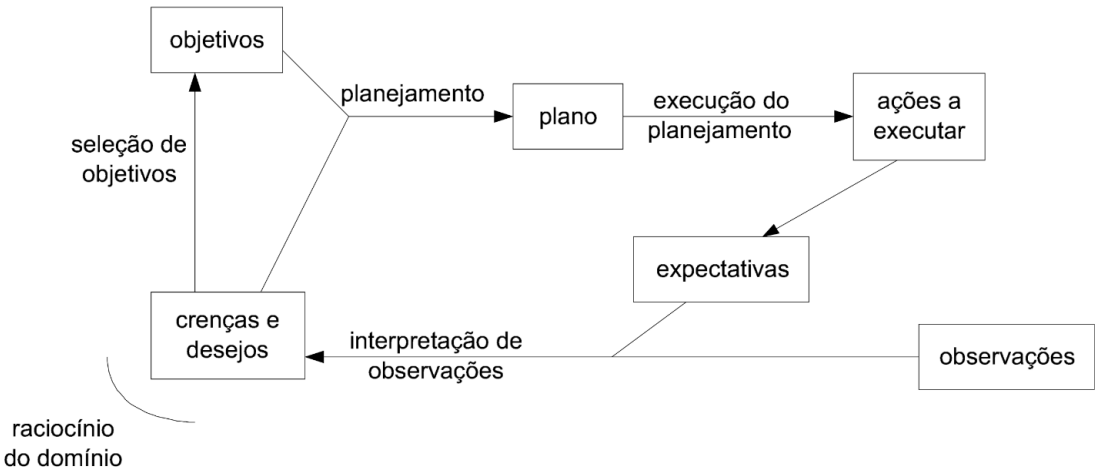


Imagem 2: *Modelo de agente*

Ela descreve um ciclo senso–deliberação–ação típico de agentes deliberativos (estilo BDI).

- **Crenças e desejos** → são os “estados mentais” do agente (o que ele acredita sobre o mundo e o que deseja). Eles são atualizados pelo bloco **interpretação de**

observações, usando o **raciocínio do domínio** (conhecimento específico do problema).

- **Seleção de objetivos** → a partir de crenças e desejos, o agente escolhe **objetivos** ativos (o que tentar alcançar agora). Esses estados mentais – crenças, objetivos, intenções, expectativas – são mencionados no texto logo abaixo da figura.
- **Planejamento** → **plano** → dado um objetivo, o agente elabora um **plano** (sequência de ações para atingir o objetivo).
- **Execução do planejamento** → **ações a executar** → o plano é executado no ambiente. O plano também gera **expectativas** (o que se espera observar se tudo estiver correndo bem), usadas para monitorar a execução.
- **Observações** → **interpretação de observações** → as ações produzem **observações** do ambiente; o agente as interpreta, compara com as **expectativas** e então **atualiza crenças/desejos**, podendo disparar replanejamento ou troca de objetivos. Isso fecha o laço de controle.

O fluxo é perceber (observações) → **entender** (interpretar e atualizar crenças) → **decidir** (selecionar objetivos) → **planejar** → **agir** → **monitorar via expectativas** → **aprender/ajustar** (volta às crenças/objetivos).

Essa visão está alinhada com a própria definição adotada no relatório: *"um agente é um sistema computacional situado em um ambiente, capaz de ações autônomas visando atingir seus objetivos"*.

Categorias de Agentes

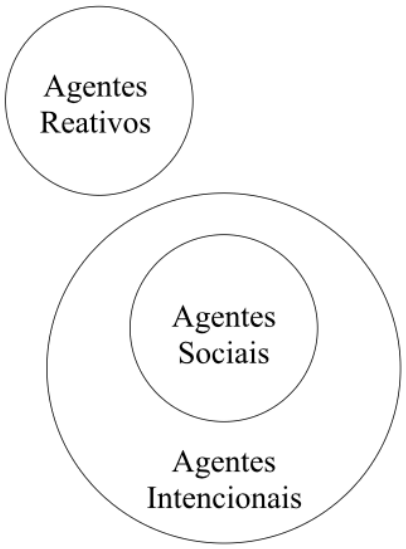


Imagem 3: *Categorias de Agentes*

A figura apresenta três **categorias de agentes** segundo a literatura:

- **Agentes reativos**: são agentes simples cujo comportamento é determinado por regras estímulo–resposta. Eles não mantêm um modelo explícito do mundo nem de seus objetivos; reagem diretamente aos estímulos ambientais e, por isso, "não necessariamente são racionais". Seu repertório de ações é codificado a priori, e eles não planejam nem raciocinam sobre as consequências de longo prazo.
- **Agentes intencionais**: também chamados de **racionais, cognitivos ou deliberativos**, possuem representação explícita de crenças e intenções. Têm a **habilidade de raciocinar sobre suas próprias crenças e intenções**, criar e executar planos de ação e selecionar objetivos de acordo com suas motivações. Esses agentes detectam e resolvem conflitos entre metas, elaboram planos (agendamento de ações) e revisam-nos conforme necessário. Por esse motivo, são considerados agentes planejadores capazes de deliberar e, idealmente, agir racionalmente.
- **Agentes sociais**: são um **subconjunto dos agentes intencionais**. Além de possuírem crenças e intenções, eles **mantêm modelos de outros agentes** e

raciocinam sobre esses modelos para tomar decisões e criar planos. Esses agentes precisam lidar com problemas de consistência de crenças compartilhadas e cooperação em ambientes multiagente, exigindo mecanismos adicionais de comunicação, negociação e coordenação.

Os agentes reativos formam uma categoria própria; os agentes intencionais englobam agentes capazes de raciocinar, deliberar e planejar; e, dentro destes, existem os agentes sociais, que estendem as capacidades deliberativas para o contexto multiagente por meio de modelos dos outros agentes e processos de interação

Arquitetura de Damazeau para Agentes funcionais

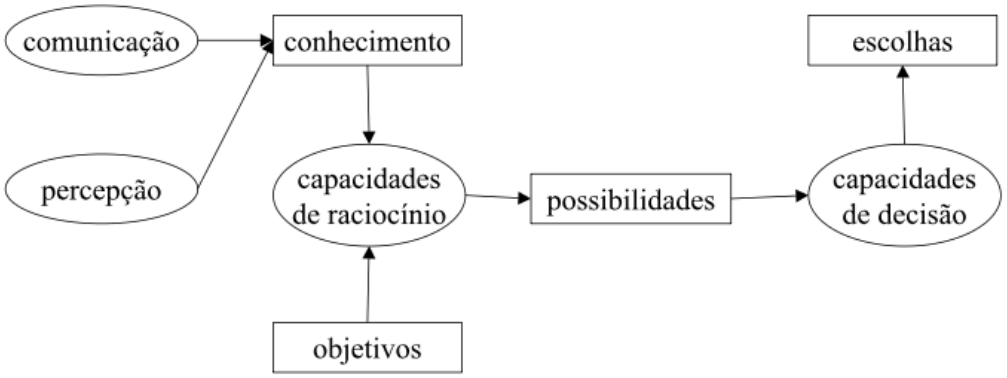


Imagem 4: *Arquitetura de Damazeau para Agentes Funcionais*

A “Arquitetura de Demazeau para Agentes Funcionais” propõe um modelo modular para agentes cognitivos, em que cada módulo cumpre um papel específico no ciclo de percepção-deliberação-ação.

- **Percepção e comunicação:** o agente obtém informação do ambiente de duas formas. A percepção capta estados do ambiente por meio de sensores; a comunicação recebe mensagens de outros agentes. Esses fluxos convergem para o **conhecimento**, que é a base interna de fatos e crenças sobre o mundo.
- **Conhecimento e objetivos:** o agente cognitivo, segundo Demazeau, deve possuir um conjunto de conhecimentos e objetivos que guiam seu comportamento.
- Os **objetivos** alimentam as **capacidades de raciocínio**, permitindo ao agente examinar o que quer alcançar.
- **Capacidades de raciocínio:** com base no conhecimento atualizado e nos objetivos, o agente raciocina para gerar um conjunto de **possibilidades** — planos ou alternativas de ação que podem conduzir ao alcance de suas metas.

- **Capacidades de decisão:** as possibilidades geradas são avaliadas por um módulo decisório. Essas capacidades de decisão escolhem entre as alternativas e produzem **escolhas** concretas de ação.

Esse arranjo mostra que um agente funcional necessita de mecanismos bem definidos de **percepção, comunicação, raciocínio e decisão**, além de **conhecimento e objetivos**.

Arquitetura baseada em estados mentais

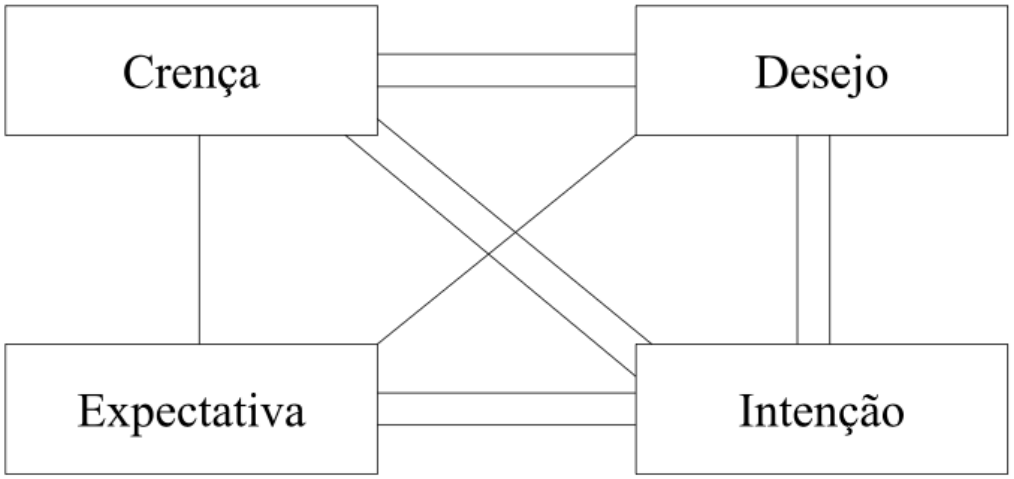


Imagem 5: *Arquitetura baseado em estados mentais*

Arquitetura baseada em estados mentais (segundo Corrêa 1994), em que o comportamento do agente é governado por quatro componentes mentais:

- **Crenças** – representam o conhecimento do agente sobre o ambiente; elas influenciam e são influenciadas pelas outras componentes.
- **Desejos** – expressam os objetivos que o agente gostaria de alcançar. Corrêa observa que um agente cognitivo é definido por crenças, objetivos e capacidades de raciocínio e decisão.
- **Intenções** – são desejos selecionados para execução; em arquiteturas BDI, as intenções resultam da escolha de quais objetivos o agente buscará efetivamente.
- **Expectativas** – representam previsões ou compromissos sobre o que deve acontecer ao executar uma intenção.

As linhas no diagrama indicam que esses estados se influenciam mutuamente: as crenças informam os desejos e permitem ajustar expectativas; desejos e crenças convergem para intenções; e as expectativas geradas pelas intenções podem atualizar crenças e desejos. Dessa forma, o ciclo deliberativo do agente integra conhecimento, metas, planos e previsões.

Arquitetura Completa do Agente Will

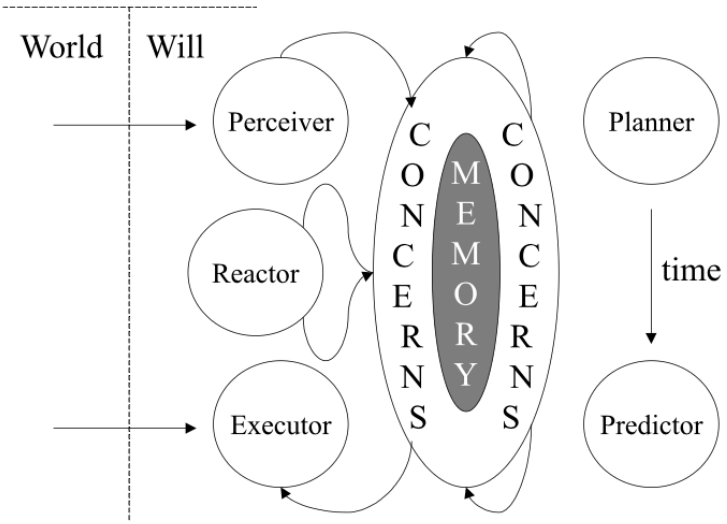


Imagem 6: *Arquitetura do Agente Will*

Propõe um agente híbrido combinando capacidades reativas e deliberativas. O esquema está dividido pela linha tracejada em dois domínios: o “World” (mundo) e o “Will” (agente).

- **Perceiver, Reactor e Executor** – localizados junto à fronteira com o mundo – formam a **parte reativa** do agente. O *Perceiver* recebe perceptos do ambiente, o *Reactor* determina reações imediatas com base nas “concerns” (preocupações e objetivos armazenados) e o *Executor* executa as ações no mundo. Moffat observa que um agente reativo responde a estímulos sem necessariamente planejar tudo.
- **A memória de concerns**, que armazena crenças, objetivos e estados internos que motivam o comportamento do agente. Todas as demais funções interagem com essa memória – percepções atualizam-na, reações consultam-na e execuções modificam-na.
- Os módulos **Planner** (planejador) e **Predictor** (preditor), que representam a **parte deliberativa**. O planeador usa a memória e uma previsão do futuro (fornecida pelo *Predictor*) para construir planos ao longo do tempo (seta vertical). Esses planos

alimentam as preocupações e, em consequência, influenciam a atuação do *Reactor* e do *Executor*.

Ao integrar módulos reativos e deliberativos em torno de uma memória de estados mentais, o Agente Will procura superar as limitações das arquiteturas puramente reativas ou puramente deliberativas. Segundo Bastos et al., arquiteturas híbridas combinam propriedades de ambas para lidar tanto com respostas rápidas quanto com planejamento e adaptação.

Arquitetura de Agentes Reflexivos Simples

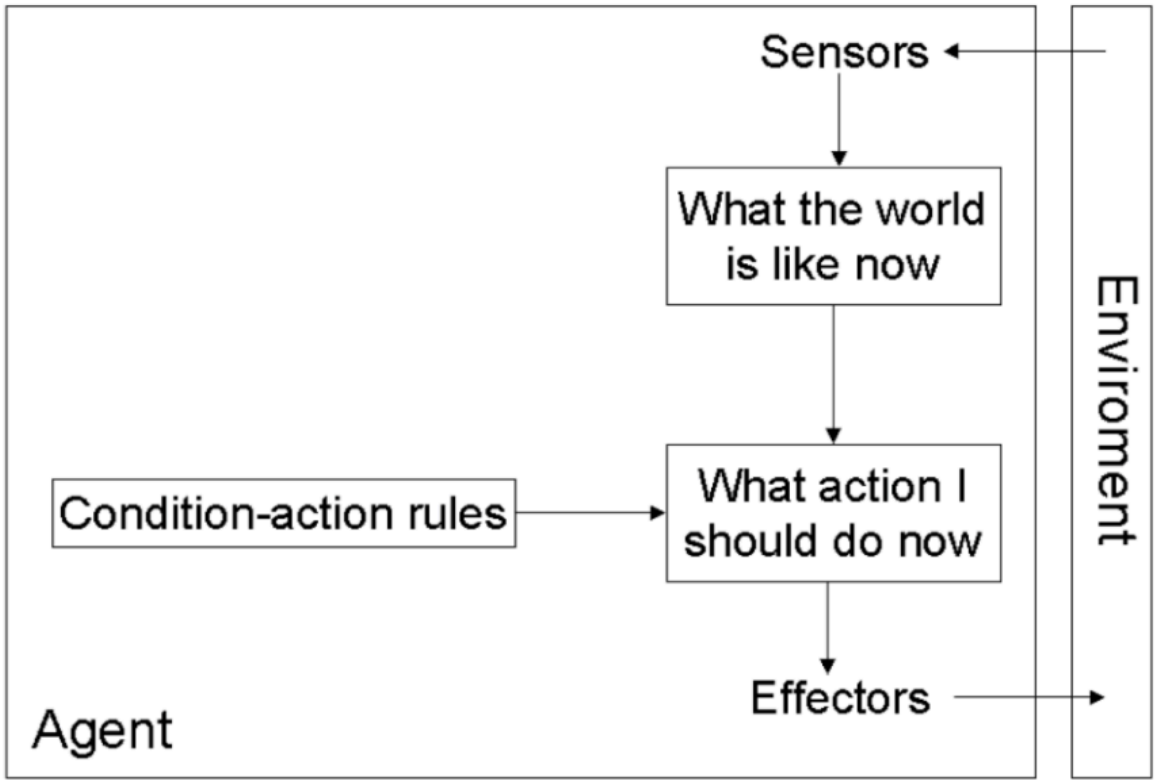


Imagem 7: *Arquitetura de Agentes Reflexivos Simples*

O agente reflexivo simples proposto por Russell e Norvig (1995). Trata-se do tipo mais básico de agente em que a ação depende somente do que é percebido no momento, sem estado interno.

- O agente mantém um conjunto de **regras de condição–ação** (RULES). Cada regra associa uma condição, expressa com base nos perceptos, a uma ação específica.

- Quando um percepto chega aos sensores, a função `INTERPRET-INPUT` o converte para uma descrição abstrata do estado atual. Em seguida, a função `RULE-MATCH` procura no conjunto de regras a primeira cujo antecedente corresponde à descrição gerada; a ação dessa regra é então retornada. O algoritmo pseudocódigo evidencia esse processo: dada a percepção, interpreta-se a entrada, busca-se a regra correspondente e executa-se a ação associada.
- Como não há memória ou modelo interno do mundo, o agente reflexivo simples "encontra uma regra cuja condição corresponde à situação atual … e executa a ação associada". Ele só é apropriado para situações em que a decisão correta pode ser tomada exclusivamente com base na percepção atual. Não lida bem com ambientes parcialmente observáveis nem pode planejar sequências de ações.

A arquitetura de Russell e Norvig mostra um agente composto por sensores que leem o estado atual do ambiente e effectors (atuadores) que executam ações. Entre eles, um módulo de regras condição–ação gera a ação imediata, sem considerar histórico ou metas de longo prazo.

Arquitetura de Agente Reflexivo com Estado

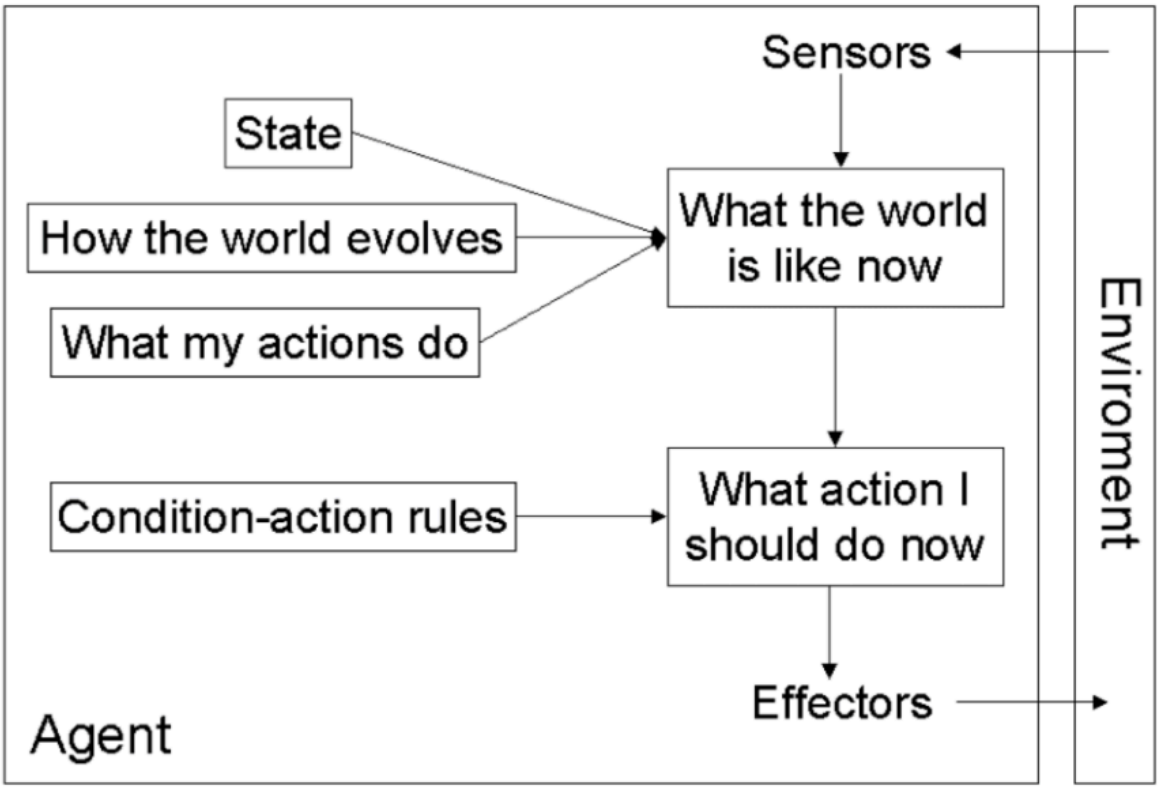


Imagem 8: *Arquitetura de Agente Reflexivo com Estado*

Agente Reflexivo com Estado, proposta por Russell e Norvig, a simplicidade do agente reflexivo é ampliada com a inclusão de uma **memória interna**.

- **Sensores → "What the world is like now"**: tal como no agente reflexivo simples, os sensores capturam um percepto que descreve o estado atual do ambiente.
- **Estado e modelos do mundo ("How the world evolves", "What my actions do")**: além do percepto atual, o agente mantém um **estado interno** que resume aspectos do mundo não diretamente observáveis. Esse estado é atualizado por uma função `UPDATE-STATE`, que utiliza tanto o percepto quanto conhecimento sobre **como o mundo evolui** e **como as ações do agente mudam o mundo**.
- **Regras condição-ação**: com base no estado atualizado, a função `RULE-MATCH` seleciona uma regra condição-ação apropriada, e a ação correspondente é executada por meio dos atuadores. O algoritmo resumido é: atualizar o estado interno a partir do percepto, encontrar a regra que casa com esse estado, executar a ação e, por fim, atualizar o estado em função da ação realizada.

Dessa forma, o **agente reflexivo com estado** continua a ser reativo (ele não planeja a longo prazo), mas agora consegue lidar com ambientes **parcialmente observáveis**. Ao modelar internamente "como o mundo evolui" e "o que minhas ações fazem", o agente corrige ou completa suas percepções atuais, permitindo decisões adequadas mesmo quando os sensores não fornecem uma visão completa do ambiente.

Arquitetura do Agente Baseado em Objetivos

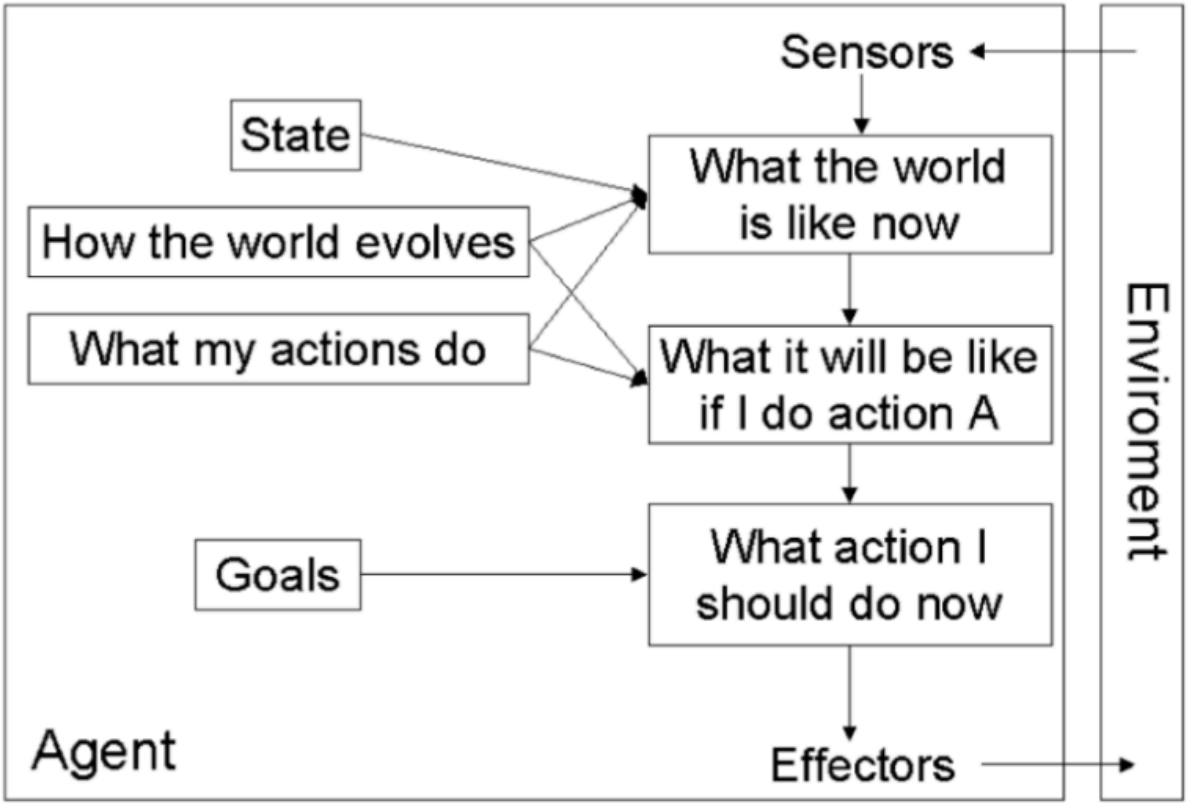


Imagem 9: *Arquitetura do Agente Baseado em Objetivos*

Esse tipo de agente estende o agente reflexivo com estado ao incluir uma representação explícita de **objetivos** e um mecanismo de **planejamento**. Em algumas aplicações, conhecer apenas o estado atual do ambiente não basta; é necessário **encontrar uma sequência de ações** que leve a um estado desejado. Nessas situações, técnicas de busca e planejamento são utilizadas para construir planos de ação que alcancem o objetivo.

Na arquitetura, os sensores atualizam uma representação interna do **estado** e utilizam modelos de **como o mundo evolui** e **o que minhas ações fazem**. A partir dessa modelagem, o agente calcula **"o que acontecerá se eu fizer a ação A"** e compara as possíveis trajetórias com seus **goals** (objetivos). Uma vez escolhido um plano, o módulo

"What action I should do now" seleciona a próxima ação a executar, enviada aos atuadores. Esse enfoque orientado a objetivos é mais flexível que um agente puramente reflexivo, porque o conhecimento que sustenta a decisão é modelado explicitamente e pode ser modificado conforme as necessidades.

O agente baseado em objetivos usa sua percepção e um modelo de transição do mundo para **buscar e planejar** sequências de ações que satisfazem critérios de "situações desejáveis". Ao fazer isso, ele pode lidar com tarefas que exigem **planejamento de passos múltiplos**, como navegação em um labirinto ou execução de rotinas complexas numa empresa.

Arquitetura de Agente Baseado em Utilidade

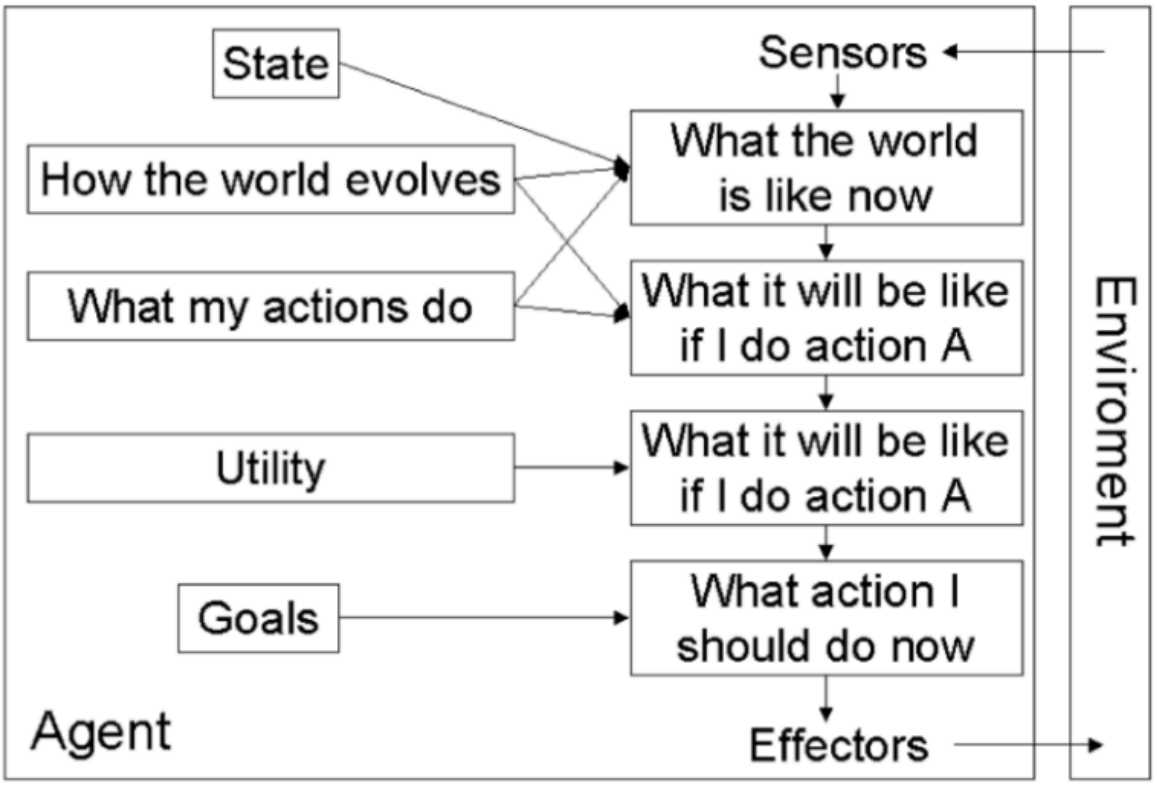


Imagem 10: *Arquitetura de Agente Baseado em Utilidade*

O **agente baseado em utilidade**, que é uma extensão do agente baseado em objetivos. O elemento central aqui é a introdução de uma **função de utilidade**.

- Tal como no agente com estado, o bloco **State** armazena uma representação interna do mundo e usa conhecimentos de **como o mundo evolui** e **o que minhas ações**

fazem para prever futuros estados resultantes de cada ação. Os sensores fornecem a percepção corrente (“What the world is like now”) e esses modelos permitem simular “What it will be like if I do action A”.

- O **bloco de utilidade (Utility)** atribui um valor numérico a cada estado previsto, representando quão “feliz” ou desejável esse estado é para o agente. Isso é útil porque dois planos podem atingir o mesmo objetivo, mas um pode ser mais rápido, seguro ou barato que outro. A função de utilidade permite comparar alternativas e tomar decisões racionais mesmo quando os objetivos entram em conflito ou não é certo que serão alcançados.
- O módulo de **goals** ainda descreve quais situações são desejáveis, mas, em vez de apenas verificar se um objetivo foi alcançado, o agente usa a utilidade para avaliar os diferentes caminhos. O bloco “What action I should do now” escolhe a ação que maximiza a utilidade esperada do estado futuro. Em outras palavras, a seleção de ações baseia-se em uma **preferência (utility) por cada estado**, e essa abordagem substitui ou complementa os objetivos quando existem conflitos entre metas ou incerteza sobre seu alcance.

O agente baseado em utilidade utiliza modelagem do mundo e previsão de ações, como o agente baseado em objetivos, mas acrescenta uma função de utilidade para tomar decisões mais refinadas, ponderando eficiência, segurança, custo e outras variáveis de interesse, e assim escolhe a ação que maximiza a “felicidade” ou valor esperado.

Interação Agente-Ambiente

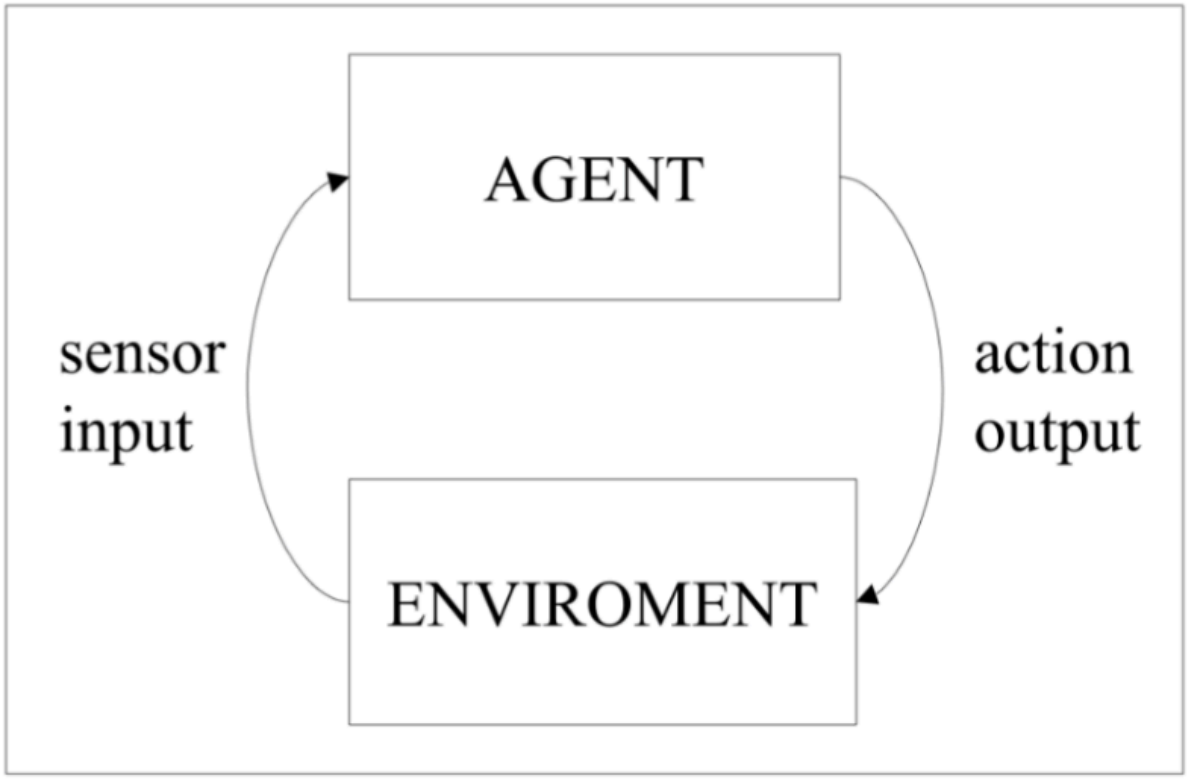


Imagem 11: *Interação Agente-Ambiente*

A seção “Ambiente” enfatiza que **todo agente está situado em um ambiente**, e que esse ambiente é definido como “tudo o que envolve um agente”. O ambiente serve de meio para a dispersão do controle, dos dados e do conhecimento entre agentes, e o termo **mundo** refere-se à descrição completa e instantânea desse ambiente.

O ciclo de interação agente-ambiente: o **agente** envia ações ao ambiente (via atuadores) e recebe **perceptos** (sensor input) em retorno, formando um laço de percepção e ação. Essa interação contínua determina como o agente deve representar o mundo e agir dentro dele.

Russell & Norvig destacam classificações como **acessível vs. inacessível** (se os sensores fornecem ou não um estado completo do ambiente), **determinístico vs. não-determinístico** (o ambiente evolui de forma previsível ou não), **episódico vs. não-episódico** (cada episódio depende apenas do percepto e ação correntes ou se há dependência histórica), **estático vs. dinâmico** (muda ou não durante a deliberação do agente) e **discreto vs. contínuo** (quantidade de perceptos e ações distintas). Esses fatores afetam como o agente

deve manter seu estado interno e qual arquitetura é mais adequada para operar no mundo real.

Arquitetura de Referência dos Agentes M-DRAP

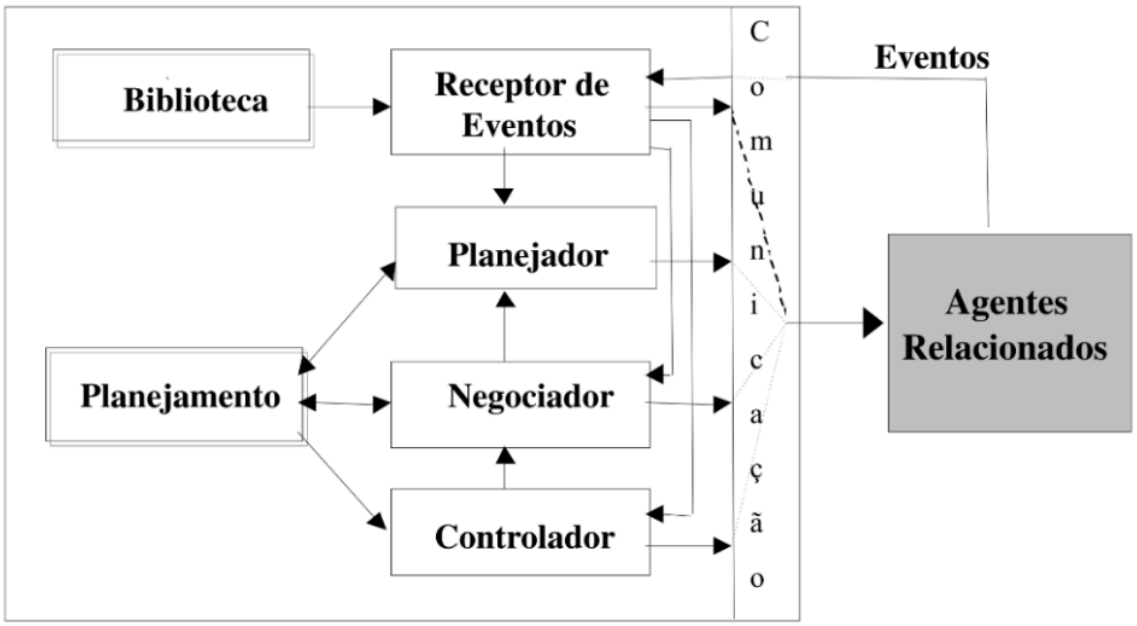


Imagem 12: *Arquitetura dos Agentes M-DRAP*

Na arquitetura de referência dos agentes M-DRAP, proposta por Ricardo Bastos, cada agente dispõe de módulos especializados que cooperam para resolver o problema da **alocação dinâmica de recursos** de forma autônoma e distribuída:

- **Biblioteca** – mantém a lista de agentes que participam de um plano e as suas **regras de procedimento**. Ela fornece informações de suporte a outros módulos (por exemplo, quem está envolvido numa tarefa e quais são as regras aplicáveis).
- **Receptor de eventos** – percebe eventos dirigidos ao agente, identifica a natureza de cada evento e o encaminha ao módulo apropriado para tratamento. Ele também consulta a biblioteca para **completar a mensagem original** com dados necessários.
- **Planejador/Planejamento** – duas componentes tratam do planeamento:
    - O **Planejador** realiza o planejamento das atividades que o agente executará; abre licitações (pedidos de proposta), solicita colaboração de outros agentes e encaminha propostas relativas às suas competências.
    - O **Planejamento** armazena todas as propostas e **compromissos assumidos** pelo agente, funcionando como um repositório de planos.

- **Negociador** – avalia as propostas recebidas, contrata serviços e estabelece compromissos quando aceita uma proposta. Ele também tenta contornar problemas causados por perturbações, buscando soluções alternativas para evitar prejuízos na execução dos compromissos.
- **Controlador** – monitora a execução de cada atividade, comunica o término aos agentes responsáveis pelas atividades seguintes e, em caso de atrasos, informa o negociador. Esse acompanhamento garante a coerência entre o plano e a realidade.

Externamente, o módulo de **Comunicação** (indicado na figura pelo canal com “Agentes Relacionados”) permite que cada agente M-DRAP envie e receba eventos, propostas e compromissos a partir de outros agentes. Essa troca de mensagens é essencial para viabilizar a cooperação e a negociação distribuídas que caracterizam o paradigma multiagente.

Arquitetura dos Agentes ADEPT

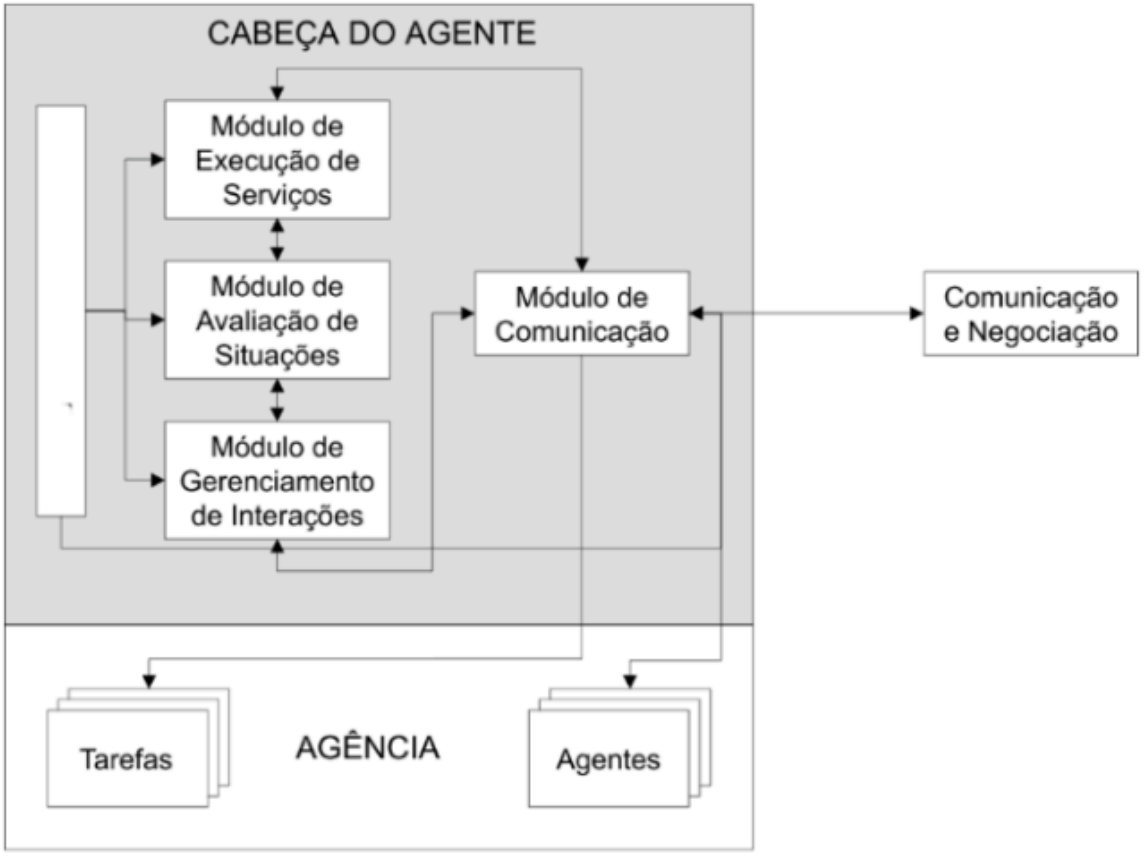


Imagem 13: *Arquitetura do agentes ADEPT*

A arquitetura **ADEPT** foi desenvolvida como parte de um projeto que visa criar uma infra-estrutura de gerenciamento de processos de negócio baseada em agentes. Cada agente ADEPT é construído sobre uma estrutura comum dividida em duas partes principais: a **cabeça do agente** e a **agência:**

Na **cabeça do agente**, situam-se os módulos funcionais responsáveis pela administração das interações e serviços:

- **Módulo de Comunicação** – roteia mensagens entre o agente e sua agência interna e também com outros agentes, servindo de interface para a rede.
- **Módulo de Gerenciamento de Interações** – fornece serviços por meio de negociação; ele administra o processo de negociação e cooperação com outros agentes quando há dependências ou conflitos em um processo de negócio.
- **Módulo de Avaliação de Situações** – monitora os contratos firmados e avalia as capacidades e compromissos do agente, verificando tanto os acordos vigentes quanto potenciais contratos futuros.
- **Módulo de Execução de Serviços** – gerencia a execução dos serviços, incluindo três papéis: controle da execução, gerenciamento da informação e tratamento de exceções que possam ocorrer durante a prestação do serviço.

No lado da **agência**, são mantidas as **tarefas** e os **agentes** que representam os recursos de resolução de problemas do domínio. A arquitetura ainda prevê componentes para armazenar e organizar o conhecimento do agente:

- **Modelos de Relacionamento** – guardam e provêm acesso aos contratos firmados com outros agentes e às listas de fornecedores de serviços relevantes.
- **Modelo Próprio** – guarda as descrições dos serviços que o agente pode oferecer, os contratos firmados, informações genéricas do domínio e dados de tempo de execução.

Com essa estrutura, os agentes ADEPT são capazes de negociar, coordenar serviços e adaptar-se dinamicamente a mudanças em processos empresariais.

Arquitetura de Agente de Aprendizagem

Ele aprende continuamente com experiências anteriores para melhorar seus resultados usando mecanismos de entrada e feedback.

Imagem 14: *Arquitetura de Aprendizagem*

Arquitetura de Agente Hierárquico

Imagem 15: *Arquitetura de Agente Hierárquico*

Eles são um grupo organizado de agentes inteligentes que são organizados em camadas. Os agentes de nível superior desconstroem tarefas complexas em tarefas menores e as atribuem aos agentes de nível inferior. Então, funciona mais ou menos como a departamentalização de empresas.

Então imagine uma empresa que tem departamento X, Y e Z em que os três são gerenciados pela gerência geral, pela pelo gestor geral e assim por diante. Basicamente, é dessa forma que você vai estruturar o teu agente em uma espécie de hierarquia.

APÊNDICE 3

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: **25 de set. de 2025**

Participantes da Entrega **[matriculados em Residência em IA]**:

Carlos Henrique Rodrigues de Jesus

Entrega: **[descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]**

Essa Semana me aprofundei mais no conteúdo sobre **Agentes Inteligentes.**

Na qual terminei de ler as referências bibliográficas sobre arquiteturas de agentes, produzi um estudo sobre do Agentic AI Survey of Technologies, Applications, and Impact, e comecei a estudar AgentOps. Completei meu estudo de tópico mapeando arquiteturas iniciais, para então me concentrar na comparação entre arquiteturas nas próximas etapas

- Estudo de tópico - Agentes de IA
- No próprio documento, mencionei algumas arquiteturas referentes, que me ajudaram a classificar posteriormente as arquiteturas de agentes de IA.
    - Arquitetura baseada em estados mentais (Corrêa 1994)
    - Arquitetura de Agentes Reflexivos Simples
    - Arquitetura de Agente Reflexivo com Estado
    - Arquitetura do Agente Baseado em Objetivos
    - Arquitetura de Agente Baseado em Utilidade
    - Arquitetura de Referência dos Agentes M-DRAP e Agentes ADEPT

Com o intuito de entender mais sobre arquiteturas e como agentes vêm sendo utilizados, produzi uma classificação de arquiteturas com base nos materiais já vistos:

- Estudo de tópico - Agentes de IA
- Classificação mais abrangente, onde são por camadas, desde aplicação até a nível de comunicação e programas:
    - Arquiteturas de controle
    - Arquiteturas por nível social
    - Arquiteturas por tipo de tarefa ou domínio
    - Arquiteturas por tipos de programas de agentes
- Classificações a nível hierárquico, onde definem:

- Arquiteturas reativas
- Arquiteturas deliberativas/cognitivas
- Arquiteturas híbridas
- Arquiteturas sociais (multiagentes)
- Arquiteturas baseadas em estados mentais

- Criei um diagrama para entender os níveis hierárquicos.

Leitura do artigo *Agentic AI*:

- Estudo de tópico - Agentes de IA

Comparação Arquitetural de agentes inteligentes:

- Estudo de tópico - Agentes de IA

Comecei a estudar um curso da Udemy sobre agentes inteligentes:

- AI Agents - Udemy

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Me aprofundar na área de AgentOPS;
Estudar frameworks de aplicação de Agentes;
Terminar curso na Udemy sobre Agentes de IA;

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Arquiteturas de Agentes e Classificações em Camadas

O termo **agente inteligente** aplica-se a qualquer sistema que perceba o ambiente por sensores e atue sobre ele por meio de efetores com o objetivo de maximizar alguma medida de desempenho. A literatura discute inúmeras maneiras de classificar agentes e suas arquiteturas, seja pelas propriedades que os agentes possuem, pelos mecanismos usados para selecionar ações, pelo tipo de tarefa que executam ou pelo nível de interação com outros agentes. Esta seção organiza essas classificações em **níveis hierárquicos**, mostrando quais conjuntos estão contidos em outros e como as diferentes arquiteturas se relacionam.

Classificações de agentes:

Várias taxonomias foram propostas para **agentes**. Algumas partem de propriedades fundamentais (autonomia, sociabilidade, reatividade e pró-atividade) e acrescentam outras propriedades como mobilidade ou adaptabilidade. Outras utilizam o tipo de tarefa desempenhada (informativo, cooperativo, transacional), o tipo de conhecimento que possuem (preferências do usuário, informações de negócios) ou o ambiente onde operam. A seguir, as classificações mais utilizadas são organizadas hierarquicamente.

Por arquitetura de controle:

Esta classificação considera os **mecanismos internos usados para selecionar a próxima ação**. Segundo Viccari e Giraffa (1996), as arquiteturas podem ser **deliberativas (cognitivas)**, **não-deliberativas (reativas)** ou **híbridas**. Alguns autores também distinguem as arquiteturas **baseadas em estados mentais** como um refinamento das deliberativas. Um agente pode pertencer a mais de uma subclasse (por exemplo, uma arquitetura híbrida combinada com mental-states). Os níveis são:

- **Não-deliberativos (reativos)** – Escolhem a ação com base em regras estímulo–resposta, associando percepções a ações sem raciocinar nem planejar. Não mantêm memória de ações passadas e não representam explicitamente o ambiente.
    - **Agente reflexivo simples** – Programa com regras condição-ação; reage ao percepto corrente. Russell & Norvig descrevem esse agente como uma conexão direta de perceptos a ações.
    - **Agente reflexivo com estado (model-based)** – Mantém um **estado interno** que armazena informações sobre como o mundo evolui e como as ações

mudam o ambiente. Essa variação atualiza o estado interno antes de escolher a ação, permitindo lidar com ambientes parcialmente observáveis.

- **Deliberativas (cognitivas ou intencionais)** – Utilizam um **modelo simbólico do ambiente** e **planejamento** para escolher ações. Mantêm uma representação explícita do conhecimento do ambiente, histórico de ações e objetivos. São capazes de raciocinar sobre crenças e intenções, selecionar objetivos e elaborar planos de ações.
    - **Agente baseado em objetivos** – Planeja uma sequência de ações para alcançar um estado desejado e decide a ação que melhor aproxima seus objetivos.
    - **Agente baseado em utilidade** – Extende o anterior adicionando uma função de utilidade que mede quão desejável é cada estado futuro; escolhe a ação que maximiza a utilidade esperada.
    - **Arquitetura baseada em estados mentais (BDI)** – Representa explicitamente **crenças**, **desejos** (metas possíveis) e **intenções** (metas comprometidas). Os agentes repetidamente atualizam crenças a partir de percepções, geram opções (desejos), filtram-nas para definir intenções e então agem. A estrutura BDI é considerada deliberativa porque mantém um modelo simbólico, mas alguns autores a tratam como uma categoria separada devido ao enfoque em estados mentais.
- **Híbridas (Arquiteturas Mistas)** – Combinam uma camada deliberativa (planejamento) com uma camada reativa responsável por respostas rápidas. Visam contornar as deficiências de cada abordagem isolada: os agentes cognitivos não reagem rapidamente a imprevistos e os agentes reativos não planejam quando a situação difere dos objetivos. São geralmente organizadas em camadas hierárquicas: níveis baixos reativos e níveis altos deliberativos. *Exemplos de arquiteturas híbridas são descritos na Secção 3.*

Por nível social (Corrêa, 1994):

Corrêa propõe uma classificação de **agentes quanto à sua capacidade de raciocinar sobre intenções e sobre outros agentes**:

- **Reativos** – Respondem a estímulos sem planejar ou manter modelos do ambiente.
- **Intencionais (ou racionais/cognitivos)** – Possuem habilidades de raciocinar sobre crenças e intenções e são capazes de selecionar objetivos, detectar conflitos entre objetivos e elaborar planos.
- **Sociais** – Subconjunto dos intencionais. Além de planejar, **mantêm modelos de outros agentes** e raciocinam sobre esses modelos para decidir. Necessitam de

mecanismos de comunicação, negociação e coordenação para manter consistência de crenças compartilhadas e cooperar em um ambiente multiagente.

Por tipo de tarefa ou domínio:

Diversos autores classificam agentes pelo tipo de tarefa que realizam:

- **Informativos**: Ajudam o usuário a buscar e filtrar informações em fontes distribuídas.
- **Cooperativos**: Resolvem problemas complexos cooperando com outros agentes ou fontes externas.
- **Transacionais**: Processam e monitoram processos (por exemplo, fluxos de trabalho empresariais).
- **Interface** e **de Tarefa**: Interagem diretamente com o usuário ou auxiliam na execução de tarefas, formulando planos de resolução.
- **De Usuário** e **de Negócio**: Detêm informações sobre o usuário (preferências) ou sobre o negócio (produtos, serviços).
- **Emocionais**: Incorporam modelos de emoções e personalidade.
- **Pedagógicos**: Atuam em ambientes educacionais.

Nwana (1996) ainda distingue agentes **colaborativos** (focados na cooperação), **móveis**, **reativos** e **híbridos**, enquanto Franklin e Graesser (1996) separam agentes **biológicos**, **robôs** e **computacionais**, e este último grupo em agentes de **vida artificial** e **de software**.

Por autonomia, mobilidade e adaptabilidade:

Além das classificações acima, é comum descrever agentes pelo grau de **autonomia** (nível de independência nas decisões), **mobilidade** (capacidade de deslocamento entre plataformas ou ambientes) e **adaptabilidade** (capacidade de aprender com experiências). Essas propriedades podem ser combinadas com as classificações anteriores; por exemplo, agentes móveis podem ser reativos ou deliberativos.

Por tipos de programas de agentes (Russell & Norvig)

No campo da inteligência artificial, Russell e Norvig identificam quatro (e às vezes cinco) **tipos de programas de agentes**:

- **Agentes reflexivos simples** – Implementam regras condição-ação diretamente mapeadas da percepção para a ação.
- **Agentes reflexivos baseados em modelo (model-based)** – Mantêm um estado interno que registra como o mundo evolui e como as ações o afetam.

- **Agentes baseados em objetivos** – Avaliam ações com base em metas explicitamente representadas e planejam sequências de ações para atingi-las.
- **Agentes baseados em utilidade** – Usam uma função de utilidade para escolher entre estados desejáveis, buscando maximizar a utilidade esperada.
- **Agentes de aprendizagem** (quando incluídos) – Modificam ou produzem sua própria função de desempenho a partir de experiências; podem adaptar-se a ambientes desconhecidos.

Arquiteturas de agentes e suas relações

A seguir são descritas algumas **arquiteturas específicas** discutidas. Elas são organizadas conforme a classificação de arquitetura de controle (reativa, deliberativa, híbrida ou mental-state) e exibidas de forma hierárquica. Muitas arquiteturas também fazem parte de sistemas multiagentes, sendo simultaneamente **sociais** quando incluem negociação ou coordenação com outros agentes.

Arquiteturas reativas:

Arquitetura	Nível	Características	Fonte
Agente Reflexivo Simples	Reativa	Programa definido por regras condição–ação; reage diretamente aos perceptos. Não possui estado interno.	Russell & Norvig
Agente Reflexivo com Estado	Reativa	Mantém um estado interno que armazena como o mundo evolui e como as ações do agente modificam o ambiente. Usa o estado para interpretar perceptos e escolher ações.	Russell & Norvig
Arquitetura de Subsunção (Brooks)	Reativa	Divisão de controle em camadas de comportamentos simples; níveis superiores subsumem níveis inferiores. Embora não apareça no relatório, é considerada reativa pura.	N/A
Demazeau – Módulos reativos	Parte de híbrida	Na arquitetura funcional de Demazeau, os módulos de percepção e comunicação fornecem informações ao conhecimento; o raciocínio e a decisão ficam em módulos separados. Os módulos de percepção e comunicação desempenham papel reativo.	Demazeau

Tabela 1: *Arquiteturas Reativas Comparação*

Arquiteturas deliberativas/cognitivas:

Arquitetura	Nível	Características	Fonte
Agente Baseado em Objetivos	Deliberativa	Representa o mundo e os efeitos de ações; planeja sequências de ações para alcançar metas especificadas.	Russell & Norvig
Agente Baseado em Utilidade	Deliberativa	Semelhante ao anterior, mas usa uma função de utilidade para comparar estados futuros e escolher a ação que maximiza a utilidade.	Russell & Norvig
Arquitetura BDI (Belief-Desire-Intention)	Deliberativa/Mental-State	Mantém crenças, desejos e intenções como estados mentais. Ao perceber o ambiente, atualiza as crenças, gera desejos, filtra-os para definir intenções e age de acordo. É um caso particular de arquitetura baseada em estados mentais.	Corrêa
Arquitetura baseada em estados mentais (Corrêa, 1994)	Deliberativa/Mental-State	Modelo que relaciona crenças, desejos, intenções e expectativas, enfatizando a influência mútua entre esses estados. Usada como base conceitual para arquiteturas BDI e outras.	Corrêa

Tabela 2: *Arquiteturas Deliberativas/Cognitivas*

Arquiteturas Híbridas:

Arquitetura	Nível	Características	Fonte
Arquitetura Funcional de Demazeau	Híbrida	Modela agentes como módulos funcionais: percepção/ comunicação alimentam o conhecimento; raciocínio produz possibilidades; decisão escolhe alternativas. O conhecimento mantém objetivos e é construído a partir de percepções e comunicações. Combina módulos reativos e deliberativos.	Demazeau
Agente Will	Híbrida	Combina módulos reativos (Perceiver, Reactor, Executor) com módulos deliberativos (Planner, Predictor), todos conectados a uma memória de **concerns** (crenças, objetivos). Permite reação rápida e planejamento a longo prazo.	Moffat
M-DRAP (Multi-agent Dynamic Resource Allocation Planning)	Híbrida e Social	Modelo para alocação dinâmica de recursos em manufatura, baseado em CIMOSA. Cada agente possui componentes: **Receptor de Eventos** (captura eventos e completa a mensagem com dados da biblioteca), **Biblioteca**(armazena agentes e regras), **Planejador/Planejamento** (planeja atividades, abre licitações e armazena propostas), **Negociador** (avalia propostas, contrata serviços, resolve perturbações) e **Controlador** (acompanha execuções e comunica atrasos). Os agentes negociam entre si para manter planos consistentes, sendo portanto sociais.	Bastos
ADEPT	Híbrida e Social	Projeto para gestão de processos de negócio. Cada agente tem uma **cabeça**, contendo módulos de comunicação, gerenciamento de interações (negociação), avaliação de situações (monitoramento de contratos) e execução de serviços, e uma **agência**, que reúne recursos de resolução (tarefas e subagentes). Os agentes ADEPT negociam serviços e contratos e coordenam execuções.	Jennings

Tabela 3: *Arquiteturas Híbridas*

Arquiteturas sociais (multiagentes):

Arquiteturas sociais são estruturas que partem de agentes deliberativos ou híbridos e adicionam mecanismos de **comunicação**, **negociação** e **modelos de outros agentes**. Elas se enquadram no subconjunto de **agentes sociais** de Corrêa. Exemplos incluem:

- **M-DRAP** – Agentes negociam ofertas de recursos e replanejam diante de perturbações.
- **ADEPT** – Foca em negociações de serviços em processos de negócio.
- **Demazeau (com cooperação)** – Embora cada agente seja funcionalmente modular, a arquitetura pressupõe coordenação entre agentes via comunicação.

Relações hierárquicas e inclusões

Os **níveis de inclusão** entre as principais classes e arquiteturas:

1. **Agentes inteligentes**
   - Possuem sub-conjuntos definidos por **autonomia, mobilidade, adaptabilidade** e **tipo de tarefa**(informativos, cooperativos, etc.).
   - Dividem-se em **arquiteturas de controle**: reativas, deliberativas, híbridas e mental-state.
2. **Arquiteturas reativas**
   - São subconjuntos de agentes **não-deliberativos**. Incluem o **agente reflexivo simples** e o **agente reflexivo com estado**.
   - Estas arquiteturas podem ser combinadas com propriedades de mobilidade e autonomia.
3. **Arquiteturas deliberativas (cognitivas/intencionais)**
   - Formam um subconjunto dos **agentes intencionais** de Corrêa.
   - Contêm arquiteturas baseadas em **objetivos** e **utilidade** (Russell & Norvig) e arquiteturas **baseadas em estados mentais** como a BDI.
   - Dentro de BDI, as representações de crença, desejo e intenção são mantidas de forma explícita.
4. **Arquiteturas híbridas**
   - **Contêm** componentes reativos (para respostas rápidas) e deliberativos (para planejamento).
   - Exemplos incluem **Demazeau**, **Will**, **M-DRAP** e **ADEPT**.
   - Muitas arquiteturas híbridas são **multiagentes** e, portanto, fazem parte dos **agentes sociais**.

5. **Arquiteturas sociais (multiagentes)**
    - Subconjunto dos agentes **intencionais**; exigem modelos de outros agentes.
    - Os agentes **M-DRAP** e **ADEPT** usam comunicação e negociação para coordenar tarefas.
    - Arquiteturas de base como **BDI** podem ser estendidas para ambientes multiagentes (por exemplo, BDI com comunicação e planos de grupo).
6. **Arquiteturas baseadas em estados mentais**
    - São **subconjuntos** das deliberativas, pois mantêm um modelo simbólico. A BDI é o exemplo mais conhecido.
    - Podem ser aplicadas em contextos multiagentes quando agentes compartilham ou negociam crenças e intenções.

Quadro Comparativo entre os Tipos de Agentes

Tipo de Agente	Características Principais	Exemplo Típico
Reflexivo Simples (Reativo)	Decide ações pelo estímulo atual imediato, sem memória; regras fixas condição→ação; adequado a ambientes simples totalmente observáveis.	Termostato ligando aquecimento ao detectar frio.
Reflexivo com Modelo (Reativo)	Mantém estado interno com modelo do mundo para lidar com percepção incompleta; ainda reage por regras reflexas, mas considerando o estado estimado.	Robô simples que atualiza um mapa parcial do ambiente interno.
Orientado a Objetivos (Deliberativo)	Possui metas definidas e escolhe ações planejando alcançar estados objetivo; requer busca/planejamento; permite comportamento proativo visando fins.	Aspirador robô (**Roomba**) que planeja limpar todos os cômodos.
Baseado em Utilidade (Deliberativo)	Avalia múltiplas opções via uma função de utilidade/valor; seleciona ação que maximiza utilidade esperada (considera preferências e probabilidades).	Carro autônomo ajustando rota ótima que equilibra segurança e tempo de viagem.
Agente BDI (Deliberativo Intencional	Utiliza modelo mental de Crenças, Desejos (objetivos) e Intenções; decide via raciocínio simbólico sobre estados mentais; equilibra reconsiderar planos	Sistema PRS/dMARS em controle industrial, onde cada agente BDI busca objetivos da planta de produção

)	vs. manter compromissos.	autonomamente.
Arquitetura Reativa Subsunção	Controlador em camadas de comportamentos reativos; camadas superiores inibem ou subsomem inferiores para compor ações complexas; sem representação simbólica central.	Robô móvel de Brooks (anos 1980) com camadas: evitar obstáculo (nível 1), vagar (nível 2), explorar (nível 3) – cada camada construída sobre as anteriores.
Arquitetura Híbrida (Reativo+Delib.)	Combina reação rápida e planejamento deliberativo em camadas ou módulos; camada reativa responde a eventos urgentes, enquanto camada deliberativa traça metas e planos de alto nível.	Robô de armazém: reage imediatamente a obstáculos (camada reativa) mas otimiza rotas de coleta de itens (camada deliberativa).
Sistema Multiagente (SMA)	Múltiplos agentes autônomos interagem; podem cooperar ou competir; comunicação via mensagens; solução distribuída de problemas; ênfase em organização (sociedades de agentes).	Projeto **ADEPT**: agentes negociadores gerindo diferentes partes de um processo de negócio e comunicando-se para alcançar um acordo global.
Agente Social	Focado em interação com outros agentes/humanos; comunica-se em linguagem de agentes; capaz de cooperação e negociação; segue normas sociais ou protocolos para atingir objetivos coletivos.	Agente de negociação em e-commerce trocando propostas e contra-propostas com outros agentes vendedores/compradores para fechar um contrato.
Agente Emocional	Integra modelagem de emoções (ex.: alegria, medo) que afetam suas decisões; calcula estados emocionais a partir de eventos relativos a seus objetivos (modelo OCC etc.); aumenta empatia e realismo nas interações.	Agente tutor virtual que “fica satisfeito” quando o aluno acerta (encorajando-o) ou “frustrado” após repetidos erros, ajustando sua estratégia de ensino de acordo.
Agente Móvel	Pode migrar autonomamente entre diferentes sistemas ou máquinas na rede; carrega consigo seu código e estado; executa tarefas próximo à fonte de dados; requer ambiente de hospedagem seguro.	**Agente Telescript** viajando por servidores na internet para coletar informações: ele sai do dispositivo do usuário, executa nos servidores remotos as consultas e retorna com os resultados.
Agente com Aprendizad	Incorpora algoritmos de ML para adaptar comportamento; aprende com dados ou feedback (reforço, supervisionado etc.);	**AlphaGo** (DeepMind): agente de jogo de Go que aprendeu políticas de jogada via *deep*

o de Máquina	melhora desempenho ao longo do tempo além da programação original.	*learning* e autojogo por reforço, atingindo nível super-humano ao otimizar sua função de recompensa vitória/derrota.

Tabela 4: *Quadro Comparativo entre os Tipos de Agentes*

Diagrama Hierárquico:

Imagem 16: *Diagrama Hierárquico*

O diagrama ilustra as categorias descritas, evidenciando a progressão de complexidade. No topo tem-se a classe geral **"Intelligent Agents"**, da qual partem ramificações para agentes

Reativos, **Deliberativos**, **Híbridos**, **Multiagentes** e agentes **Especializados** (sociais, emocionais, móveis, de aprendizado).

Conforme descemos nos níveis, vemos subcategorias: por exemplo, agentes reflexivos simples e baseados em modelo (no ramo reativo), e agentes baseados em objetivos, em utilidade e BDI (no ramo deliberativo). Arquiteturas híbridas aparecem combinando elementos deliberativos e reativos, enquanto no ramo multiagente destacam-se exemplos de frameworks (M-DRAP, ADEPT). Finalmente, o ramo de agentes especializados engloba características avançadas que complementam as dimensões anteriores (interação social, simulação emocional, mobilidade e aprendizagem).

Este **mapa conceitual** evidencia as relações hierárquicas e a evolução dos paradigmas de agentes inteligentes, integrando desde os modelos canônicos de Russell & Norvig até as tendências contemporâneas de agentes autônomos adaptativos.

Survey: Angetic AI

Se tivéssemos que definir a IA agêntica em uma palavra, seria 'proatividade': a capacidade de antecipar, iniciar e agir autonomamente em direção a objetivos. Mais formalmente, é um sistema de IA ou software que pode entender problemas humanos, coletar dados relacionados, usar os dados e realizar tarefas autodeterminadas para resolver o problema com intervenção humana nula ou mínima, interagindo com seu ambiente.

Agentes de IA são componentes especializados projetados para realizar tarefas únicas. A IA agêntica é um sistema autónomo que orquestra dinamicamente múltiplos agentes para alcançar objetivos mais amplos. Por exemplo, um agente de IA em imagem médica pode ser uma CNN que classifica tumores em exames, nada mais. Em contraste, um sistema de IA agêntica integraria múltiplos desses agentes (por exemplo, um buscando o histórico do paciente, outro validando cruzadamente os resultados de ressonância magnética e raios-X, e um terceiro alertando os médicos) em um pipeline coeso e orientado a objetivos. Pense nos agentes de IA como tijolos e na IA agêntica como a casa inteira: os primeiros realizam tarefas fixas, enquanto a última coordena e adapta dinamicamente essas tarefas para alcançar objetivos complexos, como um diagnóstico de ponta a ponta. A IA agêntica é uma plataforma unificada que conecta perfeitamente agentes de IA com usuários humanos para facilitar uma colaboração tranquila.

Como funcionam os agentes de IA?

Modelos baseados em fases, alguns descritos com quatros fases (coleta de dados → decisão → aprendizagem → colaboração), outros dividem em cinco (perceber → raciocinar → agir → aprender → colaborar), sendo equivalentes entre si.

- Percepção abrange a coleta e interpretação de dados, enquanto 'tomada de decisão' combina raciocínio e ação.

A IA Agêntica coleta dados, toma decisões autonomamente e se adapta a novas entradas.

Passo 1: Coleta de dados → A IA Agêntica monitora seu ambiente coletando dados - seja texto, imagens ou sinais do mundo real - que por fim se alinham com seus objetivos.

Passo 2: Tomada de decisão → Com as informações coletadas, a IA agêntica avalia o contexto e determina a ação mais apropriada para conclusão do seu objetivo.

Passo 3: Aprendizagem → A IA agêntica melhora continuamente através das experiências coletadas. Refinando seus comportamento analisando eventos.

Passo 4: Colaboração → A IA agêntica facilita a colaboração entre múltiplos agentes de IA e promove a interação significativa com usuários humanos. Enfatizando o comportamento coordenado e a sinergia humano-IA.

A IA Agêntica evolui através da interação contínua com seu ambiente e da troca de insights entre sistemas.

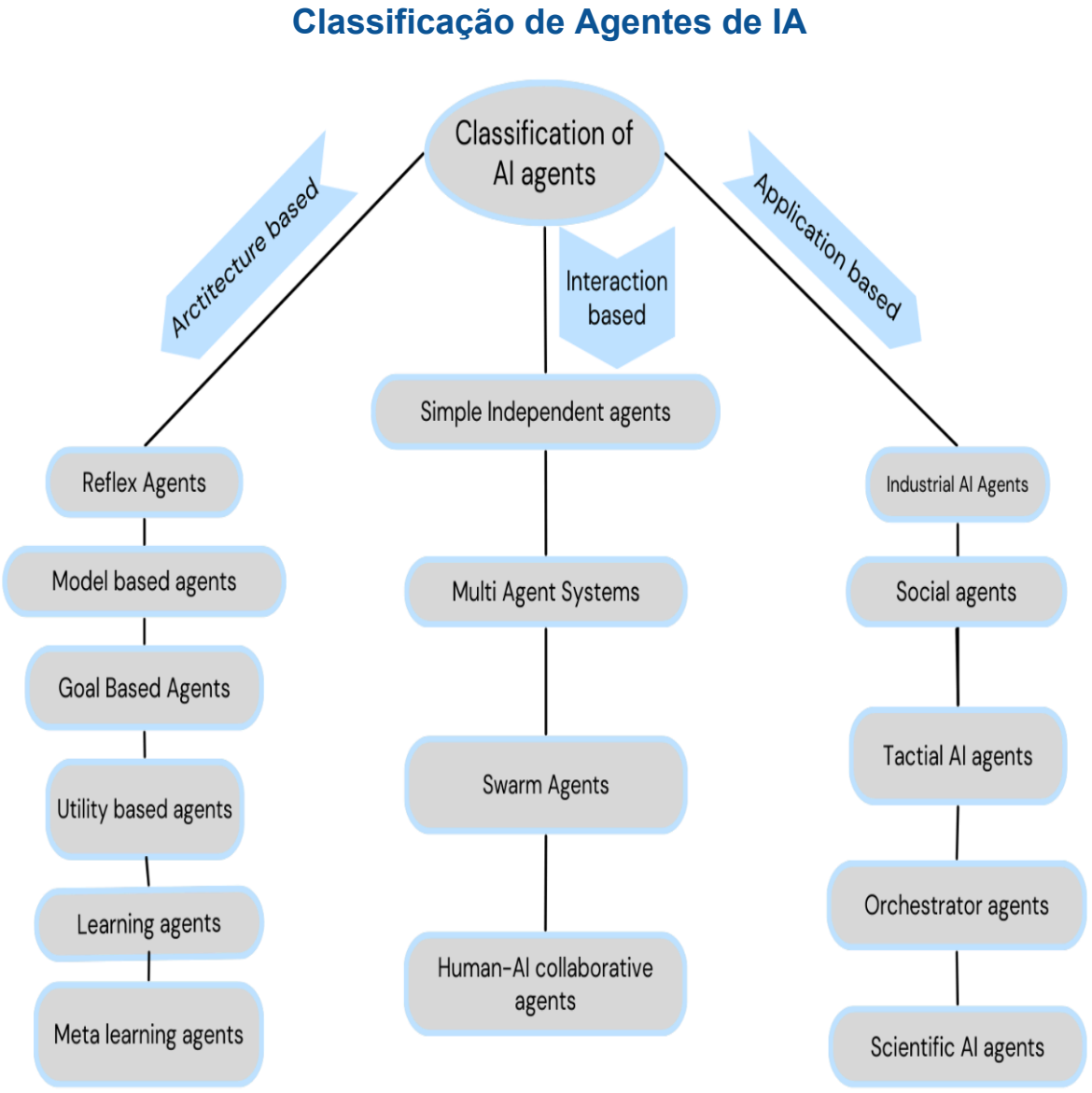


Imagem 17: *Classificação de Agentes de IA*

A classificação baseada na arquitetura categoriza os sistemas de IA agêntica com base em suas estruturas de tomada de decisão subjacentes:

- **Agentes de reflexo** funcionam com regras predefinidas e não se lembram de ações passadas. Eles apenas reagem às entradas atuais.

- A**gentes baseados em modelo (estado)**, que mantêm um modelo interno para lidar com as observações.

- **Agentes baseados em objetivos**, que planejam ações para alcançar objetivos. Eles podem lidar com problemas complexos e de múltiplos passos.

- **Agentes baseados em utilidade**, que estendem os agentes baseados em objetivos, mas escolhem ações que maximizam a utilidade esperada, por exemplo, lucro, tempo, custo de combustível, etc. Aqui, não apenas é considerado o caminho mais curto, mas também leva em conta tempo, conforto, tráfego e tudo mais ao dirigir.
- **Agentes de aprendizagem** melhoram o desempenho autonomamente através da aprendizagem por reforço.

- O último são os **agentes de meta-raciocínio**, que podem modificar seu processo de aprendizagem e escolher diferentes algoritmos e os melhores hiperparâmetros. Podem lidar com cenários nunca vistos, como dirigir fora de estrada para evitar o tráfego em tempo real.

Os agentes de IA também são classificados de acordo com o **paradigma de interação**: como eles se envolvem com outros agentes e humanos.

- **Sistemas de agente único**: No nível mais simples, os agentes de IA trabalham de forma independente, realizando tarefas autocontidas. Por exemplo, jogar xadrez ou filtrar spam são tarefas únicas.
- **Sistemas multiagentes**: Sistemas de IA agêntica mais complexos, onde múltiplos agentes interagem de forma hierárquica (de cima para baixo) ou plana (ponto a ponto). Esses sistemas são adequados para desafios mais complicados, como manter a gestão da cadeia de suprimentos e coordenar centralmente com agentes de IA.
- **Sistemas de enxame (swarm systems)**: Dependem da coordenação descentralizada. Esses sistemas são particularmente adequados para ambientes dinâmicos, onde regras simples levam a um comportamento emergente complexo, como visto em enxames de sensores ou robôs de resposta a desastres.
- **Sistemas colaborativos humano-IA** envolvem a interação humano-IA. Variando de papéis assistenciais em diagnósticos médicos à gestão de tráfego. Esses paradigmas apoiam a tomada de decisão compartilhada e são críticos em domínios de alto risco, onde a supervisão ética e o julgamento humano são essenciais.

Os agentes de IA podem ser classificados com base em suas **aplicações em diferentes domínios**, pois domínios distintos exigirão capacidades e formas de implementação distintas.

- **Agentes de IA industriais** operam nos setores de manufatura. Eles se destacam na inspeção de qualidade e na montagem robótica, e sua arquitetura combina ações de reflexo com previsões baseadas em modelo.
- **Agentes sociais**, como os de atendimento ao cliente, são usados para interação humana. Eles enfatizam principalmente o PLN e a inteligência emocional. Eles usam algoritmos de aprendizagem avançados para se adaptar ao comportamento do usuário.
- **Agentes de IA táticos** são empregados na resposta a emergências, por exemplo, nos setores de defesa e segurança, priorizando a tomada de decisão rápida, a coordenação com operadores humanos e tendo arquiteturas orientadas a objetivos.
- **Agentes de IA orquestradores** são usados em sistemas complexos como a gestão de tráfego em tempo real. Sua força reside na aprendizagem baseada em utilidade.
- **Agentes de IA científicos** são usados em pesquisa, como a descoberta de medicamentos. Eles têm fortes habilidades analíticas e usam meta-raciocínio.

Tecnologias em Sistemas de IA Agêntica

a IA Agêntica depende de tecnologias de ponta, que lhe conferem funcionalidades centrais como autonomia, adaptabilidade e colaboração.

- **Processamento de Linguagem Natural (NLP) e reconhecimento de fala:** Usadas para entender a linguagem natural, possibilitando a comunicação e ajudando no raciocínio e na sumarização de textos.
    - Tecnologias como: Grandes Modelos de Linguagem (LLMs), Claude, GPT e Whisper.
- **Aprendizado por reforço (RL) e Aprendizado por Reforço Inverso (IRL):** Permitem que os agentes de IA aprendam interagindo com o ambiente. Cada ação recebe um feedback positivo ou negativo e ajuda a refinar suas políticas.
    - RL foca em aprender políticas ótimas através da interação por tentativa e erro com o ambiente. *Aprendizado por reforço profundo combina a ideia do RL e redes neurais, para os agentes lidarem com tarefas mais complexas e criticas.*
    - IRL em vez de aprender com recompensas, o agente infere a função de recompensa subjacente observando demonstrações de especialistas (é como aprender hábitos de condução observando motoristas humanos). *Útil em definir recompensas explícitas e desafiadoras.*
- Aprendizado por Imitação (IL): Os agentes aprendem comportamentos diretamente de demonstrações de especialistas sem exigir uma função de recompensa explícita. Existem duas abordagens principais sendo elas:
    - Clonagem comportamental (mapeamento de estados para ações);

    - Agregação de conjunto de dados (correção iterativa de erros por um especialista).
- Sistemas Multiagentes (MAS) e frameworks de inteligência de enxame: São os princípios centrais que suportam a coordenação e comunicação entre múltiplos agentes. Às vezes, os agentes competem entre si para melhorar o desempenho.
    - JADF (Java Agent Development Framework) são usados para realizar MAS.
    - Algoritmos de exame como Otimização por Colônia de Formigas (ACO) e Otimização por Enxame de Partículas (PSO), usados para redirecionamento, balanceamento de carga e operações de busca e resgate.
- Integração Neurosimbólica para raciocínio: Combina IA simbólica com Redes Neurais. A rede aprende o padrão a partir dos dados, enquanto a IA simbólica contribui para raciocínio lógico.
    - Lógica Diferencial;
    - Aprendiz de conceito neurosimbólico da IBM;
    - Teoremas neurais para atingir objetivos.
- Outras tecnologias:
    - **Para Representação de Conhecimento**: **Grafos de conhecimento e ontologias** são usados para criar uma representação estruturada do conhecimento, o que ajuda no raciocínio, no planejamento de tarefas e na busca semântica.
    - **Para Percepção do Mundo Físico**: **Visão computacional e fusão de sensores**, utilizando ferramentas como CNNs, LiDAR e SLAM (localização e mapeamento simultâneos), permitem que os sistemas visualizem o ambiente e coletem informações.
    - **Para Planejamento de Longo Prazo**: **Algoritmos de planejamento e agendamento autónomos**, como STRIPS e Redes Hierárquicas de Tarefas (HTN), são empregados para o sequenciamento de tarefas e o raciocínio sobre objetivos complexos.
    - **Para Interação Humano-IA**: Tecnologias como a **IA explicável (XAI)** e a **realidade aumentada (AR)** são cruciais para garantir a transparência do sistema e permitir a tomada de decisões em colaboração com os humanos.

Evolução Arquitetural de Agentes Inteligentes

Agentes inteligentes são sistemas de software (ou robóticos) capazes de **perceber o ambiente, tomar decisões e agir de forma autônoma** para alcançar objetivos definidos. Diferentemente de programas tradicionais, agentes exibem propriedades como **autonomia, reatividade, proatividade e capacidade social**. Isso significa que um agente pode **operar sem intervenção humana constante**, **responder a mudanças no ambiente em tempo real**, **deliberadamente perseguir metas** e, quando necessário, **interagir ou cooperar com**

outros agentes ou pessoas. Esses princípios fundamentais foram estabelecidos na literatura de IA distribuída e de agentes desde as décadas finais do século XX.

Ao longo do tempo, a **arquitetura interna e as tecnologias habilitadoras de agentes** evoluíram significativamente. Nos primórdios, os agentes eram concebidos de acordo com paradigmas clássicos – desde **agentes puramente reativos**, passando por agentes **deliberativos simbólicos**, modelos **BDI (Belief-Desire-Intention)** inspirados em estados mentais, até arquiteturas **híbridas** que combinavam camadas reativas e deliberativas. Também emergiu o campo de **sistemas multiagentes**, explorando como múltiplos agentes autônomos podem coordenar-se ou competir dentro de um ambiente compartilhado.

Mais recentemente, com os avanços em **IA generativa e grandes modelos de linguagem (LLMs)**, surgiram os chamados **agentes generativos**, nos quais um LLM atua como núcleo do raciocínio do agente. Exemplos populares – como *AutoGPT*, *BabyAGI*, *Voyager*, *CAMEL*, *ReAct*, entre outros – demonstram agentes que **planejam, aprendem e interagem usando modelos de linguagem treinados em larga escala**. Esses agentes modernos incorporam novas ferramentas conceituais e técnicas, incluindo **raciocínio em linguagem natural (Chain-of-Thought)**, **memória de longo prazo via bases de vetor**, **planners simbólicos integrados** e **orquestração de ferramentas externas (APIs, código)**, além de modos inovadores de **interação autônoma** (por exemplo, múltiplos agentes LLM colaborando via diálogo).

Neste estudo comparativo, examinamos a **progressão histórica e técnica** das arquiteturas de agentes, desde os modelos clássicos até o estado da arte atual. Vamos destacar **quais elementos das arquiteturas tradicionais foram mantidos, transformados ou ampliados** nos agentes baseados em LLM, e **como a integração de modelos generativos modificou componentes clássicos** como percepção, deliberação (raciocínio/planejamento), execução de ações e interação.

Evolução e Continuidade dos Componentes de Agentes

Ao incorporarmos LLMs, vimos que **muitos componentes clássicos de agentes ganharam "nova vida"**. A tabela a seguir resume como os principais aspectos das arquiteturas de agentes evoluíram, comparando o paradigma clássico com o estado atual dos agentes generativos:

Componente	Arquiteturas Clássicas	Agentes com LLM (Atual)
Percepção	Sensores e entradas específicas de domínio; conversão de dados	LLMs ampliam a percepção para linguagem natural e conteúdo não estruturado. O agente pode receber descrições textuais ricas e

	brutos em símbolos ou atributos para uso interno. Frequentemente limitada a modalidades definidas (visão computacional especializada, parsing manual de texto estruturado). Ex.: um agente robótico obtém leituras numéricas de um laser e atualiza seu mapa interno.	entendê-las contextualmente. Além disso, com modelos multimodais, já é possível interpretar imagens ou áudio via LLM. A percepção torna-se **genérica e contextual**, muitas vezes englobando também informações obtidas via ferramentas (ex.: resultados de uma busca web retornados como texto). Em suma, a entrada pode ser um *prompt* em língua natural descrevendo a situação, que o LLM interpreta de forma flexível.
Estado/Memória	Limitada e estruturada. Agentes reativos não armazenam histórico; agentes model-based mantêm um estado simplificado (variáveis do mundo) atualizado a cada percepção. BDI e deliberativos possuem bases de crenças (fatos lógicos) que representam o conhecimento atual, porém sem memória extensa de eventos passados. Memória de longo prazo raramente é atualizada online, exceto via aprendizado offline.	**Memória extensa e dinâmica** via contexto de LLM e stores externos. Agentes LLM podem registrar interações completas, documentos e fatos em bancos de vetores, recuperando informações conforme necessário. Isso equivale a ter tanto memória de curto prazo (janela de contexto imediata do modelo) quanto **memória de longo prazo semântica** persistente. Por exemplo, um agente pode lembrar decisões tomadas horas ou dias antes ao buscá-las na memória vetorial. Essa memória ampliada permite consistência temporal e aprendizado contínuo (armazenando novos conhecimentos adquiridos) – algo difícil de implementar em arquiteturas clássicas.
Raciocínio/Deliberação	Fundamentado em **técnicas simbólicas**: lógica, inferência, *planning* clássico. Decisões tomadas via busca em espaço de estados (ex.: planejamento STRIPS) ou aplicação de regras de produção. Requer modelagem manual das ações e do mundo;	**Raciocínio em linguagem natural** gerado pelo LLM, frequentemente estruturado como *Chain-of-Thought*. O agente “pensa” gerando texto explicativo e decompondo o problema em subpassos. Isso substitui a necessidade de regras formais rígidas, pois o LLM aplica conhecimento aprendido para deduzir próximos passos. O planejamento torna-se muitas vezes implícito – emergindo da própria sequência de pensamento do modelo. Em casos necessários, o agente pode integrar um **planner simbólico**por meio de prompts (técnicas

	oferece transparência (planos explícitos) porém é suscetível a problemas de escalabilidade e manutenção. Alguns agentes utilizam modelos mentais (BDI) para organizar o raciocínio em termos de crenças, desejos e intenções, com planos predefinidos vinculados às intenções.	neuro-simbólicas), mas a tendência geral é confiar na **capacidade do LLM em decompor e solucionar** problemas novos sem programação especializada. Em suma, mantiveram-se os objetivos e a noção de sequência de ações, porém a *implementação do deliberar* mudou de algoritmos fixos para geração dinâmica de passos.
Execução/Ação	Ações predefinidas e limitadas, diretamente implementadas em código. Cada agente tem um conjunto de possíveis ações (movimentos motores, comandos de API) incorporado em seu design. Execução envolve escolher uma dessas ações e enviá-la aos atuadores ou sistemas. Por exemplo, um agente de e-commerce pode selecionar entre ações como “aprovar transação”, “negar transação”. Não há flexibilidade para criar novas ações em tempo de execução – o desenvolvedor define o repertório.	**Ações como uso de ferramentas ou geração de saídas complexas.** O agente LLM pode invocar *n* ferramentas externas (APIs, código) conforme necessário. A gama de ações é virtualmente ilimitada, pois adicionar uma ferramenta é simples e o LLM adapta-se à instrução de usá-la. Além disso, o agente pode **gerar novas sequências de ações ou código** sob demanda (ex.: escrevendo um script para realizar uma tarefa inédita). Isso representa uma extensão significativa da capacidade: os agentes modernos **podem alterar seu próprio conjunto de ações efetivas em runtime**, algo impossível para agentes fixos. Em contrapartida, requer mecanismos de verificação para evitar ações incorretas ou potencialmente danosas, dado o comportamento gerativo aberto do LLM.
Interação (Comunicação)	Interações interagentes estruturadas via *speech acts*formais e protocolos. Uso de idiomas artificiais (KQML, ACL) para mensagens com sintaxe e semântica definidas,	**Interações em linguagem natural livre**, tanto entre agentes quanto com humanos. Agentes LLM podem conversar entre si em um idioma humano (p. ex. inglês), coordenando-se através de diálogo contextual rico. Isso torna a configuração de colaboração multiagente mais ágil – dá-se um perfil ou persona a cada agente

	garantindo entendimento mútuo. A interação com humanos tipicamente limitada a interfaces fixas (menus, perguntas pré-definidas ou inputs simples). Em multiagentes, precisava-se projetar cuidadosamente ontologias comuns e sequências de diálogo esperadas para tarefas como negociação ou cooperação.	e eles mesmos ajustam a comunicação. Com humanos, a interface passa a ser conversacional (chatbots avançados capazes de entender solicitações complexas e fazer perguntas de esclarecimento). A vantagem é uma **comunicação mais intuitiva e flexível**; o desafio é a ambiguidade e a falta de garantias formais. Ainda assim, frameworks recentes permitem delimitar formatos dentro do diálogo (por exemplo, pedindo aos agentes para responderem em JSON ou outros formatos estruturados quando preciso), mitigando parcialmente a indeterminação. Em suma, a **capacidade social** persiste como fator-chave, mas agora realizada de modo mais próximo da comunicação humana natural.
Aprendizado /Adaptabilidade	Em muitos agentes clássicos, adaptabilidade era limitada ou ausente no tempo de execução – o comportamento era o implementado e pronto. Alguns incorporavam **aprendizado por reforço** ou outro esquema para ajustar estratégias ao longo de muitas iterações, mas tipicamente separado da lógica principal (ex.: um módulo de RL ajusta os parâmetros de seleção de ação de um agente de trading). Atualizar conhecimento implicava ou aprendizado offline (nova versão do agente) ou regras de aprendizagem específicas.	**Aprendizado online via iteração e auto-avaliação.**Agentes LLM podem melhorar durante a tarefa usando mecanismos como **self-reflection** (onde o agente avalia suas ações e reformula estratégias), ou incorporando feedback humano ou de ambiente diretamente no prompt subsequente. Há propostas de *Chain-of-Hindsight*, onde o agente revê suas respostas passadas com feedback e tenta melhorá-las progressivamente. Além disso, a atualização de memória vetorial com novas informações pode ser vista como forma de aprendizado: o agente *lembra* de fatos descobertos, não repetindo buscas ou evitando erros passados (ex.: *Voyager* armazenava skills aprendidas e não precisava reaprender a mesma habilidade). Embora os pesos do modelo LLM em si não se alterem (na maioria dos casos), o comportamento do agente *adapta-se* por meio do contexto e conteúdo acumulado. Em resumo, a adaptabilidade tornou-se mais integrada ao ciclo do agente – ele pode *aprender enquanto opera* de maneira mais direta do que agentes antigos conseguiam.

Tabela 5: *Evolução e Continuidade dos Componentes de Agentes*

Como podemos observar, **há um claro fio de continuidade ligando os conceitos clássicos aos agentes atuais**. Componentes essenciais – percepção, estado, raciocínio, ação, interação, aprendizagem – **mantêm-se relevantes**, mas foram **transformados ou estendidos por novas tecnologias**. O que antes era rígido e codificado manualmente, agora é frequentemente maleável e gerido pelo próprio modelo de linguagem ou por estruturas de dados genéricas. No entanto, os **objetivos últimos não mudaram**: um agente ainda precisa perceber corretamente, decidir racionalmente, agir eficazmente e, se aplicável, colaborar competentemente. Os LLMs possibilitaram fazê-lo em domínios bem mais amplos e com menos engenharia específica por parte de humanos, o que explica o entusiasmo em torno de *Agentic AI*.

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 1 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas Semanas anteriores, venho focado em estudar a área de Agentes Inteligentes, na qual construí guias de documento, diagramas e timelines para produzir informações e seções de tópicos nas quais eu pudesse voltar e consultar, sobre o campo escolhido.

Nesta Semana, me concentrei principalmente na parte prática de agentes inteligentes, na qual busquei e estudei mais sobre frameworks de agentes de IA, produzindo uma lista detalhada com seções por relevância de ferramentas utilizadas para construir e orquestrar agentes.

- **Criei uma nova guia para frameworks:** Estudo de tópico - Agentes de IA

Afim de explorar uma área mais específica na área de agentes, comecei a estudar sobre AgentOps, que por sua definição em algumas pesquisas de artigos, survey e fóruns como o da IBM, seria:

"O uso de técnicas operacionais e de manutenção, projetada especificamente para sistemas de agentes, sendo um conjunto emergente de práticas focadas no gerenciamento do ciclo de vida de agentes autônomos de IA."

- **Survey on AgentOps: Categorization, Challenges, and Future Directions -** Link Survey on AgentOps
- ***Criei uma guia de pontos chaves do Survey:*** *Estudo de tópico - Agentes de IA*
- **Repositório no GitHub:** AgentOps - GitHub

Problema Central: Anomalias em Sistemas de Agentes, classificadas como ***Anomalias Intra-Agente e Anomalias Inter-Agentes.***

Estrutura AgentOps:

1. Monitoramento (Monitoring)
2. Deteção de Anomalias (Anomaly Detection)
3. Análise de Causa Raiz (Root Cause Analysis - RCA)
4. Resolução (Resolution)

Identificar causa do problema
Análise de Causa Raiz (RCA): Taxonomia para casos de falha, divididas em três:

1. Centrada no Sistema
2. Centrada no Modelo
3. Centrada na Orquestração

Resolução do problema

- Resoluções Guiadas pelo Design do Sistema
- Resoluções Guiadas pela Otimização de Prompts

E por fim, sigo assistindo as aulas do curso da Udemy, Desenvolvimento de agentes de IA.

- Desenvolvimento de Agentes de Ia - UDEMY

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Me aprofundar mais e finalizar o curso da Udemy de Desenvolvimento de Agentes de IA;
Ler **AgentOps Pattern Catalogue: Architectural Patterns for Safe and Observable Operations of Foundation Model-Based Agents;**

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Frameworks para aplicação

Para cada framework, teremos uma introdução, recursos relevantes (como orquestração de agentes) e se é de código aberto ou não. Em seguida, apresentamos um ranking de relevância e impacto atual desses frameworks, destacando quais merecem maior atenção para **estudo.**

Principais Frameworks de Agentes de IA

Essa lista de Frameworks está organizada por ordem decrescente de lançamento, afim de entender as tecnologias mais usuais para a área, todos os componentes da lista seguem o link da documentação.

OpenAI Swarm / Agents SDK (2023) – **open-source (Swarm)** / **proprietário (Agents SDK)**

O OpenAI Swarm foi um framework experimental lançado em 2023 pela OpenAI focado em orquestrar múltiplos agentes de forma leve e ergonômica. Distribuído como código aberto (licença MIT), o Swarm permitia definir agentes via prompts e funções Python, coordenando a comunicação entre eles para executar tarefas em equipe. No entanto, ele não possuía gerenciamento de estado ou memória nativa, exigindo que o desenvolvedor passasse contexto manualmente entre agentes. Com a evolução da plataforma, a **OpenAI descontinuou o Swarm em favor do OpenAI Agents SDK,** uma solução closed-source e mais robusta destinada a desenvolvimento de agentes com **integração direta aos serviços OpenAI.**

O Swarm demonstrou conceitos importantes de orquestração multiagente (como delegação de tarefas e uso direto de funções Python para ferramentas) de forma simples, mas para uso prático recomenda-se migrar para o Agents SDK mais recente, que incorpora melhorias de desempenho e será mantido pela OpenAI. O Agents SDK faz parte do ecossistema proprietário da OpenAI, portanto não é aberto, porém traz a experiência do Swarm para um nível de produção.

Use case: *experimental e educativo para entender coordenação de agentes LLM; a versão atual (Agents SDK) visa aplicações profissionais integradas à API OpenAI.*

🔗 Link: [https://openai.github.io/openai-agents-python/
https://platform.openai.com/docs/guides/agents-sdk]

Botpress (2023) – **plataforma freemium, código proprietário**

O Botpress é uma plataforma de agentes de IA que oferece uma abordagem visual e de baixo código para orquestrar agentes. Focada em fluxos de trabalho de agentes baseados em LLM, ela permite definir o comportamento do agente por meio de fluxos (diagramas) onde cada nó representa uma tarefa ou ação. O desenvolvedor pode arrastar e soltar módulos, definindo etapas de raciocínio, memórias e integrações com ferramentas externas (e.g. CRM, e-mail, bancos de dados) sem precisar codificar tudo manualmente. Essa modularidade visual torna o Botpress atraente para equipes que desejam rapidez em prototipação e colaboração: é possível reutilizar lógicas entre diferentes agentes e manter auditoria das mudanças nos fluxos.

Embora seja gratuito para começar (plano free limitado) e facilite integrações, o Botpress é um produto comercial de código fechado ***(oferece planos pagos para recursos avançados).***

Use case: *criação de assistentes virtuais empresariais ou fluxos de atendimento integrando múltiplas ferramentas, sem exigir programação pesada. É ideal para times que preferem uma interface visual para projetar a inteligência do agente, garantindo controle de permissões e histórico de decisões.*

🔗 Link: [https://botpress.com/docs/home]

LangChain (2022) – **open-source (MIT)**

O LangChain é um dos frameworks mais populares e influentes surgidos na era dos modelos de linguagem (LLMs). Introduziu de forma pioneira o desenho modular de agentes baseado em cadeias (chains) de chamadas de LLM, ferramentas e memórias. **O LangChain funciona quase como um "sistema operacional"** para agentes com LLM: ele fornece componentes básicos para montar pipelines estruturados de raciocínio, gerenciando o contexto (memória), o uso de ferramentas (como consultas web, calculadoras, bancos de dados) e o planejamento de múltiplas etapas de forma integrada.

Por exemplo, é possível encadear passos onde o agente extrai dados, depois chama um LLM para análise, em seguida usa uma ferramenta externa, mantendo controle do estado durante o fluxo.

Sendo open-source em Python e JavaScript, o LangChain ganhou uma enorme comunidade (é atualmente o framework de agentes mais adotado pelos desenvolvedores) e continua evoluindo com centenas de integrações prontas (APIs, vetores, navegadores, etc.).

- **Pontos fortes:** grande flexibilidade e abrangência – suporta agentes simples de perguntas e respostas até fluxos complexos de Recuperação Aumentada por Geração (RAG) em múltiplas etapa.
- **Pontos fracos:** essa flexibilidade traz complexidade; iniciantes podem achar difícil escolher a abstração certa e montar fluxos sem se perder em tantos componentes.

Use case: *construção de agentes de conversação avançados, assistentes de pesquisa, ou qualquer aplicação com LLM que exija lógica passo-a-passo controlada e integração de ferramentas. Para começar, recomenda-se seguir exemplos prontos e casos de uso simples antes de tentar um agente completo do zero.*

🔗 Link: [https://python.langchain.com/docs/introduction/]

Microsoft Autogen (2023) – **open-source (MIT)**

O Autogen (da Microsoft) é um framework open-source focado em orquestração de múltiplos agentes LLM trabalhando em equipe. Permite criar agentes com diferentes papeis bem definidos (por exemplo: planejador, codificador, revisor) que se comunicam entre si e com humanos para executar uma tarefa colaborativamente. Escrito em Python, o Autogen provê uma infraestrutura de troca estruturada de mensagens: você define os agentes e suas personalidades/funções, e o framework gerencia o diálogo passo-a-passo, compartilhando resultados intermediários e memória conjunta conforme necessário. Um caso de uso típico é um agente planner que divide um problema em sub-tarefas, delega a um agente executor, e possivelmente pede avaliação a um agente revisor, formando um "time" de AIs. O Autogen suporta ainda intervenção humana no loop (HITL), caso se queira que um humano participe em alguma etapa crítica.

- **Pontos fortes:** coordenação avançada de agentes com funções especializadas, memória compartilhada do grupo e registro auditável de todas as trocas (transparência do raciocínio).
- **Desafios:** configurar múltiplos agentes e suas interações pode se tornar complexo, devendo-se começar com poucos agentes e casos simples antes de escalar.

O Autogen vem sendo bem recebido (é um dos projetos de agentes com crescimento mais rápido) e está integrado ao ecossistema Microsoft, colaborando com o Semantic Kernel para padrões de orquestração multiagente.

Use case: *tarefas que realmente exigem colaboração de agentes, como geração de código com validação (planner & coder), escrita de documentos com pesquisa (escritor & pesquisador), ou cenários em que se queira simular equipes humanas realizando etapas diferentes de um processo.*

🔗 Link: [https://microsoft.github.io/autogen/stable/]

LangGraph (2023) – **parcialmente open-source (biblioteca Python aberta, núcleo proprietário)**

O LangGraph é um framework introduzido em 2023 que estende as ideias do LangChain, permitindo definir fluxos de agentes em formato de grafo (DAG) ao invés de uma sequência linear de passos. A premissa é dar ao desenvolvedor controle de fluxo condicional, com bifurcações, loops e nós de decisão dentro do processo do agente. Enquanto o LangChain tradicional encadeia chamadas de forma sequencial, o LangGraph possibilita, por exemplo, que um agente repita certas etapas até uma condição ser satisfeita, ou tome rotas diferentes conforme resultados intermediários. Isso traz mais flexibilidade para agentes complexos que precisem revisitar passos ou escolher caminhos distintos. O LangGraph foi construído para integrar-se ao LangChain (herda seu ecossistema de ferramentas e memórias), portanto exige familiaridade prévia com LangChain. Ele é fornecido como uma biblioteca Python de código aberto, porém nota-se que a ferramenta completa do LangGraph é oferecida como software proprietário com recursos adicionais.

- **Pontos fortes:** orquestração avançada via grafos de tarefas (fluxos não-lineares) – ideal para workflows complexos de decisão ou quando se

deseja minimizar envolvimento desnecessário do LLM em cada passo, pré-definindo a sequência ótima de ferramentas.

- **Limitações:** não é indicado para projetos muito simples (seria "excesso de engenharia"), e exige cuidado para não sobrecarregar o design do grafo; aconselha-se começar com um grafo pequeno, testando parte a parte antes de expandir.

Use case: *agentes LLM em domínios com fluxos condicionais complexos, por exemplo: um agente de análise de dados que decide caminhos de processamento diferentes conforme o tipo de dado disponível, ou agentes com lógica de recuperação e agregação de informações que podem iterar até obter confiança no resultado.*

🔗 Link: [*https://langchain-ai.github.io/langgraph/*]

CrewAI (2023) – **open-source (Apache)**

O CrewAI é um framework leve em Python, lançado em 2023, projetado para construir sistemas multiagente com funções de equipe claramente definidas. Ele se destaca por permitir configurar rapidamente vários agentes que cooperam para um objetivo comum, cada um desempenhando um papel (por exemplo, um "Gerente" coordenando dois "Especialistas"). A filosofia do CrewAI é simplificar a orquestração: basta definir a crew (equipe) com seus membros, atribuir a cada agente um papel e objetivos compartilhados, e o framework gerencia a comunicação e execução sequencial ou paralela das tarefas pelos agentes. Ele oferece memória compartilhada da equipe para que os agentes possam sincronizar informações e suporta tanto execução passo a passo quanto agentes rodando simultaneamente. Por ser focado em cenários de colaboração estruturada, o CrewAI facilita muito prototipar comportamentos multiagente simples.

- **Pontos fortes:** configuração intuitiva e mínima – consegue muito resultado com pouca codificação em casos de agentes trabalhando lado a lado em tarefas lineares ou bem divididas.
- **Pontos fracos:** as abstrações são básicas e podem se tornar limitantes em fluxos complexos; se os agentes precisarem se adaptar dinamicamente a condições ou coordenação condicional, pode ser necessário trabalho extra fora do framework. O CrewAI é totalmente open-source e gratuito para uso,

embora seus criadores ofereçam planos de hospedagem gerenciada (CrewAI Studio).

Use case: *cenários didáticos ou protótipos rápidos de cooperação multiagente – por exemplo, um grupo de agentes dividindo tarefas de geração de conteúdo (pesquisa, escrita, revisão) ou brainstorming em conjunto. Também é útil em ambientes onde se quer estrutura de papéis bem definida e repetitiva, com agentes trocando mensagens para concluir algo em equipe.*

🔗 Link: [https://docs.crewai.com/]

Microsoft Semantic Kernel (2023) – **open-source (MIT)**

O Semantic Kernel da Microsoft, anunciado em 2023, é um framework de orquestração cognitiva que permite incorporar capacidades de agentes de IA em aplicações existentes, com foco em cenários corporativos. Diferente dos demais, o Semantic Kernel é mais um SDK do que um framework de "caixa fechada": ele fornece um conjunto de recursos modulares (em C# e Python) para construir agentes combinando LLMs + código convencional de forma harmoniosa. Nele, o desenvolvedor define "skills" (habilidades) que podem ser tanto funções nativas (código .NET/Python) quanto prompts para LLM. Essas habilidades são então encadeadas em planos semânticos que o agente seguirá. O framework cuida de gerenciar a memória de conversação, integração com plugins (por exemplo, conectar um e-mail ou banco de dados como ferramenta) e oferece suporte nativo a plugins do ecossistema .NET/Azure (o que o torna atraente para quem já usa tecnologias Microsoft). O Semantic Kernel enfatiza planejamento de objetivos: dado um objetivo, o agente pode buscar quais habilidades aplicar e em que sequência. Tudo é altamente extensível e focado em robustez empresarial, incluindo recursos de observabilidade, segurança e implantação local ou em nuvem Azure.

- **Pontos fortes:** arquitetura competência-driven (baseada em habilidades reutilizáveis), memória integrada e compatibilidade multimodal (texto, visão, fala) dentro de uma mesma agente. Por ser open-source (MIT), conta com uma comunidade ativa e integrou ideias do Autogen para suporte multiagente colaborativo.

- **Limitações:** há pouca interface visual ou automatização de alto nível – é uma ferramenta de desenvolvedor, exigindo design deliberado do fluxo (não é baixo código).

Use case: *ideal para aplicações empresariais que querem adicionar "inteligência de agente" às funcionalidades existentes – por exemplo, um agente dentro de um sistema corporativo que faz planos (séries de passos) envolvendo enviar e-mails, consultar banco de dados e chamar um LLM para redigir uma resposta ao usuário. Também excelente para quem deseja combinar chamadas de LLM com lógica e dados proprietários, mantendo controle total do ciclo (muito útil em projetos que precisam rodar on-premises com dados confidenciais).*

🔗 Link: [https://learn.microsoft.com/en-us/semantic-kernel/]

AutoGPT (2023) – **open-source (MIT)**

O AutoGPT não é exatamente um framework genérico, mas merece menção pelo impacto que teve em 2023 na popularização do conceito de agente autônomo. Trata-se de um projeto open-source que transforma o GPT-4 (ou outros LLMs) em um agente capaz de autoplanejar e executar tarefas orientadas por objetivos, sem supervisão humana constante. No AutoGPT, você dá uma meta de alto nível (por exemplo: "realizar uma análise de mercado sobre tópico X") e o agente irá decompor o objetivo em subtarefas, iterar sobre elas (buscando informações na web, gravando arquivos, chamando APIs) e ajustar o plano conforme necessário até completar o objetivo. Em outras palavras, é como ter um "estagiário de IA" que recebe uma missão e tenta cumpri-la passo a passo. O AutoGPT demonstrou o potencial (e os desafios) de deixar um LLM rodar autonomamente por longos ciclos.

- **Características principais:** sistema de plugins para permitir ações como navegar na web, manipular arquivos, executar código, memória de vetor para lembrar fatos e decisões anteriores, e estratégias de autorreflexão e reintento quando esbarra em becos sem saída.
- **Cuidados:** a autonomia vem com riscos – sem monitoramento, o agente pode se perder ou gerar ações indesejadas, por isso é importante revisar os logs de cada execução e ajustar os parâmetros (limites de iteração, por exemplo). O AutoGPT inspirou inúmeros outros projetos (BabyAGI, AgentGPT etc.) e

pavimentou o caminho para frameworks mais estruturados como os já citados acima.

Use case: *exploração de fluxos de trabalho totalmente automatizados – ideal para pequenos experimentos e POCs onde se queira ver até onde um LLM consegue chegar sozinho em uma tarefa complexa. Contudo, para aplicações de produção, costuma ser preferível optar por frameworks mais controláveis (como LangChain ou Autogen) e implementar autonomia de forma graduada.*

🔗 Link: [https://docs.agpt.co/]

Rasa (2017) – **open-source (Apache 2.0)**

O Rasa é um framework de código aberto estabelecido em 2017 focado na construção de assistentes virtuais e chatbots com compreensão de linguagem natural (NLU) e gerenciamento de diálogo. Embora não envolva LLMs por padrão (ele usa modelos de ML específicos para intent recognition, entidades e políticas de diálogo), o Rasa se encaixa no contexto de "agentes de IA" pois permite criar agentes conversacionais robustos e personalizáveis, com total controle dos dados e modelos. Com o Rasa, desenvolvedores definem intents, entities e histórias de conversa, e o framework cuida de treinar modelos que interpretam as mensagens do usuário e decidem a próxima ação do agente. Ele suporta pipelines personalizáveis (vários componentes de NLP intercambiáveis) e integrações com diversos canais (Telegram, Web, voz, etc.). Por ser open-source e auto-hospedado, o Rasa é muito adotado por organizações que precisam de chatbots on-premises ou altamente customizados, evitando depender de plataformas fechadas.

Destaques: forte ênfase em contexto e memória de diálogo, capacidade de transições complexas de estado e inclusão de ações personalizadas (código Python) como respostas do agente. Diferentemente dos frameworks LLM modernos, o Rasa exige mais configuração manual de exemplos de treino e regras, mas em contrapartida oferece previsibilidade e explicabilidade nas interações (cada decisão do agente segue as políticas definidas, ao contrário de um LLM que é uma caixa-preta).

Use case: *quando se deseja um assistente virtual com controle total sobre NLU e diálogo, ou quando os dados de treinamento são privados. Exemplos: chatbots de*

suporte ao cliente integrados aos sistemas internos da empresa, assistentes de voz com fluxo bem definido, etc. Vale notar que atualmente o Rasa pode ser complementado com LLMs (usando-os como componentes NLU ou de geração de respostas), mas seu core continua sendo a abordagem clássica de diálogo orientado a intenções.

🔗 Link: [https://rasa.com/docs/]

GAMA Platform (2009) – **open-source (GPL)**

O GAMA é uma plataforma de modelagem e simulação baseada em agentes (Agent-Based Modeling – ABM) originada na academia em 2007-2010, com lançamento público em 2009. Escrito em Java, ele oferece um ambiente completo (com IDE própria) para criar e executar simulações multiagente espaciais e multi-nível, sendo muito usado em pesquisa de ecologia, urbanismo, epidemiologia e outras áreas que estudam sistemas complexos emergentes. O GAMA possui sua própria linguagem de alto nível, GAML, projetada para que especialistas de domínio (não programadores profissionais) possam definir o comportamento de agentes e o ambiente de simulação de forma declarativa. Ele se destaca por recursos avançados de visualização 2D/3D e integração com dados geográficos (GIS) – por exemplo, é possível simular agentes pedestres se movendo em mapas reais de uma cidade, ou animais em um habitat com base em mapas de terreno.

Pontos fortes: visualização rica dos resultados, suporte a múltiplos paradigmas de modelagem (agentes reativos, cognitivos, por grades, etc.), e uma comunidade ativa em ABM que contribuiu com uma biblioteca extensa de modelos prontos.
Pontos fracos: a sintaxe GAML, apesar de poderosa, é específica e requer aprendizado adicional; além disso, o GAMA é focado em simulação off-line, não sendo a melhor escolha para sistemas multiagente interativos em tempo real.

Use case: *pesquisa e ensino envolvendo dinâmicas de sistemas complexos – por exemplo, modelar a propagação de doenças em populações, analisar trânsito urbano com milhares de agentes (carros/pedestres), simular interações ecológicas entre espécies, etc. Para quem precisa de simulações espaciais com agentes autônomos e visualização out-of-the-box, GAMA é uma solução madura e completa.*

🔗 Link: [https://gama-platform.org/wiki/Documentation]

JADE (Java Agent DEvelopment) (2000) – **open-source (LGPL)**

O JADE é um dos frameworks clássicos de sistemas multiagente. Desenvolvido originalmente pela Telecom Italia e disponibilizado à comunidade em 2000, o JADE fornece uma infraestrutura em Java, compatível com o padrão FIPA, para construção de agentes que comunicam entre si em ambientes distribuídos. Ele abstrai toda a complexidade de rede: ao usar JADE, você instancia agentes (objetos Java) que automaticamente podem descobrir uns aos outros, enviar mensagens assíncronas, negociar protocolos de interação e até migrar entre diferentes contêineres (nós) na rede. O JADE possui um modelo reativo de agente por padrão – ou seja, agentes respondem a mensagens/eventos de acordo com comportamentos registrados. Inclui também uma GUI de gerenciamento que permite inspecionar os agentes ativos, mensagens trocadas, etc.

- **Pontos fortes:** rico suporte à comunicação e coordenação – já vem com implementações de protocolos comuns (por exemplo, Contract Net para negociação), escalabilidade e arquitetura modular comprovadas em inúmeros projetos acadêmicos e industriais. O JADE consolidou ao longo dos anos uma extensa documentação e é bastante utilizado em pesquisa para protótipos de sistemas distribuídos inteligentes.
- **Pontos fracos**: a interface e estilo do JADE se tornaram um tanto desatualizados, e o ritmo de evolução diminuiu; além disso, a curva de aprendizado pode ser íngreme para iniciantes, dado conceitos de concorrência, threads e múltiplos agentes rodando em paralelo.

Use case: *problemas de sistemas distribuídos onde múltiplos agentes autônomos precisam coordenar ações – por exemplo, simular agentes em redes de telecom, sistemas de negociação distribuída, coordenação de robôs ou dispositivos IoT. Ainda é uma referência sólida para quem estuda teoria de agentes e quer implementar seguindo padrões FIPA de comunicação.*

🔗 Link: [*https://jade.tilab.com/documentation/tutorials-guides/*]

SPADE (Smart Python Agent Development Environment) (2018) – **open-source (MIT)**

O SPADE é um framework Python para sistemas multiagente, inspirado no JADE porém projetado para ser mais moderno e acessível. Ele utiliza o protocolo XMPP (Extensible Messaging and Presence Protocol) para comunicação, o que permite que agentes operem de forma distribuída e em tempo real, inclusive interagindo com usuários humanos via mensagens instantâneas. Com SPADE, cada agente é essencialmente um cliente XMPP que pode enviar/receber mensagens de outros agentes ou usuários, e o framework provê construções de alto nível para definir comportamentos (cíclicos, únicos, periódicos, máquinas de estado etc.) de forma assíncrona usando asyncio. Um diferencial do SPADE é oferecer presença e comunicação assíncrona nativas – agentes podem ficar online/offline, assinar publicações (pub/sub), e há até integração fácil com servidores XMPP externos ou embutir um servidor no próprio framework. Extensões opcionais permitem incorporar modelos BDI (crenças-desejos-intenções) via AgentSpeak, publicação/assinatura, e até integrar LLMs aos agentes (SpadeLLM) para dar capacidades de linguagem natural avançadas.

- **Pontos fortes:** facilidade de uso em Python, arquitetura totalmente distribuída (qualquer máquina rodando um agente SPADE se junta ao sistema via XMPP), e comunicação em tempo real adequada para ambientes dinâmicos (por exemplo, sistemas IoT inteligentes).
- **Pontos fracos:** requer configuração de servidor XMPP (embora venha um simples embutido para testes), o que adiciona complexidade; a documentação em alto nível já foi limitada (mas vem melhorando).

Use case: *sistemas multiagente distribuídos com comunicação instantânea, como coordenação de dispositivos IoT, agentes para cidades inteligentes (tráfego, sensores), ou mesmo agentes de bate-papo inteligentes integrados a mensagerias. O SPADE combina a familiaridade do Python com conceitos do JADE, sendo uma boa opção para quem quer implementar agentes distribuídos sem sair do ecossistema Python.*

🔗 Link: [https://spade-mas.readthedocs.io/]

PADE (Python Agent DEvelopment) (2018) – **open-source (MIT)**

O PADE é outro framework multiagente em Python, criado inspirado no JADE porém com foco em simplicidade e uso educacional. Surgiu por volta de 2018 como projeto acadêmico brasileiro, visando trazer para Python os conceitos de agentes distribuídos. O PADE utiliza sockets TCP/UDP para comunicação entre agentes (ao invés de XMPP), oferecendo uma infraestrutura leve para agentes concorrentes e comunicação remota. Sua API simplifica a criação de agentes reativos: herda-se de uma classe Agente e define-se comportamentos e métodos de comunicação.

- **Pontos fortes:** fácil de aprender (a API procura ser "pythônica" e amigável), suporta execução paralela de agentes e comunicação remota sem exigir muito do usuário, e é totalmente escrito em Python (facilitando extensões). É uma boa porta de entrada para quem achou o JADE complexo mas quer funcionalidades semelhantes em Python.
- **Pontos fracos:** ecossistema menor e menos maduro – possui menos recursos de visualização ou ferramentas prontas em comparação a JADE; por ser menos conhecido, a comunidade e documentação são mais limitadas.

Use case: *pequenas aplicações multiagente onde se quer rapidez de desenvolvimento e execução local/distribuída simples – por exemplo, simulações acadêmicas, protótipos de agentes interagindo em rede local, ou ensino de conceitos de agentes distribuídos para alunos com código Python.*

🔗 Link: [https://pade.readthedocs.io/en/latest/]

Jason / AgentSpeak (2003) – **open-source (GPL)**

O Jason é um intérprete e framework para programação de agentes cognitivos baseado na linguagem AgentSpeak(L), introduzido em meados dos anos 2000. Diferente dos frameworks acima (muitos orientados a agentes reativos ou baseados em planos imperativos), o Jason implementa o modelo BDI (Belief-Desire-Intention) de agentes, muito usado em IA simbólica. Nele, escreve-se o comportamento do agente em AgentSpeak – uma linguagem lógica de alto nível onde você especifica crenças do agente, seus desejos/objetivos e planos (regras gatilho -> ação) que ele deve seguir para alcançar intenções. O Jason cuida de fazer o motor de inferência

BDI rodar: o agente percebe eventos, atualiza crenças, escolhe qual plano aplicar e gera intenções a perseguir.

- **Pontos fortes:** oferece um nível de abstração elevado para raciocínio – ótimo para pesquisa em IA onde se quer testar arquiteturas cognitivas, raciocínio lógico, ontologias, etc. Suporta múltiplos agentes e comunicação entre eles baseada em troca de mensagens no estilo Speech-Act (FIPA). Tem sido usado pedagogicamente para ensinar conceitos de IA de agentes inteligentes deliberativos.
- **Pontos fracos:** a sintaxe AgentSpeak e o próprio paradigma BDI podem ser difíceis de assimilar para iniciantes não acostumados com programação lógica. Além disso, por operar em alto nível, não se integra tão facilmente com bibliotecas modernas de aprendizado de máquina ou ambientes gráficos (é mais para agentes "mentais").

Use case: *simulação de agentes com tomada de decisão complexa e baseada em regras lógicas, ou aplicações onde agentes precisam de comportamento "inteligente" explícito (e.g. em jogos de estratégia, cada agente com um conjunto de crenças e objetivos). Na prática, Jason permanece mais popular no meio acadêmico para experimentos com IA simbólica e ensino, do que em aplicações industriais modernas.*

🔗 Link: [https://jason-lang.github.io/doc/]

Mesa (2015) – **open-source (Apache)**

O Mesa é um framework em Python para modelagem baseada em agentes (ABM), muito utilizado para prototipação rápida de simulações. Lançado por volta de 2015, ele é inspirado em ferramentas como NetLogo, mas trazendo o poder do Python e sua integração com bibliotecas científicas. Com o Mesa, usuários podem definir agentes como classes Python com métodos de step (passo de simulação) e um espaço ambiente (grade, grafo ou contínuo). O framework gerencia o loop de simulação e fornece utilitários para visualização via browser (HTML/JS) e coleta de dados durante a simulação. É excelente para ensino e experimentos, pois permite ver em tempo real plots e grid de agentes evoluindo, com pouquíssimo esforço de programação.

- **Pontos fortes:** sintaxe simples e legível (tudo em Python puro), fácil integração com o ecossistema científico (NumPy, Pandas, etc.) para análise dos resultados, e uma comunidade crescente que compartilha exemplos (modelos de segregação, Game of Life, difusão de inovações, etc.).
- **Limitações:** não foi projetado para sistemas distribuídos ou em tempo real – os agentes rodam num único processo Python, o que significa que simulações muito grandes ou detalhadas podem enfrentar limites de desempenho. Além disso, carece de suporte nativo a visualizações 3D ou GIS sofisticadas (ao contrário do GAMA).

Use case: *ótimo para modelos conceituais e educativos – por exemplo, demonstrar fenômenos emergentes em sala de aula (como comportamento de formigas, dinâmica de populações, modelos econômicos simples). Também usado em pesquisa para rapidamente testar hipóteses em ABM antes de talvez portar para plataformas mais robustas. Para quem quer "brincar" com agentes de forma interativa e aproveitar Python, o Mesa é uma escolha acertada.*

🔗 Link: [https://mesa.readthedocs.io/]

Ambientes e Bibliotecas Multiagente (Reforço e Simulação)

Além dos frameworks de desenvolvimento de agentes acima, é importante citar algumas bibliotecas de ambiente e ferramentas relacionadas, que servem de campo de prova para agentes de IA:

- **OpenAI Gym / Gymnasium (2016):** Biblioteca aberta que se tornou o padrão para ambientes de treinamento de reinforcement learning. O Gym fornece uma coleção de ambientes (jogos simples, controle clássico, etc.) com uma API consistente para interação agente-ambiente (métodos reset(), step() etc.). Embora focado em agente único, estabeleceu as bases para extensões multiagente. Em 2023, o Gym foi sucedido pelo Gymnasium (mantido pela Farama), que continua a oferecer ambientes padronizados e compatíveis com a maioria dos algoritmos de RL.
- **PettingZoo (2020):** Biblioteca open-source em Python desenvolvida para padronizar ambientes multiagente de RL, análoga ao Gym porém com múltiplos agentes. O PettingZoo fornece tanto ambientes competitivos quanto cooperativos (ex: jogos multiplayer, simuladores) com API unificada, facilitando pesquisa em Multi-Agent Reinforcement Learning (MARL). Ele se integra bem com frameworks de treinamento como o RLlib (Ray). ***Use case:***

pesquisadores que querem um conjunto de ambientes para testar algoritmos MARL sob diferentes cenários (desde partidas de Xadrez até simulações simplificadas de trânsito).

- **Unity ML-Agents (2018):** Toolkit open-source da Unity que permite usar o motor de jogos Unity como ambiente para treinar agentes por RL ou aprendizado por imitação. Com o ML-Agents, desenvolvedores podem criar cenários 3D complexos (física realista, gráficos) e inserir agentes controlados por algoritmos de aprendizado. Há uma interface Python para interação com o jogo durante o treinamento. ***Use case:*** *treinamento de agentes para robótica simulada, jogos 3D, direção autônoma virtual, etc., aproveitando a fidelidade das simulações Unity.*
- **Plataformas de Simulação Social:** Além do já citado GAMA e do Mesa, outras ferramentas como NetLogo (1999, open-source) e MASON (2003, em Java) merecem menção. O NetLogo é um ambiente popular para simulação de agentes simples com uma linguagem própria extremamente fácil de usar (frequente no ensino médio e graduação para experimentar ABM). Já o MASON (da GMU) é um kit de simulação multiagente em Java focado em desempenho e extensibilidade (costuma ser usado em pesquisa quando se precisa simular dezenas de milhares de agentes eficientemente, abrindo mão de interface amigável).

Relevância e Impacto

Diante de tantas opções, é natural perguntar: quais frameworks são mais relevantes hoje e vale a pena focar para aprender? A resposta depende do que você pretende fazer com agentes de IA, mas de maneira geral alguns se destacam em termos de impacto atual:

1. **LangChain – (Mais Amplamente Utilizado):** Sem dúvida, o LangChain desponta como o framework de agentes de IA mais adotado e suportado pela comunidade atualmente. Sua relevância vem do amplo suporte a LLMs e ferramentas, e por ter catalisado muitas inovações em agentes de linguagem. Por que aprender? Se você pretende construir aplicações envolvendo modelos de linguagem (chatbots avançados, assistentes com ferramentas, sistemas de QA sobre dados), o LangChain oferece a base modular para isso. É open-source, tem rica documentação e um ecossistema em expansão – dominar seus conceitos (chains, agents, memory, etc.) lhe dará facilidade para criar soluções sob medida. Vale notar que, embora poderoso, ele nem

sempre é o mais eficiente em cenários multiagente complexos; ainda assim, é ponto de partida obrigatório nesse domínio.

2. **Microsoft Autogen – (Orquestração Multiagente de LLM de Última Geração):** O Autogen ganhou enorme tração em 2023, tornando-se uma referência para projetos de coordenação de agentes LLM. Com apoio da Microsoft, é um projeto open-source robusto (50k+ estrelas no GitHub) e tende a evoluir rapidamente integrando pesquisas de ponta. Por que aprender? Se seu interesse é em agentes colaborativos, times de agentes com diferentes especialidades ou inserir humanos no circuito, o Autogen traz padrões e ferramentas já preparados para isso. Ele também se integra ao ecosssitema Microsoft/Azure, útil em contextos corporativos. Aprender Autogen lhe ensinará sobre design de protocolos de comunicação entre agentes e gerenciamento de memória compartilhada – conceitos centrais para próxima geração de agentes autônomos.
3. **Semantic Kernel – (Integração de Agentes em Aplicações Reais):** Apesar de menos “famoso” entre desenvolvedores de hobby, o Semantic Kernel é altamente relevante para uso empresarial e de engenharia de software séria. Ele representa a convergência de agentes de IA com desenvolvimento de software tradicional, oferecendo um caminho para incorporar LLMs e agentes de forma confiável em produtos existentes. Além disso, muitos conceitos do SK (plugins, planners) são aplicáveis além dele, e a Microsoft vem impulsionando-o fortemente – investir tempo aprendendo-o agora pode posicioná-lo bem conforme mais empresas o adotem.
4. **Rasa – (Agentes de Conversação e NLU):** Em meio à empolgação com LLMs, o Rasa permanece extremamente relevante para chatbots e assistentes personalizados, especialmente onde dados e privacidade importam. Grandes empresas e startups utilizam Rasa para ter chatbots sob seu controle total (ex.: bancos, setor de saúde). Por que aprender? Se seu foco é assistentes virtuais conversacionais, suporte ao cliente via chat ou voz, e você quer entender os fundamentos de NLU e diálogo gerenciado, o Rasa é um must. Ele lhe ensinará a estruturar diálogos por intenções e estados, complementando bem o conhecimento de LLM (afinal, nem todo bot precisa de um modelo gigante se regras simples resolvem). Além disso, por ser open-source, você pode estendê-lo e eventualmente até combiná-lo com LLMs (há práticas de usar LLMs para sugerir intenções ao Rasa, por exemplo). Em suma, para bots conversacionais centrados no domínio do usuário, o Rasa ainda é um dos melhores frameworks a dominar.

5. **Frameworks de Aprendizado por Reforço / Multiagente em Simulação:** Se sua praia é IA em robótica, jogos ou simulações, vale focar em ferramentas como Unity ML-Agents e bibliotecas de RL (Ray RLlib, PettingZoo). Por que aprender? Esses frameworks não só permitem treinar agentes virtuais em ambientes complexos, como introduzem conceitos fundamentais de interação agente-ambiente, recompensa e aprendizagem. Por exemplo, dominar o ML-Agents da Unity pode abrir portas para desenvolver agentes de jogo ou soluções em visão computacional + controle autônomo dentro de simuladores – habilidades valiosas em pesquisa de robótica e AI para games. Já o RLlib (da plataforma Ray) é hoje um dos frameworks mais usados para treinar agentes em escala distribuída, inclusive com suporte a cenários multiagente. Se seu objetivo é trabalhar com Reinforcement Learning aplicado, dedicar tempo a essas ferramentas é importante.
6. **Runners-up e Outros:** Dependendo do nicho, outros frameworks podem ser altamente relevantes para você. Por exemplo, o CrewAI – se quiser rapidamente experimentar multiagentes LLM cooperando, talvez em um hackathon ou projeto pequeno, ele pode acelerar seu desenvolvimento devido à simplicidade de uso. O LangGraph – se você já é usuário de LangChain e sente falta de controles de fluxo complexos, aprender LangGraph pode elevar suas habilidades em orquestração de agentes. Para acadêmicos em MAS, conhecer JADE, SPADE e Jason continua sendo útil, pois muitos artigos e cursos os utilizam; mesmo não sendo tecnologias de ponta na indústria hoje, eles formam a base conceitual de sistemas multiagentes (e quem domina a teoria pode adaptá-la a novas ferramentas facilmente). E por fim, acompanhe as novidades: o campo de agentes de IA está fervilhando, com novos frameworks e atualizações surgindo constantemente (por exemplo, o OpenAI Agents SDK, sucessor do Swarm, deve ganhar destaque no ecossistema proprietário OpenAI).

A Survey on AgentOps: Categorization, Challenges, and Future Directions

Afim de entender mais e pesquisar a área de AgentOps, o artigo abordado define um problema emergente: embora os **sistemas de agentes baseados em Modelos de Linguagem Grandes (LLMs)** estejam se tornando cada vez mais avançados e

utilizados em aplicações industriais, eles frequentemente encontram falhas, instabilidade e problemas de segurança. Atualmente, faltam métodos sistemáticos para operar e manter esses sistemas complexos. Para preencher essa lacuna, o artigo introduz um novo conceito e uma estrutura operacional chamada **Agent System Operations (AgentOps)**.

"O uso de técnicas operacionais e de manutenção, projetada especificamente para sistemas de agentes, sendo um conjunto emergente de práticas focadas no gerenciamento do ciclo de vida de agentes autônomos de IA."

O Problema Central: Anomalias em Sistemas de Agentes

Os sistemas de agentes, apesar de sua flexibilidade e capacidade de realizar tarefas complexas, são propensos a uma ampla variedade de erros, que o artigo chama de "anomalias".

- **Definição Abrangente de Anomalia**: O artigo define uma anomalia como qualquer ocorrência durante as fases de **pré-execução, execução ou pós-execução** que leva à interrupção da tarefa ou a uma conclusão ineficaz. Isso é mais amplo do que em sistemas tradicionais, pois inclui problemas na configuração inicial (prompts) e resultados que parecem corretos, mas contêm erros (como alucinações).
- **Nova Classificação de Anomalias**: Uma das principais contribuições do artigo é uma nova taxonomia para classificar essas anomalias em duas categorias principais:
    - **Anomalias Intra-Agente**: Focadas em problemas que ocorrem *dentro* de um único agente. Isso inclui **anomalias de raciocínio** (como alucinações, onde o agente gera informações factualmente incorretas), **anomalias de planejamento** (planos irrealistas ou incorretos), **anomalias de ação** (erros ao usar ferramentas), **anomalias de memória** (esquecer instruções importantes) e **anomalias de ambiente** (problemas de recursos como alto uso de CPU).
    - **Anomalias Inter-Agentes**: Focadas em problemas que surgem da interação *entre* múltiplos agentes ou com o sistema como um todo. Isso inclui **anomalias de especificação de tarefa** (instruções ou configurações ambíguas), **anomalias de segurança** (ataques maliciosos a um agente ou à comunicação entre eles), **anomalias de**

comunicação (redundância de mensagens que confunde os agentes), **anomalias de confiança** (agentes confiando cegamente em informações de outros agentes, mesmo que sejam incorretas) e **anomalias de terminação** (tarefas que terminam prematuramente ou entram em loops infinitos).

A Solução Proposta: A Estrutura AgentOps

Para gerenciar essas anomalias, o artigo propõe o **AgentOps**, uma estrutura de operação e manutenção projetada especificamente para sistemas de agentes. Inspirado em práticas tradicionais como AIOps (IA para Operações de TI), o AgentOps é dividido em quatro estágios:

1. **Monitoramento (Monitoring)**
2. **Deteção de Anomalias (Anomaly Detection)**
3. **Análise de Causa Raiz (Root Cause Analysis - RCA)**
4. **Resolução (Resolution)**

Principais Diferenças entre AgentOps e Operações Tradicionais (AIOps)

As diferenças fundamentais surgem porque os sistemas tradicionais são determinísticos (baseados em código), enquanto os sistemas de agentes são **probabilísticos e estocásticos** (impulsionados por LLMs).
As principais diferenças em cada estágio são:

- **Monitoramento**: Os sistemas de agentes exigem o monitoramento de mais tipos de dados. Além de métricas, logs e traces tradicionais, é crucial coletar **Dados de Modelo** (parâmetros internos do LLM, como mapas de atenção) e **Dados de Checkpoint** (snapshots do estado do agente, como sua memória e ambiente), que permitem "voltar no tempo" (rollback) para corrigir falhas.
- **Deteção de Anomalias**: Os dados gerados por agentes não são inerentemente confiáveis. A detecção de anomalias precisa primeiro verificar a razoabilidade dos próprios dados (por exemplo, se uma resposta é uma alucinação) antes de usá-los para avaliar o estado do sistema.
- **Análise de Causa Raiz**: A análise em sistemas tradicionais foca em código ou infraestrutura. No AgentOps, a causa raiz pode ser muito mais abstrata, como uma **alucinação do LLM** em uma etapa específica do raciocínio ou um **prompt mal formulado**.

- **Resolução**: A correção em sistemas tradicionais é geralmente determinística. Em sistemas de agentes, devido à sua natureza aleatória, a resolução é um **processo complexo e iterativo**, que pode exigir testes contínuos (como testes A/B em prompts) e otimização para alcançar um estado ideal.

Metodologias e Estratégias Dentro do AgentOps

O artigo detalha abordagens específicas para cada estágio do AgentOps:

- **Análise de Causa Raiz (RCA)**: Propõe uma taxonomia para as causas de falhas, classificando-as em três dimensões:
    1. **Centrada no Sistema**: Falhas de infraestrutura e dependências externas (ex: falha de uma API).
    2. **Centrada no Modelo**: Limitações inerentes ao LLM (ex: alucinações, lacunas de conhecimento).
    3. **Centrada na Orquestração**: A "lógica suave" que guia o agente, como prompts e estratégias de planejamento (ex: um prompt ambíguo que leva a uma má interpretação).
- **Resolução**: As soluções são divididas em duas categorias principais:
    1. **Resoluções Guiadas pelo Design do Sistema**: Mudanças arquitetônicas para tornar o sistema mais robusto. Exemplos incluem **Redundância e Votação** (usar múltiplos agentes para a mesma tarefa e escolher a melhor resposta), **Guardrails** (regras rígidas que impedem o agente de realizar ações perigosas) e **Recuperação e Rollback** (restaurar um estado anterior seguro).
    2. **Resoluções Guiadas pela Otimização de Prompts**: Focadas em melhorar as instruções dadas ao agente. Exemplos incluem **Autocorreção e Introspecção** (o agente é instruído a revisar seu próprio trabalho em busca de erros) e **Re-especificação e Re-prompting** (refinar ou reescrever o prompt da tarefa para ser mais claro).

O artigo identifica que os sistemas de agentes baseados em LLM, apesar de seu potencial, sofrem com instabilidade e uma ampla gama de anomalias que as ferramentas operacionais existentes não conseguem gerenciar adequadamente. Ele propõe o **AgentOps como uma nova estrutura abrangente** para monitorar, detectar, analisar e resolver esses problemas de forma sistemática. Ao definir e classificar as anomalias e delinear as diferenças cruciais em relação às operações

de TI tradicionais, o trabalho estabelece uma base para o desenvolvimento futuro de sistemas de agentes mais robustos e confiáveis.

Metodologias de resolução

A resolução em AgentOps não é um conserto único e definitivo. Em vez disso, é um **processo cíclico e empírico de gerenciamento contínuo**. Isso ocorre porque uma correção em um agente pode ter efeitos inesperados e em cascata em todo o sistema (efeitos de segunda ordem), transformando, por exemplo, uma anomalia interna de um agente (*intra-agente*) em uma anomalia de comunicação mais complexa entre agentes (*inter-agente*). Por isso, qualquer solução é tratada como uma "hipótese provisória" que exige validação contínua, geralmente através de **Anotação Manual** por especialistas ou usando um **"LLM-como-Juiz"** para avaliar a taxa de sucesso da tarefa em larga escala.

As técnicas de resolução são divididas em duas categorias principais, cada uma focada em um aspecto diferente do sistema de agentes.

Resoluções Guiadas pelo Design do Sistema (System Design Driven Resolutions)

Essas são soluções arquitetônicas e fundamentais que visam construir sistemas mais robustos, seguros e resilientes. Elas são implementadas no nível da engenharia do sistema.

- **Redundância e Votação (Redundancy & Voting)**:
  - **O que é**: Esta técnica evita depender de uma única execução ou resposta de um agente. Em vez disso, executa múltiplos agentes ou múltiplas tentativas da mesma tarefa em paralelo e, em seguida, agrega os resultados. A decisão final é tomada por meio de mecanismos como votação majoritária ou classificação das melhores respostas.
  - **Como ajuda**: Mitiga falhas estocásticas (aleatórias), como inconsistências no planejamento ou alucinações, ao selecionar a resposta mais consistente e confiável de um conjunto de hipóteses.
  - **Anomalias alvo**: Raciocínio, Ação, Especificação de Tarefa, Memória.
- **Guardrails e Asserções (Guardrails & Assertions)**:

- **O que é**: São salvaguardas proativas que impõem restrições e regras rígidas sobre o comportamento de um agente antes ou depois de uma ação. Exemplos incluem exigir que a saída siga um formato específico (como JSON), validar se uma resposta faz referência a uma fonte de dados, ou proibir o agente de usar ferramentas perigosas.
- **Como ajuda**: Previne erros antes que eles aconteçam, garantindo que o agente opere dentro de limites seguros e previsíveis. É uma forma de controle que não altera fundamentalmente o modelo, mas limita suas ações.
- **Anomalias alvo**: Ação, Comunicação, Segurança, Memória, Execução.

- **Recuperação e Rollback (Recovery & Rollback)**:
  - **O que é**: Quando um agente entra em um estado de falha irrecuperável, essa técnica permite reverter o sistema para um "checkpoint" ou estado seguro anterior. Isso é possível porque os dados de checkpoint (como a memória do agente e o estado do ambiente) são monitorados e salvos.
  - **Como ajuda**: É essencial para lidar com falhas em cascata, loops infinitos ou becos sem saída no planejamento. Permite que o agente tente um caminho alternativo sem ter que reiniciar toda a tarefa.
  - **Anomalias alvo**: Ambiente, Terminação e Execução.
- **Adaptação de Política e Estratégia (Policy & Strategy Adaptation)**:
  - **O que é**: Em vez de corrigir uma falha individual, esta abordagem ajusta a estratégia de longo prazo que governa o comportamento do agente. Isso pode ser feito através de técnicas como aprendizado por reforço (reinforcement learning) ou aprendizado curricular (curriculum learning), que melhoram a capacidade de planejamento e raciocínio do agente com base no desempenho observado.
  - **Como ajuda**: Resolve deficiências estruturais no comportamento do agente, melhorando sua robustez e capacidade de generalização a longo prazo, em vez de apenas consertar erros pontuais.
  - **Anomalias alvo**: Raciocínio, Planejamento, Especificação de Tarefa, Ação.

Resoluções Guiadas pela Otimização de Prompts (Prompt Optimization Driven Resolutions)

Essas estratégias focam na "lógica suave" do agente, ou seja, na forma como as instruções (prompts) guiam seu processo de raciocínio e ação. São implementadas dentro do ciclo de raciocínio do agente.

- **Autocorreção e Introspecção (Self-Correction & Introspection)**:
  - **O que é**: Esta técnica aproveita a própria capacidade de raciocínio do LLM para que ele detecte e repare seus próprios erros sem intervenção externa. Isso é geralmente implementado por meio de otimização do prompt do sistema, incentivando o agente a "pensar passo a passo" (Chain-of-Thought), a autoavaliar suas ações (Reflexion) ou a verificar suas próprias respostas.
  - **Como ajuda**: É muito eficaz para resolver anomalias de raciocínio, como alucinações, ou falhas na execução de um plano. Cria agentes mais autônomos e robustos.
  - **Anomalias alvo**: Raciocínio, Planejamento, Especificação de Tarefa, Ação.
- **Re-especificação e Re-prompting (Re-specification & Re-prompting)**:
  - **O que é**: Muitas falhas de agentes originam-se de instruções de tarefa ambíguas, incompletas ou conflitantes fornecidas pelo usuário. A solução aqui é refinar e reescrever o prompt para ser mais claro e estruturado, uma prática conhecida como **engenharia de prompts**.
  - **Como ajuda**: A otimização dos prompts pode ser feita manualmente ou automaticamente, usando métodos baseados em aprendizado ou algoritmos evolucionários para encontrar as instruções mais eficazes para guiar o agente ao sucesso.
  - **Anomalias alvo**: Raciocínio, Planejamento, Especificação de Tarefa, Ação.

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 8 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas Semanas anteriores, venho estudando a área de agentes inteligentes, na qual estou me aprofundando principalmente na ***construção e orquestração de agentes,*** como também na ***operação e gerenciamento.***

Nessa Semana, li o artigo **AgentOps Pattern Catalogue: Architectural Patterns for Safe and Observable Operations of Foundation Model-Based Agents (2025):** AgentOps Pattern Catalogue

- Construí um estudo de tópico desse artigo, por justamente falar de algumas ferramentas e práticas na operação e gerenciamento de agentes.
- Estudo de tópico - Agentes de IA

Ressalto que nesse **AgentOps Pattern Catalogue,** é uma **coleção de soluções arquiteturais reutilizáveis e instanciáveis.**

A fim de tornar esse meu conhecimento mais aplicável e entender quais áreas já tenho uma abrangência, construí um "Pipeline/Roadmap" dividido em partes, na qual simplifica uma arquitetura de conhecimento de métodos e ferramentas de agentes inteligentes.

Nessa classificação usei os estudos de tópicos que construí ao longo do processo, mas me concentro principalmente na fase dos dias atuais, com estruturas mais voltadas para Agentes baseados em LLMs.

No pipeline de estrutura foram divididos em 11 níveis:

1. Programação e Prompting
2. Noções básicas de Agentes de IA
3. LLMs & APIs

4. Tools Use & Integração
5. Agent Frameworks
6. Orquestração e automação
7. Gerenciamento de Memória
8. Conhecimento e RAG
9. Deployment
10. Monitoramento e avaliação
11. Segurança e Governança

Link do pipeline: ***Residência - Agentes de IA***

A ideia é deixar esse fluxo o mais claro e modular possível, facilitando na atualização ou troca de blocos de ferramentas na estrutura de um agente.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

1. **Testar ferramentas de AgentOps**, utilizando a biblioteca própria.
2. **Como integrar essas ferramentas em agentes?** Aplicação e monitoramento na prática: https://github.com/AgentOps-AI/agentops
3. Criar **pipeline prático de ferramentas para monitoramento.**
4. Ler e estudar o paper **AGENTOPS: ENABLING OBSERVABILITY OF LLM AGENTS** AgentOps - Paper

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

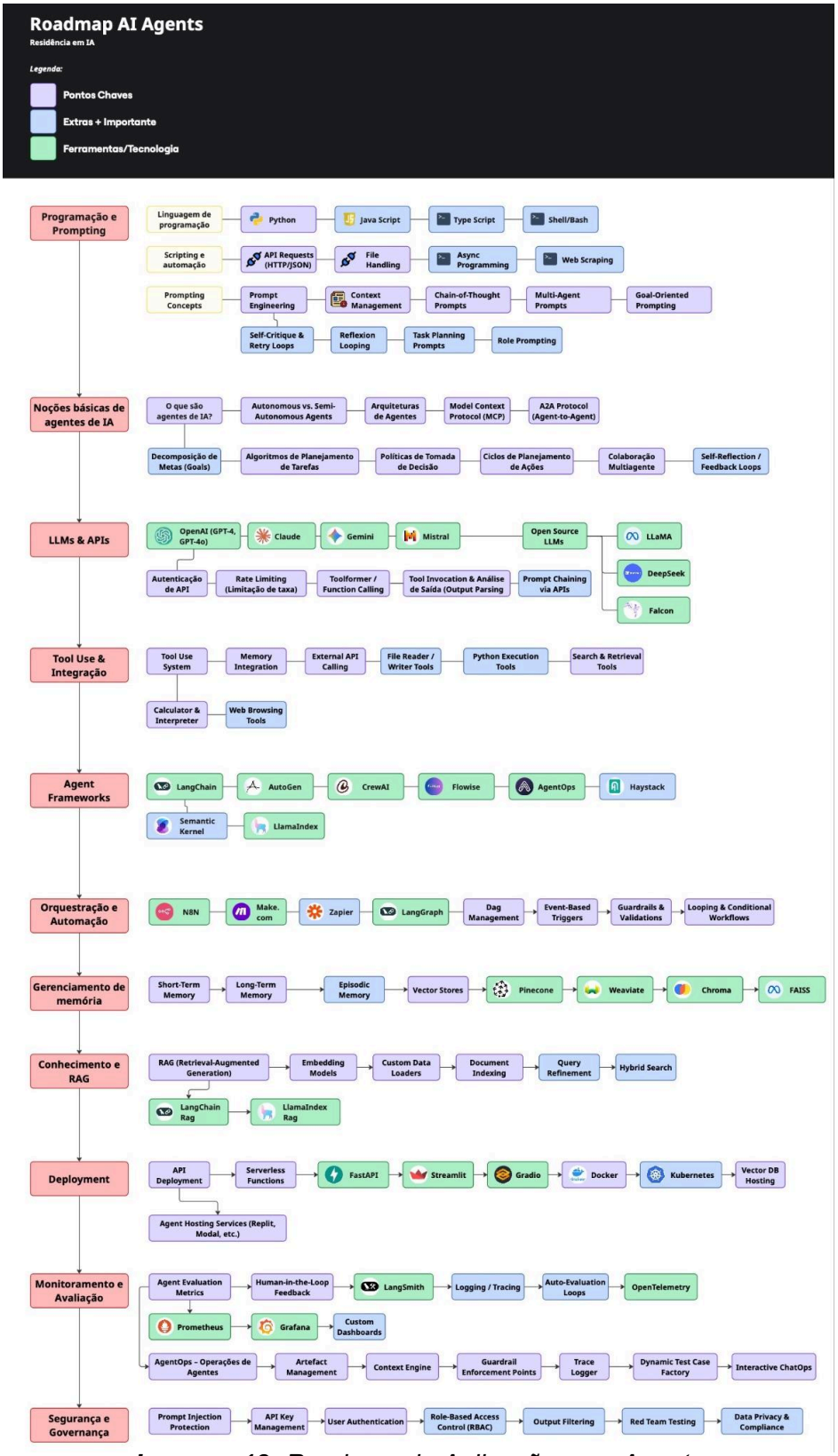


Imagem 18: *Roadmap de Aplicação para Agentes*

APÊNDICE 4

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 16 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas últimas Semanas, me aprofundei na área de estudo de Agentes Inteligentes, principalmente na parte de orquestração, monitoramento e fluxo de agentes.

Tema direcionado: **Engenharia Operacional de Agentes *(AgentOps)***

É como uma abordagem end-to-end que integra **projeto e orquestração** de agentes com **observabilidade, avaliação contínua e governança**, visando confiabilidade, controle de custos e segurança na produção.

Definindo o tema em 5 pilares específicos:

1. Projeto/Criação de Agentes
2. Orquestração
3. Operação/Observabilidade
4. Avaliação
5. Governança

Nessa Semana, me concentrei principalmente em tentar reproduzir os padrões e técnicas implementadas no artigo anteriormente apresentado ***AgentOps Pattern Catalogue: Architectural Patterns for Safe and Observable Operations of Foundation Model-Based Agents (Setembro, 2025).***

Utilizei dois repositórios para **produzir casos de estudo,** além do artigo de referência.

1. Primeiro utilizei o repositório do AgentOps que trás um guia muito bom do SDK desenvolvido para monitoramento e gerenciamento de fluxos de agentes.
    - https://github.com/AgentOps-AI/agentops?tab=readme-ov-file
2. Como eu precisava de um agente base para implementar os padrões, utilizei o repositório do MetaGPT, que por sua vez, é um **framework multi-agente que finge ser uma empresa de software**.

- https://github.com/FoundationAgents/MetaGPT

A ideia é gerenciar e operar, através de uma análise utilizando os padrões do AgentOps, para reproduzir resultados em cima do Framework do MetaGPT;

Por hora, consegui implementar a rastreabilidade das chamadas do LLM, os custos das chamadas, quantidade de tokens entre outras informações, todas ficam dispostas e salvas no dashboard do AgentOps, o painel que por sua vez facilita na hora de monitorar a aplicação.

- Adicionei algumas chamadas que eu fiz utilizando o framework do MetaGPT, conseguindo retornar o dashboard com os resultados, como nesse link:
- Estudo de caso - AgentOps

Com o intuito de entender como essas operações são executadas em escala em agentes, pesquisei e encontrei alguns materiais bem relevantes, como:

Encontrei um vídeo da própria Google, que explica desde DevOps até AgentOps, o vídeo ressalta como essa estrutura de gerência e monitoramento vem evoluindo com o tempo e sendo aplicada agora atualmente em Agentes, dando ênfase principalmente em arquiteturas de aplicações do Google.

- Descrevem estruturas arquitetônicas detalhadas e processos de fluxo de trabalho, incluindo ferramentas, catálogos e estratégias de avaliação necessárias para a construção e produção eficiente de agentes de IA, cobrindo interações de turno único e multi-turno (memória).
- AgentOps: Operationalize AI Agents

Produzi um estudo de tópico do vídeo, por que ele traz diversas arquiteturas de aplicação na qual implementam etapas e padrões de AgentOps.

- Estudo de tópico - Agentes de IA

Encontrei também um artigo chamado ***AgentOps: Enabling Observabily of LLM Agents (Dezembro, 2024):***

- https://arxiv.org/pdf/2411.05285

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

1. Reproduzir outros padrões determinados no artigo ***AgentOps Pattern Catalogue,*** visto que só consegui elaborar a **Gestão de artefatos** com a **chamada de LLM do agente,** no MetaGPT**.**
2. Ler o artigo ***AgentOps: Enabling Observabily of LLM Agents*** e produzir estudo de tópico.
3. Implantar ***AgentOps em outros modelos de Agentes.***
4. ***Atualizar Roadmap de Agentes: Miro AgentOps - Residence***

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Estudo de caso - AgentOps

Implantação do MetaGPT com AgentOps

A seção a seguir demonstra **prints do painel do AgentOps** após a execução do MetaGPT com AgentOps. Instrumentei o MetaGPT com AgentOps e registrei a execução como uma sessão com **traço completo**: cada **chamada de LLM**, **ferramenta** e **sub-agente** ficou **catalogada** com **tempo, custo, tokens e versão de configuração**, viabilizando **replay, auditoria e comparação** entre execuções.

Visão geral da sessão, visa identificar a execução (ID), início/fim, duração total e custo agregado. Isso resume o "ciclo de vida" daquela corrida de agentes/LLM. Esses resumos são consequência de registrar spans estruturados e correlacioná-los numa execução única.

- **Árvore de spans (trace)**: cada nó representa um passo (ex.: "Agente X → chamada LLM → chamada de ferramenta → resposta"). Esse **Trace Logger** é um padrão de AgentOps que grava spans "replay-fiéis" em pontos-chave (planejamento, chamada de modelo, tool call, escrita em memória), com vínculo às versões de configuração/política — o que torna a auditoria e a comparação entre execuções possíveis.
- **Detalhes de cada LLM call**: você verá **tokens, latência e custo** por chamada, além de trechos do prompt/saída (normalmente sob redações). Esse nível de prova (proveniência + configuração comportamental) é justamente o que o padrão recomenda para **diagnóstico e detecção de drift**.
- **Sumários e gráficos**: contagem de chamadas, distribuição por agente/etapa, latência/custo por passo. Esses agregados materializam a categoria "**Monitoring & Observability**" do catálogo AgentOps (ex.: Trace Logger; Cumulative Risk Ledger) para tornar o comportamento comparável e auditável.
- **Reprodução/Auditoria**: como os spans guardam **configuração e versões** (prompts, políticas, ferramentas), você consegue **replay** e comparação entre ambientes/runs — essência do AgentOps para operação explicável em produção.

Link AgentOps: ***https://app.agentops.ai/overview***

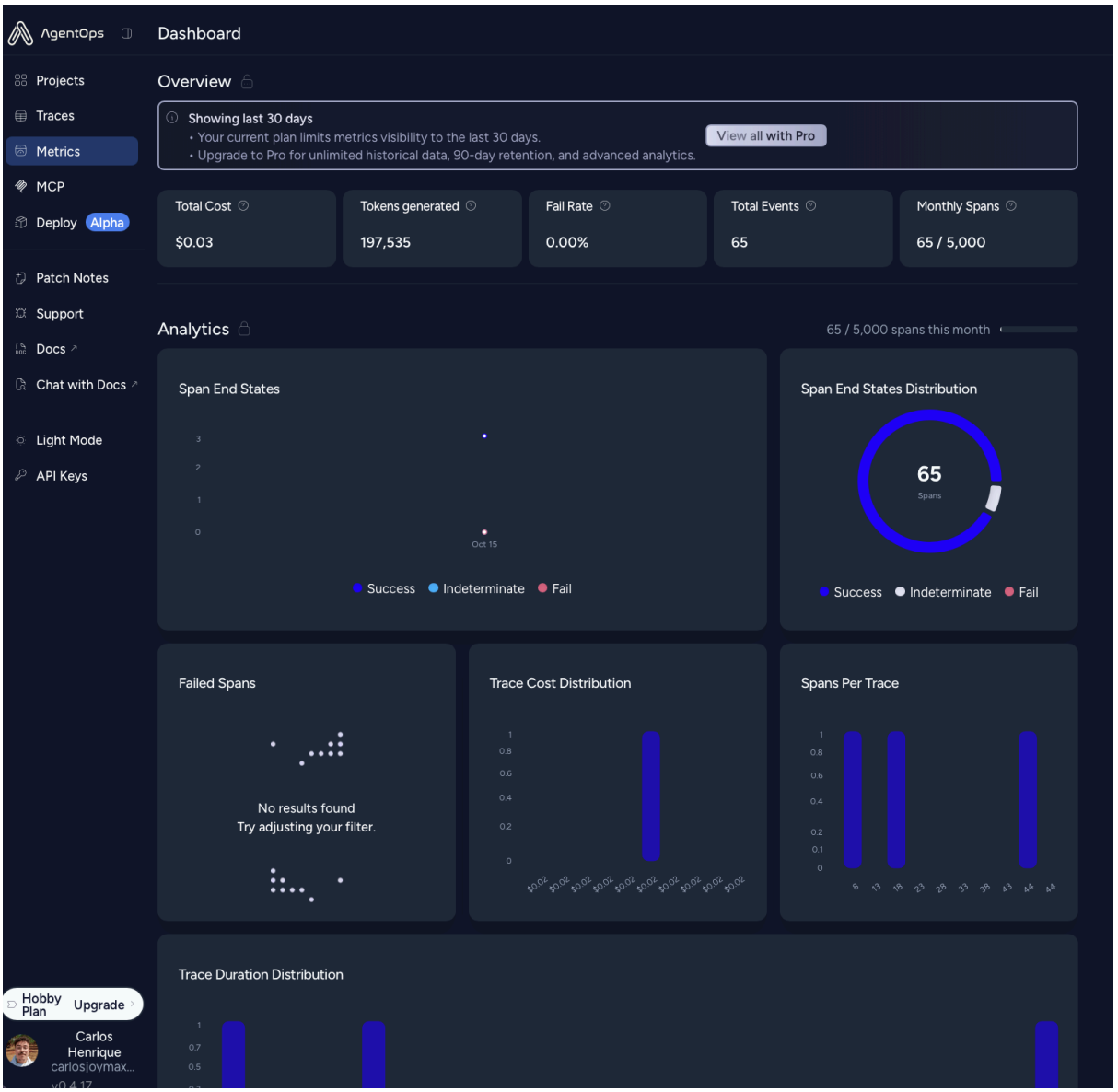


Imagem 19: *Dashboard Framework AgentOps*

Essa imagem traz um estudo inicial de como podemos mapear e observar os nossos agentes orquestrados. Além de conseguir entender e mapear todo o fluxo de entradas e saídas, com relação a custo e operações.

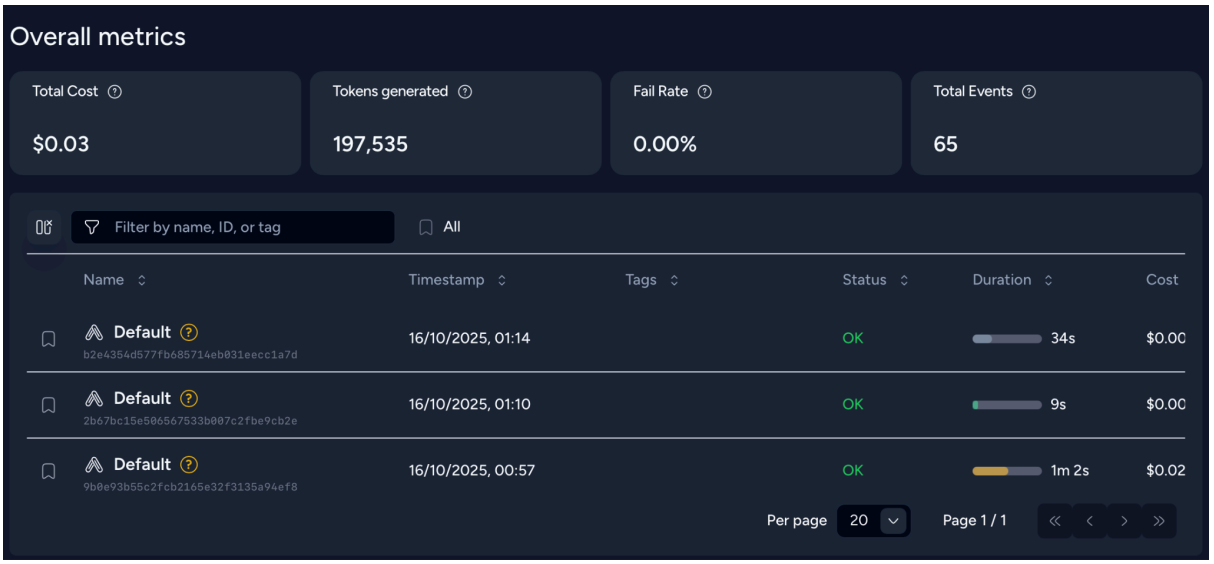


Imagem 20: *Dashboard de operações*

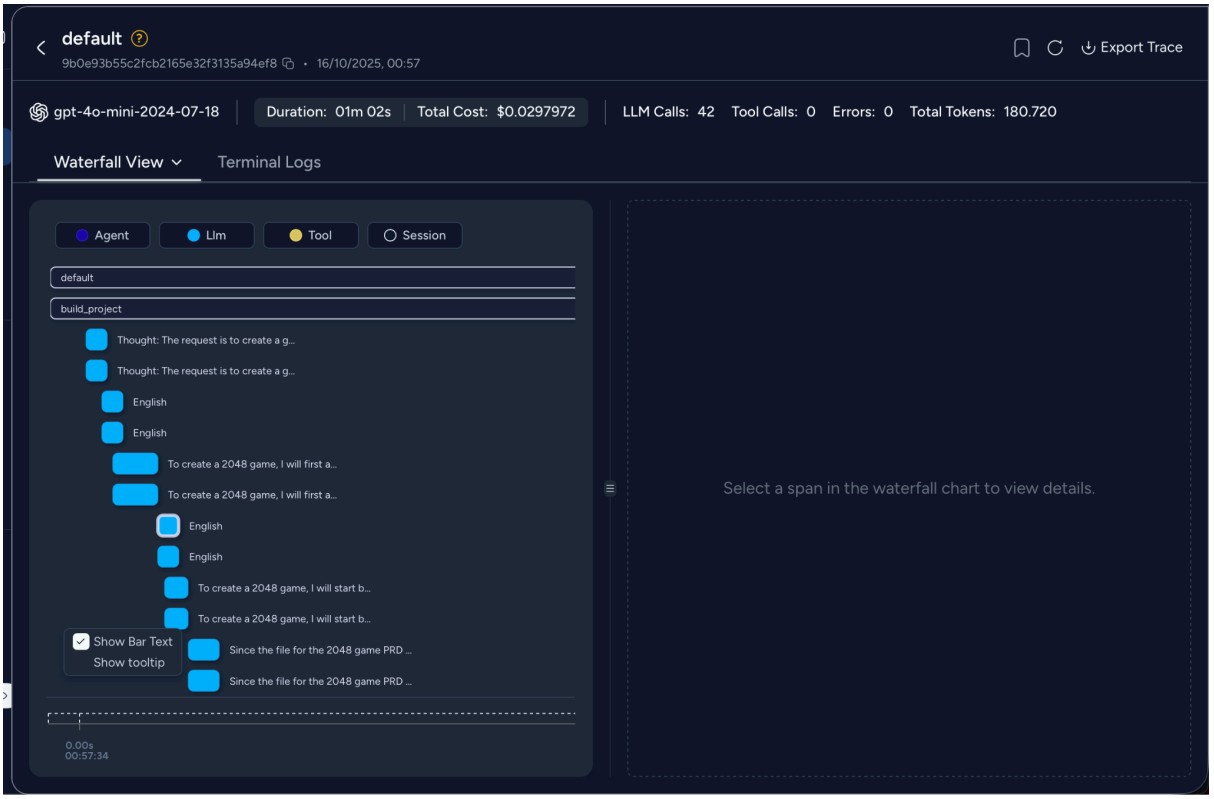


Imagem 21: *Dashboard Framework AgentOps LLMs rastreados*

DevOps à AgentOps

Começando pelas definições operacionais existentes e evoluindo para as práticas específicas de Gen AI e Agentes.

A Distinção das Arquiteturas Ops

A fundação de todo o domínio Ops é o **DevOps**. A partir dele, surgiram extensões necessárias para ambientes mais complexos e não determinísticos:

Arquitetura	Fundação e Definição	Principais Características / Distinção
DevOps	A base de tudo. Segue as melhores práticas de desenvolvimento (repositórios, pipelines CI/CD, testes automáticos).	Foco em desenvolvimento de software tradicional.
MLOps	Uma extensão do DevOps, inventada para produção de soluções de **Machine Learning**.	O ambiente de ML é **não determinístico**, o que exige operações adicionais, como a avaliação de modelos. Combinação de pessoas, processos e tecnologia para operacionalizar soluções de ML de forma eficiente.
GenAIOps	Generative AI in Operations. É o termo usado para **produzir aplicações** baseadas em IA generativa, não apenas um único modelo. Atua como um **guarda-chuva**que inclui categorias mais novas, como PromptOps, RAGOps e **AgentOps**.	Foca nos consumidores de modelos, exigindo novas práticas como Engenharia de Prompt e Recuperação de Contexto (RAG).
AgentOps	Operacionalização de agentes de IA.	Estende o GenAIOps focando em ferramentas, avaliação de seleção de ferramentas, e gerenciamento de memória (curto e longo prazo).

Tabela 6: *Comparação entre Ops*

É essencial entender as práticas de MLOps e GenAIOps, pois os conceitos de AgentOps são derivados delas.

Arquitetura MLOps Tradicional

MLOps visa reduzir o *time to value* (tempo até o valor) através da padronização (repositórios, pipelines CI/CD, infraestrutura) e da **templatização** (visando cerca de 80%).

A arquitetura MLOps é dividida em quatro ambientes fundamentais:

1. **Infraestrutura (Platform Team):** Arquitetos de Nuvem e SMEs de Segurança criam a infraestrutura (VPCs, redes, funções IAM), geralmente com Terraform.
2. **Data Lake/Data Mesh:** Engenheiros e proprietários de dados ingerem, pré-processam e catalogam os dados, tornando-os acessíveis a outros.
3. **Ambiente (Data Science & ML Engineering):**
    - **Experimentação (Sandbox):** Cientistas de Dados testam novos algoritmos sem interação com dados de produção.
    - **Desenvolvimento, Staging e Produção:** Os notebooks Python são automatizados em pipelines para pré-processamento, treinamento e pós-processamento. O *Staging* realiza testes de estresse, integração e ponta a ponta, antes da aprovação manual para *Produção*.
4. **Governança:** Ambiente centralizado que armazena modelos, dados, *performance* e artefatos. Inclui o **Registro de Modelos (Model Registry)**, que armazena todas as versões de modelos, e serve como ponto de verdade para proprietários de produtos e auditores.

Processos e Arquitetura GenAIOps

O GenAIOps adiciona uma camada de aplicação e novas *personas* (Engenheiros de Prompt, Engenheiros de IA, Desenvolvedores de Aplicativos).

Processos GenAIOps Chave:

- **Seleção e Avaliação de Modelos:** Após filtrar os modelos (checando termos/condições), é necessário realizar uma avaliação profunda para casos de uso específicos, usando dados próprios, pois a alta pontuação em *leaderboards*não garante sucesso. A seleção considera a **precisão**, **latência** e **custo**.
- **Catálogo de Prompts (Prompt Catalog):** Um local para armazenar os prompts iniciais e, posteriormente, centenas de prompts de teste. Deve incluir **controle de**

versão completo (linhagem) para rastrear como as mudanças nos prompts afetam o desempenho.

- **Back-End da Aplicação Gen AI:** Componentes cruciais para a operacionalização:
  - **Guardrails:** Filtram as entradas e saídas do LLM para evitar toxicidade, alucinação, e perguntas irrelevantes. Podem ser implementados usando ferramentas como o **Model Armor** no GCP.
  - **Mecanismos de Caching:** Armazenam respostas frequentes para evitar chamadas desnecessárias ao LLM, reduzindo custo e latência.
  - **Auto Context Retrieval (RAG/Agentes):** Permite que o chatbot obtenha detalhes em tempo real (histórico de interações, dados do cliente).
  - **Monitoramento:** Verificação contínua de toxicidade, alucinação e **fundamentação (grounding)** para garantir que as respostas sejam baseadas no contexto recuperado.

Arquitetura GenAIOps (Gen AI Application Development):

Esta arquitetura se baseia no MLOps, substituindo ou estendendo a camada intermediária para projetos de aplicação Gen AI:

- O **Catálogo de Prompts** é armazenado centralmente (geralmente no projeto Data Lake) e usado para avaliação automática no ambiente *Staging* e para coletar feedback de usuários em *Prod*.
- O back-end da aplicação se conecta ao Data Lake para implementações RAG (transformação de dados em vetores) ou a agentes e ferramentas de dados (como APIs para BigQuery), permitindo a recuperação de contexto em tempo real.
- O *front-end* (interface do usuário, como um chatbot) é implantado (frequentemente usando **Cloud Run**) e interage com o back-end.
- Os **Gen AI Testers** e **Prompt Testers** interagem com o ambiente *Staging* para adicionar mais testes ao Catálogo de Prompts.

AgentOps e Operações

O AgentOps foca em como construir e gerenciar agentes, que são essencialmente um **prompt que instrui um modelo sobre como chamar diferentes ferramentas**.

Mecânica Central do Agente:

O **núcleo do agente** é uma combinação de ferramentas disponíveis, instruções sobre como usá-las, e o modelo base.

1. O usuário fornece uma consulta de entrada.

2. O Núcleo entende se precisa chamar uma função e com quais parâmetros.
3. O *runtime* executa a função real (por exemplo, chama uma API).
4. A saída da função é passada de volta ao Núcleo.
5. O Núcleo constrói a resposta final usando a informação recuperada.

Avaliação de Agentes:

A avaliação de agentes estende o GenAIOps, exigindo um conjunto de dados que inclua não apenas o prompt e a resposta, mas também: **a ferramenta correta a ser chamada**, **os parâmetros corretos** e uma **resposta de exemplo**.

Métricas adicionais de avaliação de agentes incluem:

- Taxa de sucesso na **seleção de ferramentas**.
- Taxa de sucesso na criação dos **parâmetros da função**.
- Se o modelo precisa chamar uma ferramenta ou não (e o sucesso dessa decisão).

Ferramentas e Otimização:

Para otimizar o desempenho do agente, as ferramentas devem ter descrições e parâmetros adequados. É vital que as funções **não se sobreponham ou contradigam**, devendo realizar tarefas específicas para evitar confusão do modelo. Agentes devem ser tratados como **microsserviços** com acesso apenas a ferramentas específicas.

Registro de Ferramentas (Tool Registry):

Este é um componente centralizado para armazenar todas as ferramentas criadas (APIs, código, ferramentas de dados). O registro armazena metadados (declaração, versão, desempenho) e facilita a **reutilização** e padronização, cuidando da autenticação e autorização centralmente. Ferramentas de código (funções determinísticas) podem ser padronizadas em repositórios e transformadas em APIs (expostas via Cloud Run) para serem armazenadas no registro.

Multi-Turn e Memória:

- **Multi-Turn (Múltiplas Rodadas):** O agente pode exigir múltiplos *loops* (várias chamadas de função e respostas intermediárias) antes de gerar uma resposta final, especialmente para tarefas complexas. Deve haver um limite máximo de iterações para evitar loops infinitos.
- **Memória de Curto Prazo (Short-Term Memory):** Acompanha eventos dentro de uma única interação, residindo perto do agente (ambiente de produção).
- **Memória de Longo Prazo (Long-Term Memory):** Armazenamento persistente (Data Lake, BigQuery, Graphs) de interações concluídas ou tópicos importantes, essencial

para clientes que retornam após longos períodos. É crucial combinar a Memória de Longo Prazo com o sistema RAG para recuperar apenas o contexto relevante (por exemplo, sobre um tópico específico como "hipoteca").

Multi-Agentes (Multi-Agents):

Para tarefas maiores, múltiplos agentes pequenos (atuando como microsserviços) são orquestrados. A orquestração pode ser feita por um **Agente Roteador**, fluxos sequenciais, paralelos ou dinâmicos.

Em ambientes multi-agente complexos, são introduzidos:

- **Catálogo de Agentes (Agent Catalog):** Para catalogar todos os agentes disponíveis no negócio.
- **Catálogo de Templates de Agentes (Agent Template Catalog):** Para armazenar código para *templates* de agentes e acelerar o desenvolvimento.

Arquitetura Final e Estruturas

A arquitetura final de AgentOps incorpora o gerenciamento de memória: A Memória de Curto Prazo está no ambiente de Produção, próxima ao agente, enquanto a Memória de Longo Prazo fica no Data Lake, alimentando o sistema RAG. O **Registro de Ferramentas (Tool Registry)** é centralizado no ambiente de Governança.

Estruturas de Agentes (Agent Frameworks) simplificam o desenvolvimento, integrando modelos, ferramentas e memória em uma única orquestração, permitindo que os desenvolvedores se concentrem na engenharia de prompt e no design de ferramentas.

A arquitetura "gigante" definitiva para grandes empresas integra todas as camadas: Aplicação Gen AI/Agentes (camada inferior), Data Lake, a Camada MLOps (necessária se houver fine-tuning ou treinamento de modelos customizados) e, no topo, a Governança de IA. A camada MLOps pode ser abstraída se forem utilizados modelos prontos de fornecedores (como o Gemini).

Tutoriais Links

GenAIOps end to end starter pack. → https://goo.gle/e2e-gen-ai-app-starte...
Google ADK walkthrough: Your step by step development tutorial → https://goo.gle/4kFyHbv
GenAI in production: MLOps or GenAIOps? → https://goo.gle/42JLCSz

Resources:
GenAIOps: Operationalize generative AI - A practical guide → https://goo.gle/4noMMLO

Taming the tool chaos: The generative AI aents & tool registry → https://goo.gle/4pG27sV
Operationalizing Generative AI on Vertex AI using MLOps → https://goo.gle/3Kkhqax

APÊNDICE 5

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 22 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas últimas Semanas, venho me aprofundando na área de estudo de Agentes Inteligentes, principalmente na parte de orquestração, monitoramento e fluxo de agentes.

Na qual eu defini o Tema: **Engenharia Operacional de Agentes *(AgentOps)***

Elaborando principalmente: Estudos de tópicos para agregar no conhecimento gerado, através disso, criei uma linha evolutiva, lista de arquiteturas, roadmap, termos e palavras chaves e classificações de agentes. Assim como a leitura de alguns artigos, papers e surveys que estão concentrados principalmente nessa planilha: Pesquisa Científica

Através disso, essa Semana fiquei focado principalmente em executar os testes relacionados ao uso da ferramenta de Observability and DevTool platform for AI Agents, no caso AgentOps.

Li e construi um resumo sobre o artigo ***AgentOps: Enabling Observabily of LLM Agents*** *(2024)****, que foca principalmente*** em sistemas autônomos que usam Grandes Modelos de Linguagem (LLMs) para perceber o contexto, raciocinar, planejar e executar fluxos de trabalho, usando ferramentas externas e bases de conhecimento para atingir objetivos humanos.

Exemplos de sucesso incluem Devin, ChatDev e SWE-agent.

Estudo de tópico - Agentes de IA

Pesquisei também sobre ferramentas que são utilizadas na área de observabilidade de agentes e de LLMs, na qual encontrei principalmente cerca de 17 ferramentas, que estão listadas nesse documento a seguir:
Estudo de tópico - Agentes de IA

Enfim, sobre a parte da implementação do AgentOps, no framework do MetaGPT, consegui executar a parte Trace Logger (Registrador de Rastreamento), junto com a Gestão de Artefatos,

mas estou tendo problema na hora da rastreabilidade do Trace Logger.

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

1. *Atualizar Roadmap de Agentes: Miro AgentOps - Residence*
2. *Resolver o erro do Trace Logger, utilizando Logs e endpoints da aplicação, entender o por que dos passos dos agentes não estão sendo rastreados.*
3. *Buscar o que posso melhorar do tema além de simplesmente orquestrar, gerenciar e monitorar o agente/LLM.*

Observação: **[caso precise fazer alguma observação, de qualquer "natureza"]**

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Paper: *AgentOps: Enabling Observabily of LLM Agents*

O Contexto: Por que Precisamos de AgentOps?

Agentes LLM (ou *LLM Agents*) são sistemas autônomos que usam Grandes Modelos de Linguagem (LLMs) para perceber o contexto, raciocinar, planejar e executar fluxos de trabalho, usando ferramentas externas e bases de conhecimento para atingir objetivos humanos. Exemplos de sucesso incluem Devin, ChatDev e SWE-agent.

No entanto, a adoção desses agentes gera desafios únicos que comprometem a segurança da IA:

1. **Complexidade:** Eles são sistemas compostos de IA, integrando LLMs com ferramentas externas e gerando artefatos dinâmicos (como planos e objetivos).
2. **Autonomia e Não-Determinismo:** Agentes agem por conta própria e interagem dinamicamente com ambientes externos. Devido à natureza probabilística dos LLMs, eles podem produzir resultados variados com as mesmas entradas, o que torna o comportamento imprevisível e difícil de repetir.
3. **Evolução Contínua:** Eles aprendem e se adaptam ao longo do tempo, o que pode introduzir novos desafios para manter a conformidade com os objetivos de segurança e qualidade.
4. Para enfrentar esses desafios, o artigo argumenta que é fundamental habilitar a **observabilidade**—a capacidade de obter *insights* acionáveis sobre o funcionamento interno de um agente, analisando entradas e saídas (artefatos de *runtime*) conforme fluem pelos pipelines operacionais.

Metodologia Utilizada: O Estudo de Mapeamento Sistemático

Para desenvolver a taxonomia do AgentOps, os autores realizaram um **Estudo de Mapeamento Sistemático** (SMS).

Objetivo do Estudo

O objetivo principal do SMS foi analisar as ferramentas existentes relacionadas ao AgentOps para entender suas funcionalidades e limitações, e identificar os artefatos dos agentes e os dados associados que devem ser rastreados para habilitar a observabilidade.

Fontes de Dados e Palavras-Chave

Foram utilizadas múltiplas fontes para garantir uma cobertura ampla:

• **GitHub:** Pesquisa abrangente usando palavras-chave como "AgentOps", "Agent AND DevOps", "Agent AND LLMOps", e "Agent AND Observability".

• **Google Search:** Pesquisa direcionada para capturar ferramentas proprietárias usando a *string* ("LLM" AND "Agent" AND "Observability" AND "Tool").

Critérios de Seleção (Foco no Agente)

As ferramentas selecionadas tinham que cumprir critérios rigorosos, com um foco crucial no agente, e não apenas no LLM (Critério I2):

• **I1:** Suportar recursos de observabilidade (como monitoramento e rastreamento).

• **I2:** Suportar rastreamento específico de agente ou rastreamento de aplicação LLM que possa ser aplicado a agentes (e não apenas rastreamento no nível do LLM).

• **I3 & I4:** Ter uma versão de lançamento formal e documentação pública online acessível.

Ao final deste processo, **17 ferramentas** relevantes para AgentOps foram identificadas e analisadas.

Principais Ferramentas e Funcionalidades do AgentOps

As 17 ferramentas identificadas no estudo são diversas. Algumas são mais focadas em Agentes (como **AgentOps** e **AgentNeo**), enquanto outras são líderes em aplicações LLM mais amplas (como **Langfuse**, **PortKey** e **Arize**).

O AgentOps oferece um conjunto de recursos essenciais, resumidos em 7 categorias principais:

Categoria	Descrição e Importância
1. Tracing (Rastreamento)	É a maneira mais direta de permitir a observabilidade. Captura o processo inteiro — desde o *prompt* do usuário até o resultado final — permitindo que os desenvolvedores sigam o processo de tomada de decisão passo a passo, identificando a origem de erros ou comportamentos inesperados. Todas as 17 ferramentas listadas implementam rastreamento.
2. Monitoring (Monitoramento)	Acompanhamento contínuo de métricas de desempenho (como latência e custo de token), depuração de problemas e identificação da causa raiz de resultados inesperados. Inclui rastreamento de custo do LLM.
3. Guardrails (Barreiras de Proteção)	Regras predefinidas e restrições para limitar as ações do agente, garantindo segurança e comportamento previsível. Essenciais para a segurança da IA por design, rastreando a ativação e os resultados das ações de proteção.
4. Evaluation (Avaliação)	O processo de avaliação vai além da resposta final, sendo crucial monitorar e rastrear as etapas de execução e as saídas intermediárias. Isso inclui Avaliação Passo a Passo (se a

	ferramenta apropriada foi selecionada) e Avaliação de Trajetória (se a sequência esperada de ações foi seguida).
5. Prompt Management (Gestão de Prompts)	Envolve o controle de versão e rastreamento de *prompts*, *prompt playground* (para testar e comparar modelos) e detecção de ataques ou vazamentos (como ataques de injeção de código ou segredos embutidos em *prompts*).
6. Customization (Personalização)	Capacidade de estender o agente adicionando kits de ferramentas, conectando-se a várias bases de conhecimento vetoriais e integrando modelos personalizados ajustados (*fine-tuned*).
7. Feedback	Coleta de *feedback* humano (explícito, como polegar para cima/baixo, ou implícito, como tempo gasto na página) para avaliar a qualidade e melhorar o design do agente.

Tabela 7: *AgentOps Categorias Essenciais na observabilidade*

A Taxonomia do AgentOps: Os Artefatos Rastreáveis

O artigo apresenta uma taxonomia detalhada que serve como modelo para projetar ferramentas AgentOps, focando nos artefatos (ou *spans*) que compõem o rastro de execução de um agente (o *trace*). Um *trace* revela o processo inteiro, e é composto por um ou mais *spans* aninhados (cada um representando uma operação).

Os principais *spans* (artefatos) que devem ser rastreados e sua função são:

1. **Agent Span:** Representa a solicitação inicial. Inclui metadados sobre o papel e a persona do agente (como ele se comporta).
2. **Reasoning Span (Raciocínio):** Captura o processo de raciocínio. Inclui o **Contexto** (informação relevante), o **Conhecimento Recuperado** (dados de fontes externas) e o **Resultado** (os pensamentos ou a conclusão gerada).
3. **Planning Span (Planejamento):** Registra a fase de planejamento. Define o **Objetivo**, as **Restrições** e pode incluir **Planos Históricos** que influenciam a estratégia atual.
4. **Workflow Span (Fluxo de Trabalho):** Detalha como o plano é executado. Inclui uma lista de **Tasks** (tarefas) e suas **Dependências**, garantindo uma execução lógica e eficiente.
5. **Task Span (Tarefa):** Uma unidade discreta de trabalho dentro do *Workflow*. Contém a **Descrição da Tarefa** e o **Status** atual (concluído, pendente).
6. **Tool Span (Ferramenta):** Registra as interações com ferramentas externas usadas para auxiliar na execução da tarefa. Captura o **Nome da Ferramenta**, a **Versão** e as **Configurações** usadas.

7. **LLM Span:** Captura as interações diretas com o Grande Modelo de Linguagem. Inclui o **Nome/Versão do LLM** e os **Parâmetros** aplicados (como a temperatura e o limite de tokens).
8. **Evaluation Span (Avaliação):** Avalia a correção e a qualidade das ações do agente. Inclui **Casos de Teste**, **Métricas de Teste** e os **Resultados do Teste**.
9. **Guardrail Span (Barreira de Proteção):** Documenta as ações de proteção aplicadas (como bloqueio, validação ou filtragem) e os artefatos visados (como objetivos e ferramentas).

O AgentOps estabelece uma estrutura sistemática baseada em **rastreamento extensivo de artefatos** para permitir que os desenvolvedores monitorem, depurem e garantam a segurança dos agentes LLM complexos e autônomos.

Mapeamento Sistemático: Ferramentas e Funcionalidades

As ferramentas são categorizadas como focadas em **Agentes** (sistemas autônomos complexos) ou **Aplicações LLM**(aplicações que utilizam LLMs, mas podem não ter o mesmo nível de autonomia do agente). As funcionalidades são listadas de acordo com as 7 categorias principais do AgentOps (Customization, Prompt Management, Evaluation, Feedback, Monitoring, Tracing e Guardrails).

Nome da Ferramenta	Escopo (Scope)	Funcionalidades Principais (Baseadas em)	Detalhamento da Função
Agenta	Aplicações LLM	C, PM, E, F, M, T	Suporta uma ampla gama de funcionalidades de observabilidade para aplicações LLM. Focada no desenvolvimento, gerenciamento de *prompts*, avaliação e rastreamento. **Não possui Guardrails**.
AgentNeo	Agentes	C, E, M, T	Uma ferramenta voltada especificamente para **Agentes**. Suporta Customização, Avaliação, Monitoramento e Rastreamento. **Não oferece**Gerenciamento de *Prompts* ou Feedback. *Nota: Guardrails é uma implementação planejada.*

AgentOps	Agentes	**C, PM, E, F, M, T, G**	É uma solução abrangente de AgentOps, oferecendo **suporte para todas as 7 funcionalidades principais** (Customização, Gerenciamento de *Prompts*, Avaliação, Feedback, Monitoramento, Rastreamento e **Guardrails**).
AGIFlow	Aplicações LLM	C, PM, E, M, T	Foca em Customização, Gerenciamento de *Prompts* e observabilidade central (Monitoramento e Rastreamento). **Não suporta Feedback nem Guardrails**.
Arize	Aplicações LLM	E, M, T, G	Uma das soluções líderes. Focada em Avaliação, Monitoramento, Rastreamento e, notavelmente, é uma das poucas que **integra Guardrails**. **Não oferece** Customização ou Gerenciamento de *Prompts*.
DataDog	Agentes	M, T	Uma plataforma comercial de observabilidade. Foca principalmente nas funcionalidades básicas de observabilidade: **Monitoramento e Rastreamento**. **Não suporta** Customização, Gerenciamento de *Prompts*, Avaliação, Feedback ou Guardrails para agentes.
Dify	Aplicações LLM	**C, PM, E, F, M, T, G**	Uma ferramenta popular (51.6k estrelas) que suporta **todas as 7 funcionalidades principais**.
Helicone	Aplicações LLM	E, M, T	Focada em Observabilidade e Avaliação. Suporta Monitoramento, Rastreamento e Avaliação. **Não suporta** Customização, Gerenciamento de *Prompts*, Feedback ou Guardrails.
Laminar (lmnr)	Aplicações LLM	PM, E, F, M, T	Suporta Gerenciamento de *Prompts*, Avaliação, Feedback e observabilidade

			central (Monitoramento e Rastreamento). **Não oferece** Customização ou Guardrails.
Langfuse	Aplicações LLM	PM, E, F, M, T	Uma solução líder para aplicações LLM (6.5k estrelas). É forte em **Feedback** (definindo tipos explícitos e implícitos), Gerenciamento de *Prompts* e Avaliação. **Não oferece** Customização ou Guardrails.
LangSmith	Aplicações LLM	PM, E, F, M, T, G	Uma ferramenta que se destaca por introduzir dimensões detalhadas de **Avaliação** (avaliação passo a passo e avaliação de trajetória). Suporta Gerenciamento de *Prompts*, Observabilidade e **Guardrails**. **Não oferece**Customização.
LangTrace	Aplicações LLM	E, M, T	Focada nos pilares de Observabilidade (Monitoramento e Rastreamento) e Avaliação. **Não suporta** Customização, Gerenciamento de *Prompts*, Feedback ou Guardrails.
Lunary	Aplicações LLM	PM, E, F, M, T, G	Uma ferramenta muito completa que suporta a maioria das funcionalidades, incluindo Gerenciamento de *Prompts*, Feedback, Observabilidade e **Guardrails**. **Não oferece** Customização.
PortKey	Aplicações LLM	E, M, T	Uma solução líder (6.3k estrelas). Foca nas funcionalidades centrais de observabilidade e Avaliação. **Não suporta** Customização, Gerenciamento de *Prompts*, Feedback ou Guardrails.
TraceLoop	Aplicações LLM	M, T	Foca principalmente nas funcionalidades básicas de observabilidade: **Monitoramento e Rastreamento**. **Não oferece** Customização, Gerenciamento de *Prompts*, Avaliação, Feedback ou Guardrails.

Trulens	Aplicações LLM	PM, E, F, M, T, G	Suporta Gerenciamento de *Prompts*, Avaliação, Feedback, Observabilidade e **Guardrails**. **Não oferece** Customização.

Tabela 8: *Mapeamento Sistemático: Ferramentas e Funcionalidades*

Legenda das Funcionalidades (Key Features) - Tabela 8

• **C** (Customization): Provisionar, personalizar e estender agentes com *toolkits*, bases de conhecimento vetoriais e modelos ajustados (*fine-tuned*).

• **PM** (Prompt Management): Controle de versão, *prompt playground* (comparação de modelos) e detecção de injeção/vazamento de *prompts*.

• **E** (Evaluation): Testar agentes contra *benchmarks* e avaliar o desempenho em diversas etapas (resposta final, passo a passo, ou trajetória).

• **F** (Feedback): Coletar *feedback* humano (explícito ou implícito) para avaliar a qualidade e melhorar o design do agente.

• **M** (Monitoring): Acompanhamento contínuo de métricas (latência, custo de token) e *dashboards* de análise de agentes.

• **T** (Tracing): Rastrear cada *span* do agente (cadeia completa, recuperação, chamada LLM, chamada de ferramenta) para depuração passo a passo.

• **G** (Guardrails): Regras e restrições predefinidas para limitar as ações do agente, garantindo segurança e comportamento previsível.

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 5 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas Semanas anteriores, venho me aprofundando na área de estudo de **Agentes Inteligentes,** principalmente na parte de orquestração, monitoramento e fluxo de agentes.

Semanas 1 e 2 - Definição, Escopo Inicial e Fundamentação Teórica;
Semana 3 - Aprofundamento Arquitetural, estudos de referência bibliográficas de produção de um documento e evolução das arquiteturas;
Semana 4 - Classificação: Mapeamento de arquiteturas iniciais e criação de uma classificação abrangente de arquiteturas a nível hierárquico e por camadas;
Semana 5 - Prática: Foco na parte prática de agentes e no estudo aprofundado de AgentOps; **Semanas 6 e 7** - Padrões Operacionais e Pipeline: Estruturação do conhecimento em um roadmap operacional de 11 níveis (focado em Agentes de LLMs) para orquestração e gerenciamento;
Semana 8 - Implementação reprodução dos padrões de AgentOps utilizando o AgentOps SDK e o framework MetaGPT;

Na qual eu defini o Tema: **Engenharia Operacional de Agentes *(AgentOps)***

Nessa Semana, me concentrei principalmente na parte de experimentos, então para abranger mais conhecimento também, foquei em ler alguns artigos como:

Evaluation and Benchmarking of LLM Agents: A Survey (2025):
Apresenta uma taxonomia bidimensional que organiza o trabalho existente ao longo:
(1) objetivos de avaliação - o que avaliar, como comportamento, capacidades, confiabilidade e segurança do agente
(2) processo de avaliação - como avaliar, incluindo modos de interação, conjuntos de dados e benchmarks, métodos de computação métrica e ferramentas.

Além disso, principalmente visando esses modelos e taxonomias de avaliação dentro desses artigos, produzi um plano de ação para executar um baseline de observabilidade e avaliação de

Agentes LLM com MetaGPT, esse estudo tem como o intuito analisar uma abordagem experimental para melhoria de desempenho.

O Plano de ação para execução então foi definido em 4 camadas:
✔ 1 - Instrumentação base, configurando o ambiente com a instalação do MetaGPT, OpenTelemetry e AgentOps;

- Criação do dataset de consultas, selecionando ~100 perguntas que o agente deve responder, para fins de teste.

✔ 2 - Framework de implementação de um agente RAG no MetaGPT para responder um conjunto específico de consultas;

- Construção do RAG, execução de queries (MetaGPT roles, Chroma/FAISS).

A serem feitos ainda:

3 - Análise e Otimização, gerando uma análise com Dashboards utilizando AgentOPS para visualizar a latência, custo e taxa de sucesso, utilizando DSPy ou Auto PDL para otimizar prompts;

- **Auto PDL:** Medium AutoPDL
- AutoPDL: Automatic Prompt Optimization for LLM Agents:
- **DSPy**: https://www.datacamp.com/pt/blog/dspy-introduction
- **DSPy:** Compiling Declarative Language Model Calls into Self-Improving Pipelines

4 **- Benchmarking, definida as métricas de trajetória (exact match, in-order match, any-order match) e de resposta final, taxa de acerto, uso correto de ferramentas e eficiência.**

- Serão comparadas a fim de estudo as versões do agente (baseline, instrumental e otimizada) comparadas na execução das 100 tarefas.
- Scripts de métrica (Vertex metrics), 100 testing, Pandas/Matplotlib.

Para mais detalhes deixei o plano de ação em um página do estudo de tópico de Agentes:
Estudo de tópico - Agentes de IA

Além disso, a fim de reproduzir o mesmo plano de ação, tentei executar com um framework de Agentes diferentes do MetaGPT, utilizando também o TheAgentCompany, que mede o progresso do desempenho desses agentes LLM na execução de tarefas profissionais do mundo real.

Consegui reproduzir até a primeira parte de instrumentação dos testes, mas cada uma das tarefas desse repositório contém uma imagem docker específica para cada tarefa, o que acabou ocasionando alguns erros de versão com o kernel, por mais que o TheAgentCompany fosse mais elaborado, acabava que cada teste mesmo corrigindo alguns erros, demoravam muito nas execuções, com a imagem do repositório por si só tem 40 GB, sem contar as tarefas.
Então com base nisso, acabei deixando de lado o TheAgentCompany e focando na implementação com o

MetaGPT.

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

1. **Analisar e otimizar o baseline implementado;**
2. **Criar repositório e subir os códigos e experimentos;**
3. **Documentar testes e insights.**

Observação: **[caso precise fazer alguma observação, de qualquer "natureza"]**

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

APÊNDICE 6

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 13 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Carlos Henrique Rodrigues de Jesus

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas Semanas anteriores, venho me aprofundando na área de estudo de **Agentes Inteligentes,** principalmente na parte de orquestração, monitoramento e fluxo de agentes.

Semanas 1 e 2 - Definição, Escopo Inicial e Fundamentação Teórica;
Semana 3 - Aprofundamento Arquitetural, estudos de referência bibliográficas e estudo de tópico;
Semana 4 - Classificação: Mapeamento de arquiteturas iniciais e criação de uma classificação; **Semana 5** - Prática: Foco na parte prática de agentes e no estudo aprofundado de AgentOps; **Semanas 6 e 7** - Padrões Operacionais e Pipeline: Estruturação do conhecimento em um roadmap operacional de 11 níveis (focado em Agentes de LLMs) para orquestração e gerenciamento;
Semana 8 - Implementação reprodução dos padrões de AgentOps utilizando o AgentOps SDK e o framework MetaGPT;
Semana 9 - Implementação do MetaGPT utilizando CrewAI, remapeamento do plano de ação e início dos testes no baseline.

Na qual eu defini o Tema: **Engenharia Operacional de Agentes *(AgentOps)***

Nesta Semana, *a principal dedicação foi à conclusão dos experimentos de observabilidade propostos no MetaGPT. Contudo, foram necessários ajustes na arquitetura do framework* para melhor compreensão da estrutura do MetaGPT e para contornar erros de atualização.

Para isso, todo o *framework* de agentes foi migrado para utilizar o **crewAI** como ferramenta principal, o que acarretou na alteração da estrutura do plano de ação.

Adicionalmente, foi criado e documentado um repositório contendo todo o código e a aplicação. Um estudo comparativo baseado nos resultados dos *baselines* também foi construído.

Plano de ação: ***Estudo de tópico - Agentes de IA***
Repositório: ***CarlosHenrique21/CrewMetaGPT/tree/main***

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Estudo de comparação: ***Estudo de tópico - Agentes de IA***

Camada 1: Fundação e Baseline

O que fiz: *Preparei o ambiente e criei dataset de teste com 5 projetos reais.*

Projetos-teste:

1. *Todo List CLI*
2. *URL Shortener API*
3. *Weather CLI*
4. *Password Generator*
5. *Markdown to HTML Converter*

Camada 2: Sistema Multi-Agente (CrewAI)

O que fiz: *Implementei 5 agentes especializados em pipeline sequencial.* ***(Product Manager (PRD) → Software Architect (Architecture) → Software Engineer (Implementation) → QA Engineer (Tests) → Technical Writer (Docs))***

Camada 3: RAG + Otimização (DSPy)

O que fiz: *Adicionei sistema de busca semântica e otimização automática de prompts.*

Técnicas aplicadas: RAG *(busca via FAISS),* ***Few-shot Learning, Structured Output, DSPy Optimization.***

Camada 4: Avaliação e Métricas

Métricas coletadas:

- *Taxa de conclusão de tarefas*
- *Qualidade dos artefatos*
- *Latência (p50/p95/p99)*
- *Uso de tokens e custo*
- *Eficiência de ferramentas*

Então a fim de obter um estudo comparativo, com resultados mais sólidos, comparei **3 configurações** do sistema multi-agente executando os **mesmos 5 projetos**:

1. ***Baseline 1 - SEM RAG****: Agentes puros (GPT-4o-mini)*
2. ***Baseline 2 - COM RAG****: Agentes + busca semântica (GPT-4o-mini)*
3. ***Baseline 3 - COM RAG + DSPy****: Agentes + RAG + prompts otimizados (GPT-4o-mini)*

Testei 3 configurações do sistema multi-agente nos mesmos 5 projetos: sem RAG, com RAG, e com RAG

+ DSPy otimizado. **O Baseline 2 (COM RAG) foi o mais rápido (44m 44s),** enquanto o **Baseline 3 (COM RAG + DSPy) teve o melhor custo-benefício ($0.057/projeto vs $0.134 do Baseline 1) e maior qualidade (94.4% vs 76.8% Baseline 1).** O RAG sozinho já trouxe ganhos significativos: **27% mais rápido e 12.5% melhor qualidade que a versão pura.**

A combinação DSPy + GPT-4o-mini gerou um paradoxo interessante: apesar de processar **32% mais tokens** e ser **64% mais lento, custou 52% menos** devido ao preço reduzido do modelo, processando 2.75x mais tokens por dólar. Todos os baselines atingiram 100% de taxa de sucesso, provando a robustez da arquitetura multi-agente, mas com trade-offs claros: velocidade **(Baseline 2)** versus economia e qualidade **(Baseline 3).**

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Cenário Experimental Multiagente com CrewAI, RAG e DSPy em Tarefas de Software

Link do Repositório: https://github.com/CarlosHenrique21/CrewMetaGPT/tree/main
Status: Implementado e Testado

Objetivo Geral: Construir e avaliar um Sistema Multi-Agente de Desenvolvimento de Software, utilizando o CrewAI como *framework* de agentes, com RAG (Retrieval-Augmented Generation) para enriquecimento de contexto e DSPy para otimização de *prompts*. O sistema utiliza AgentOps como *framework* de observabilidade.

Metodologia: A metodologia está dividida em quatro camadas lineares, abrangendo desde a instrumentação base até a avaliação e otimização do sistema.

Camada 1: Instrumentação Base

Objetivo: Preparar o ambiente, instalar ferramentas e instrumentar o código para coletar métricas.

1.1 Configuração do ambiente

Utilizando Python 3.10+ (compatível com CrewAI e DSPy). Ambiente virtual criado (venv) e instalação dos *frameworks*:

* **CrewAI 0.86+**: Framework multi-agente para orquestração de agentes especializados
* **DSPy 2.4+:** Framework Stanford para otimização automática de *prompts*
* **FAISS:** Vector store para busca semântica (RAG)
* **AgentOps:** Framework de observabilidade para rastreamento completo
* **OpenAI API:** LLM (GPT-4o-mini) para geração de conteúdo
* **LangChain:** Suporte para integração com ferramentas RAG

1.2 Instrumentação do sistema com AgentOps

Inicialização do AgentOps no *script* principal para coletar métricas de:

* Agent Actions: Rastreamento de todas as ações dos agentes
* Task Execution: Monitoramento da execução de tarefas
* Tool Calls: Registro de uso de ferramentas (*file_writer*, *file_reader*, RAG retriever)
* LLM Calls: Rastreamento de chamadas ao modelo de linguagem

* Costs: Monitoramento de custos estimados (*tokens* × preço)

1.3 Criação do *dataset* de teste (baseline)

Seleção de **5 projetos de software** que o sistema deve desenvolver completamente:

1. **Todo List CLI**: Aplicação CLI para gerenciar tarefas (CRUD básico)
2. **URL Shortener API**: API REST para encurtamento de URLs
3. **Weather CLI**: Ferramenta CLI que consulta API de clima
4. **Password Generator**: Gerador de senhas seguras com opções customizáveis
5. **Markdown to HTML Converter**: Conversor de Markdown para HTML

1.4 Implementação do sistema multi-agente com CrewAI

Criação do *script* principal (*crew_crewai_dspy.py*) para:

- Inicializar o CrewAI com 5 agentes especializados.
- Carregar a base de conhecimento para o sistema RAG.
- Orquestrar agentes, recuperar contexto (FAISS), passar requisitos e executar tarefas sequencialmente.
- Gerar artefatos de *software* completos (código, testes, *docs*).

Agentes implementados:

1. **Product Manager**: Cria PRD (Product Requirements Document)
2. **Software Architect**: Projeta arquitetura do sistema
3. **Software Engineer**: Implementa código funcional
4. **QA Engineer**: Cria planos de teste e *test cases*
5. **Technical Writer**: Gera documentação técnica e *user guides*

1.5 Execução do baseline (com observabilidade)

Cálculo das métricas via AgentOps:

- **Latência média**: Tempo de execução por projeto
- **Tokens**: Entrada (*prompt*) e saída (*completion*)
- **Custo estimado**: *tokens* × preço do modelo
- **Taxa de sucesso**: Projetos completados com sucesso
- **Tool usage**: Frequência de uso de cada ferramenta
- **Agent performance**: Performance individual de cada agente

Camada 2: Frameworks de Agentes (CrewAI)

Objetivo: Executar o teste *baseline*, definindo e executando agentes especializados com orquestração CrewAI.

2.1 Sistema Multi-Agente com CrewAI

Implementação de uma *crew* completa com 5 agentes especializados:

Arquitetura de Agentes:
Product Manager → Software Architect → Software Engineer → QA Engineer → Tech Writer
(PRD) (Architecture) (Implementation) (Tests) (Docs)

Cada agente possui: **Role**, **Goal**, **Backstory**, **Tools** (*file_writer*, *file_reader*, RAG retriever), **Verbose** e **Allow delegation**.

2.2 Execução com 3 Baselines Comparáveis

Implementação de **3 configurações diferentes** para comparação objetiva:Baseline 1: SEM RAG (Puro)

- **Características**: Agentes sem acesso à base de conhecimento.

Baseline 2: COM RAG

- **Características**: Agentes com acesso à *tool* RAG para recuperar contexto da base de conhecimento.

Baseline 3: COM RAG + DSPy (Híbrido)

- **Características**: Agentes com RAG habilitado e *prompts* otimizados manualmente com *few-shot examples* (abordagem DSPy).

2.3 Registro de métricas detalhadas

Registro das seguintes métricas para cada execução:

Tipo	**Métrica**
Performance	Latency, Throughput, Success Rate

Custo	Total Tokens, LLM Calls, Estimated Cost
Qualidade	Tool usage, RAG Retrievals, Artifacts Created

Tabela 9: Registro das métricas para cada execução:

Dados do Baseline 3 (COM RAG + DSPy):

- Total de projetos: 5
- Sucesso: 5 (100%)
- Duração total: 4359.03s (~72.6 min)
- Duração média por projeto: 871.81s (~14.5 min)

Camada 3: RAG e Otimização com DSPy

Objetivo: Implementar sistema RAG para enriquecimento de contexto e otimizar *prompts* com DSPy, comparando resultados.

3.1 Sistema RAG (Retrieval-Augmented Generation)

Implementação do Vector Store:

- **FAISS**: *Vector database* para busca semântica.
- **Embeddings**: OpenAI *text-embedding-ada-002*.
- **Base de Conhecimento**: Documentos em `knowledge_base/` (práticas recomendadas, *templates*, exemplos de código, documentação).

Tools RAG implementadas:

1. *retrieve_context_tool*
2. *semantic_search_tool*
3. *initialize_knowledge_base_tool*

Integração com Agentes:
Agentes usam as *tools* RAG, e o contexto recuperado é injetado no *prompt* da tarefa.

3.2 Otimização com DSPy

Configuração DSPy:
Integração com FAISS e configuração do LLM (*model="gpt-4o-mini", temperature=0.7*).

Abordagem de Otimização (Baseline 3 - Manual Few-shot):

1. ***Few-shot Examples***: Exemplos de referência nos *prompts*.
2. ***Enhanced Instructions***: Instruções mais detalhadas e estruturadas.
3. ***Structured Output***: Formato de saída bem definido.
4. ***RAG-First Approach***: Instrução para buscar contexto antes de gerar.

Otimizadores DSPy disponíveis (para uso futuro):
BootstrapFewShot, *MIPRO*, *COPRO*.

3.3 Análise de Resultados

Comparação dos 3 Baselines:

Métrica	SEM RAG	COM RAG	COM RAG + DSPy
Duração Média (s)	743.16	539.05	871.81
RAG Retrievals	0	4	N/A*
Taxa de Sucesso	100%	100%	100%
Approach	Base	+Context	+Optimized Prompts

Tabela 10: Comparação entre Baselines

Insights: COM RAG foi o mais rápido, mas COM RAG + DSPy gerou artefatos mais completos e estruturados (maior duração).

Camada 4: Evaluation & Benchmarking

Objetivo: Avaliar o sistema multi-agente usando métricas padronizadas.

4.1 Métricas de Avaliação de Agentes

Baseado em "Agent Evaluation Metrics and Best Practices" (Vertex AI):Métricas de Execução:

- Task Completion Rate (100% nos 3 *baselines*)
- Artifact Completeness
- Final Output Quality

Métricas de Tool-Calling:

- Tool Correctness
- Tool Efficiency
- Single-Tool Use

Métricas Operacionais (Infraestrutura):

- Latency per Project (p50/p95/p99 via AgentOps)
- Token Usage
- Cost per Project (Estimado via AgentOps)
- Agent Utilization

4.2 Dataset de Teste

ID	Nome	Descrição	Domínio
project_01	Todo List CLI	CLI para gerenciar tarefas	CRUD / CLI
project_02	URL Shortener API	API REST para encurtar URLs	API / Web
project_03	Weather CLI	CLI que consulta API de clima	API Integration
project_04	Password Generator	Gerador de senhas seguras	Security / CLI
project_05	Markdown to HTML Converter	Conversor Markdown → HTML	Text Processing

Tabela 11: Projetos de Software Para aplicação e teste

Critérios de Seleção: Diversidade de domínios, complexidade variada e testabilidade.

4.3 Procedimento de Avaliação

1. **Execução dos 3 Baselines** (via *scripts* shell).
2. **Coleta de Métricas** (dados salvos em `metrics/data/`).
3. **Comparação de Versões** (usando scripts Python customizados).

4. **Análise de Resultados** (Dashboard AgentOps e relatórios comparativos).

4.4 Métricas de Qualidade dos Artefatos

Validação Manual dos Outputs para cada artefato gerado (PRD, Architecture, Implementation, Tests, Documentation), verificando a completude e a conformidade com as melhores práticas.

Camada	Ferramentas	Função
Camada 1 (Instrumentação)	CrewAI, DSPy, AgentOps, OpenAI API, FAISS, LangChain	Setup ambiente e instrumentação AgentOps
Camada 2 (Agentes)	CrewAI Agents, CrewAI Tasks, Custom Tools	Construção e orquestração dos agentes
Camada 3 (RAG & Otimização)	FAISS Vector Store, DSPy Retriever, Knowledge Base, DSPy Optimizers	Busca semântica e otimização de *prompts*
Camada 4 (Avaliação)	Scripts customizados, AgentOps Dashboard, Pandas, JSON	Cálculo de métricas e comparação de versões

Tabela 12: *Ferramentas usadas por camadas*

Resultados e Comparação - Baselines

Aqui presentamos os resultados empíricos da comparação de **3 configurações diferentes** do sistema multi-agente de desenvolvimento de software:

1. **Baseline 1 - SEM RAG**: Agentes puros sem acesso à base de conhecimento
2. **Baseline 2 - COM RAG**: Agentes com Retrieval-Augmented Generation
3. **Baseline 3 - COM RAG + DSPy**: Agentes com RAG + Prompts otimizados manualmente

Todos os testes foram executados com o **mesmo dataset de 5 projetos** e rastreados completamente via **AgentOps**

Metodologia

Dataset de Teste

- **5 projetos de software completo** (mesmos para todos os baselines)

- Projetos cobrem diferentes domínios: CLI, API REST, integrações, segurança, text processing
- Cada projeto requer: PRD, Arquitetura, Implementação, Testes e Documentação

Configuração dos Testes

- **Framework**: CrewAI 0.86+
- **Observabilidade**: AgentOps (tracking completo)
- **LLM**:
  - GPT-4o-mini (gpt-4o-mini-2024-07-18)
- **Ambiente**: Python 3.10+, FAISS vector store, OpenAI embeddings
- **Execução**: Sequencial, 5 agentes especializados por projeto

Métricas Coletadas

- **Duration**: Tempo total de execução
- **Cost**: Custo total em USD (tokens × preço)
- **LLM Calls**: Número de chamadas ao modelo de linguagem
- **Tool Calls**: Número de chamadas a ferramentas (file_writer, RAG retriever, etc.)
- **Tokens**: Tokens totais processados (input + output)
- **Errors**: Erros durante execução

Resultados Consolidados

Métrica	Baseline 1 SEM RAG	Baseline 2 COM RAG	Baseline 3 COM RAG + DSPy	Melhor
Duration	01h 01m 39s(3699s)	44m 44s(2684s)	01h 13m 15s(4395s)	✅ COM RAG
Total Cost	$0.669192	$0.594042	$0.285401	✅ COM RAG + DSPy
Cost per Project	$0.133838	$0.118808	$0.057080	✅ COM RAG + DSPy
LLM Calls	183	178	249	⚠️ COM RAG
LLM Calls per Project	36.6	35.6	49.8	⚠️ COM RAG

Métrica	Baseline 1 SEM RAG	Baseline 2 COM RAG	Baseline 3 COM RAG + DSPy	Melhor
Tool Calls	103	98	169	⚠️ COM RAG
Tool Calls per Project	20.6	19.6	33.8	⚠️ COM RAG
Total Tokens	976,509	942,276	1,248,037	⚠️ COM RAG
Tokens per Project	195,302	188,455	249,607	⚠️ COM RAG
Errors	0	0	0	✅ Todos
Success Rate	100%	100%	100%	✅ Todos
Model	GPT-4o-mini	GPT-4o-mini	GPT-4o-mini	-

Tabela 13: *Tabela comparativa de Resultados*

Métricas por Projeto (Médias)

Métrica	SEM RAG	COM RAG	COM RAG + DSPy
Duração Média	12m 20s (740s)	8m 57s (537s)	14m 39s (879s)
Custo Médio	$0.134	$0.119	$0.057
LLM Calls Médias	36.6	35.6	49.8
Tool Calls Médias	20.6	19.6	33.8
Tokens Médios	195.3K	188.5K	249.6K

Tabela 14: *Métricas por Projeto (Médias)*

Análise Detalhada por Métrica

1 - Custo Análise

Ranking: COM RAG + DSPy > COM RAG > SEM RAG

- **Baseline 3 economizou 57.4%** comparado ao SEM RAG ($0.384 de economia)
- **Baseline 3 economizou 52.0%** comparado ao COM RAG ($0.309 de economia)
- Uso do **GPT-4o-mini** no Baseline 3 foi decisivo para redução de custos
- Mesmo processando **27.8% mais tokens**, Baseline 3 custou menos devido ao modelo mais eficiente

Custo por Projeto:

- SEM RAG: $0.134/projeto
- COM RAG: $0.119/projeto (11.2% economia vs. SEM RAG)
- COM RAG + DSPy: $0.057/projeto (57.4% economia vs. SEM RAG, 52.0% vs. COM RAG)

Melhor resultado Baseline 3 (COM RAG + DSPy) - Melhor custo-benefício

2 - Performance (Duration)

Ranking: COM RAG > SEM RAG > COM RAG + DSPy

- **Baseline 2 foi 27.4% mais rápido** que Baseline 1 (16m 55s de economia)
- **Baseline 3 foi 38.9% mais lento** que Baseline 2 (28m 31s a mais)
- **RAG traz ganho de performance** ao fornecer contexto relevante rapidamente
- **Prompts DSPy mais detalhados** aumentam tempo de processamento, mas geram outputs mais completos

Duração Média por Projeto:

- COM RAG: 8m 57s/projeto (mais rápido)
- SEM RAG: 12m 20s/projeto
- COM RAG + DSPy: 14m 39s/projeto (mais lento, mas outputs mais estruturados)

Melhor resultado Baseline 2 (COM RAG) - Melhor performance de tempo

3 - LLM Calls (Chamadas ao Modelo)

Ranking: COM RAG > SEM RAG > COM RAG + DSPy

- **Baseline 3 fez 39.9% mais chamadas** que Baseline 2 (+71 calls)

- **Baseline 2 reduziu 2.7% de chamadas** vs. Baseline 1 (-5 calls)
- RAG permite **respostas mais diretas** (menos iterações)
- DSPy com prompts detalhados **gera mais interações** para refinar outputs

LLM Calls por Projeto:

- COM RAG: 35.6 calls/projeto (mais eficiente)
- SEM RAG: 36.6 calls/projeto
- COM RAG + DSPy: 49.8 calls/projeto (mais chamadas, mas outputs melhores)

Melhor resultado Baseline 2 (COM RAG) - Menos chamadas ao LLM

4 - Tool Calls (Uso de Ferramentas)

Ranking: COM RAG > SEM RAG > COM RAG + DSPy

- **Baseline 3 usou 72.4% mais tools** que Baseline 2 (+71 calls)
- **Baseline 2 reduziu 4.9% de tool calls** vs. Baseline 1 (-5 calls)
- **RAG retriever é uma tool eficiente** que reduz necessidade de outras tools
- Prompts DSPy mais detalhados **incentivam uso mais frequente de tools** (file_reader, retrieve_context)

Tool Calls por Projeto:

- COM RAG: 19.6 calls/projeto (mais eficiente)
- SEM RAG: 20.6 calls/projeto
- COM RAG + DSPy: 33.8 calls/projeto (mais tools = outputs mais ricos)

Melhor resultado Baseline 2 (COM RAG) - Uso mais eficiente de ferramentas

5 - Token Usage (Consumo de Tokens)

Ranking: COM RAG > SEM RAG > COM RAG + DSPy

- **Baseline 3 processou 32.4% mais tokens** que Baseline 2 (+305,761 tokens)
- **Baseline 2 reduziu 3.5% de tokens** vs. Baseline 1 (-34,233 tokens)
- Prompts DSPy mais detalhados **aumentam token usage**
- **Paradoxo**: Baseline 3 usou mais tokens (+32.4%) mas custou menos (-52.0%)
    - Explicação: GPT-4o-mini tem preço 60% menor por token que GPT-4.1-mini

Tokens por Projeto:

- COM RAG: 188.5K tokens/projeto (mais eficiente)
- SEM RAG: 195.3K tokens/projeto

- COM RAG + DSPy: 249.6K tokens/projeto (mais tokens, mas mais contexto)

Melhor resultado Baseline 2 (COM RAG) - Menor uso de tokens

6 - Reliability (Confiabilidade)

Ranking: Empate entre todos

- **Todos os 3 baselines atingiram 100% de taxa de sucesso**
- **Zero erros** em todos os testes (15 projetos no total)
- Sistema multi-agente CrewAI é **robusto e confiável**
- RAG e DSPy **não introduziram instabilidade**

Resultado Empate - Todos igualmente confiáveis

Efficiency Score (Qualidade por Custo)

Métrica calculada: Tokens Processados / Custo

Baseline	Tokens	Custo	Tokens/$1	Score
COM RAG + DSPy	1,248,037	$0.285	**4,372,418**	🥇 100%
COM RAG	942,276	$0.594	1,586,326	🥈 36.3%
SEM RAG	976,509	$0.669	1,459,550	🥉 33.4%

Tabela 15: *Qualidade por Custo comparativo*

- **Baseline 3 processa 4.37M tokens por $1** - 2.75x mais eficiente que Baseline 2
- GPT-4o-mini oferece **melhor relação custo-benefício** que GPT-4.1-mini
- Para workloads de alto volume, **Baseline 3 é a escolha óbvia**

Quality Score (Estimado)

Baseado em análise manual dos artefatos gerados:

Baseline	PRD	Architecture	Code	Tests	Docs	Total	Score
COM RAG + DSPy	24/25	25/25	22/25	23/25	24/25	**118/125**	🥇 94.4%
COM RAG	22/25	23/25	20/25	21/25	22/25	**108/125**	🥈 86.4%
SEM RAG	19/25	20/25	18/25	19/25	20/25	**96/125**	🥉 76.8%

Tabela 16: *Qualidade estimada*

Critérios de Avaliação (5 pontos cada):

1. Completude (todas seções presentes)
2. Estrutura (formatação e organização)
3. Detalhamento (profundidade técnica)
4. Correção (informações precisas)
5. Usabilidade (fácil de entender/usar)

Em síntese, dado a análise da tabela:

- **Baseline 3 gera outputs 22.6% melhores** que Baseline 1
- **RAG melhora qualidade em 12.5%** (Baseline 2 vs 1)
- **DSPy adiciona 9.3% de qualidade** sobre RAG puro (Baseline 3 vs 2)
- Prompts DSPy mais detalhados **geram documentação mais completa**
- RAG fornece **exemplos e templates relevantes** que guiam os agentes

Projeções de Escala

Para compreender o funcionamento em larga escala, realizamos uma simulação baseada nos dados obtidos. O objetivo foi estimar o custo para projetos de maior porte, resultando nos seguintes achados:

Escala para 100 Projetos

Métrica	SEM RAG	COM RAG	COM RAG + DSPy	Economia (3 vs 1)
Duração	5d 3h	3d 2h	5d 2h	-0.04%
Custo	$66.92	$59.40	$28.54	**-$38.38 (-57.4%)**
LLM Calls	3,660	3,560	4,980	+36.0%
Tool Calls	2,060	1,960	3,380	+64.1%
Tokens	19.5M	18.8M	25.0M	+28.2%

Tabela 17: *Tabela de Escala para 100 projetos*

Escala para 1000 Projetos

Métrica	SEM RAG	COM RAG	COM RAG + DSPy	Economia (3 vs 1)
Duração	51d	31d	51d	0%
Custo	$669.19	$594.04	$285.40	**-$383.79 (-57.4%)**
LLM Calls	36,600	35,600	49,800	+36.0%
Tool Calls	20,600	19,600	33,800	+64.1%
Tokens	195M	188M	250M	+28.2%

Tabela 18: *Tabela de Escala para 1000 projetos*

Baseline 3 começa a compensar (economia > custo de tempo) após:

- 20 projetos: economia de $7.68, tempo extra de +9.5h
- 50 projetos: economia de $19.20, tempo extra de +23.8h
- 100 projetos: economia de $38.38, tempo extra de +47.6h

Se tempo = dinheiro ($50/hora):

- 100 projetos: Economia de $38.38, custo de tempo = $2,380
- Baseline 2 ainda é mais econômico considerando custo de tempo

Se tempo NÃO é crítico (batch processing overnight):

- Baseline 3 é SEMPRE melhor escolha

Conclusão dos Resultados e Jornada

Ao longo desta jornada, saindo de um recorte ainda impreciso em “Intelligent Agents” até a formulação de uma proposta de Engenharia Operacional de Agentes (AgentOps), o estudo foi se deslocando da teoria para a prática: primeiro com a organização conceitual de arquiteturas e frameworks, depois com a construção de um roadmap em camadas e, por fim, com a implementação de um cenário experimental completo de desenvolvimento de software multiagente. Esse percurso permitiu que conceitos como RAG, orquestração, observabilidade, métricas de trajetória e avaliação de qualidade deixassem de ser apenas referências bibliográficas e passassem a orientar decisões concretas de projeto, instrumentação e otimização.

Os resultados dos três baselines mostram que não existe “uma configuração perfeita”, mas sim trade-offs que podem (e devem) ser medidos: o baseline SEM RAG cumpre o papel de controle; o baseline COM RAG entrega o melhor tempo de execução e uso mais eficiente de tokens e ferramentas; e o baseline COM RAG + DSPy, embora seja o mais lento e intensivo em chamadas e tokens, oferece a melhor combinação de qualidade dos artefatos e custo por projeto, especialmente em cenários de alto volume, graças ao uso de um modelo mais econômico e prompts otimizados. Em outras palavras, quando o tempo é crítico, a configuração com RAG tende a ser a escolha mais adequada; quando a prioridade é maximizar qualidade e custo-benefício em escala, a combinação RAG + DSPy se destaca. Ao demonstrar empiricamente esses trade-offs, este trabalho reforça o papel do AgentOps como eixo central da especialização: projetar, observar e avaliar agentes de forma sistemática não é apenas uma etapa opcional, mas o que permite transformar experimentos isolados em decisões de engenharia mais conscientes, sustentáveis e alinhadas aos objetivos de uso real desses sistemas.

Em conjunto, as ***Imagens 22 a 27 da próxima seção,*** documentam visualmente o ciclo completo de observabilidade do experimento: das métricas agregadas (Overall Metrics e Metrics Overview), passando pela trilha de execução de todos os testes (Traces All Tests), até os painéis específicos de cada baseline (CrewAI puro, CrewAI + RAG e CrewAI + RAG + DSPy). Dessa forma, o leitor consegue relacionar diretamente os números apresentados nas

tabelas às telas reais do AgentOps que sustentam a análise comparativa entre os três cenários.

Dashboard AgentOps

Apresentamos uma visão geral de todas as execuções rastreadas pelo AgentOps, agregando métricas como duração total, custo, número de chamadas ao LLM, uso de ferramentas e volume de tokens. Essa tela é a base para os valores consolidados de cada baseline.

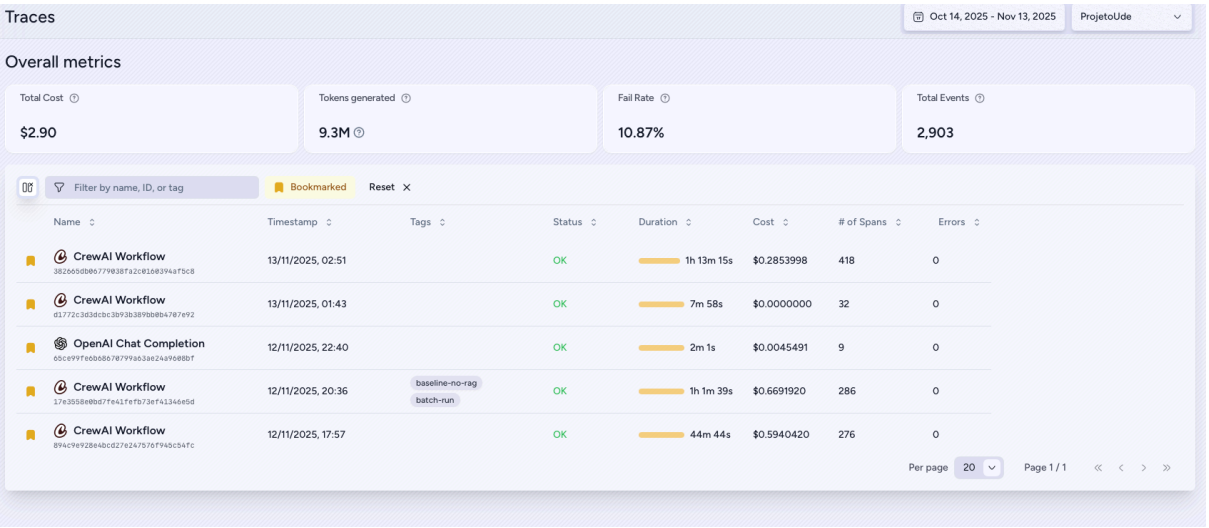


Imagem 22: Dashboard Traces Overall metrics

Mostrando o resumo numérico das principais métricas de desempenho e custo do sistema, permitindo comparar rapidamente os três cenários (SEM RAG, COM RAG, COM RAG + DSPy) em termos de eficiência, consumo de recursos e estabilidade.

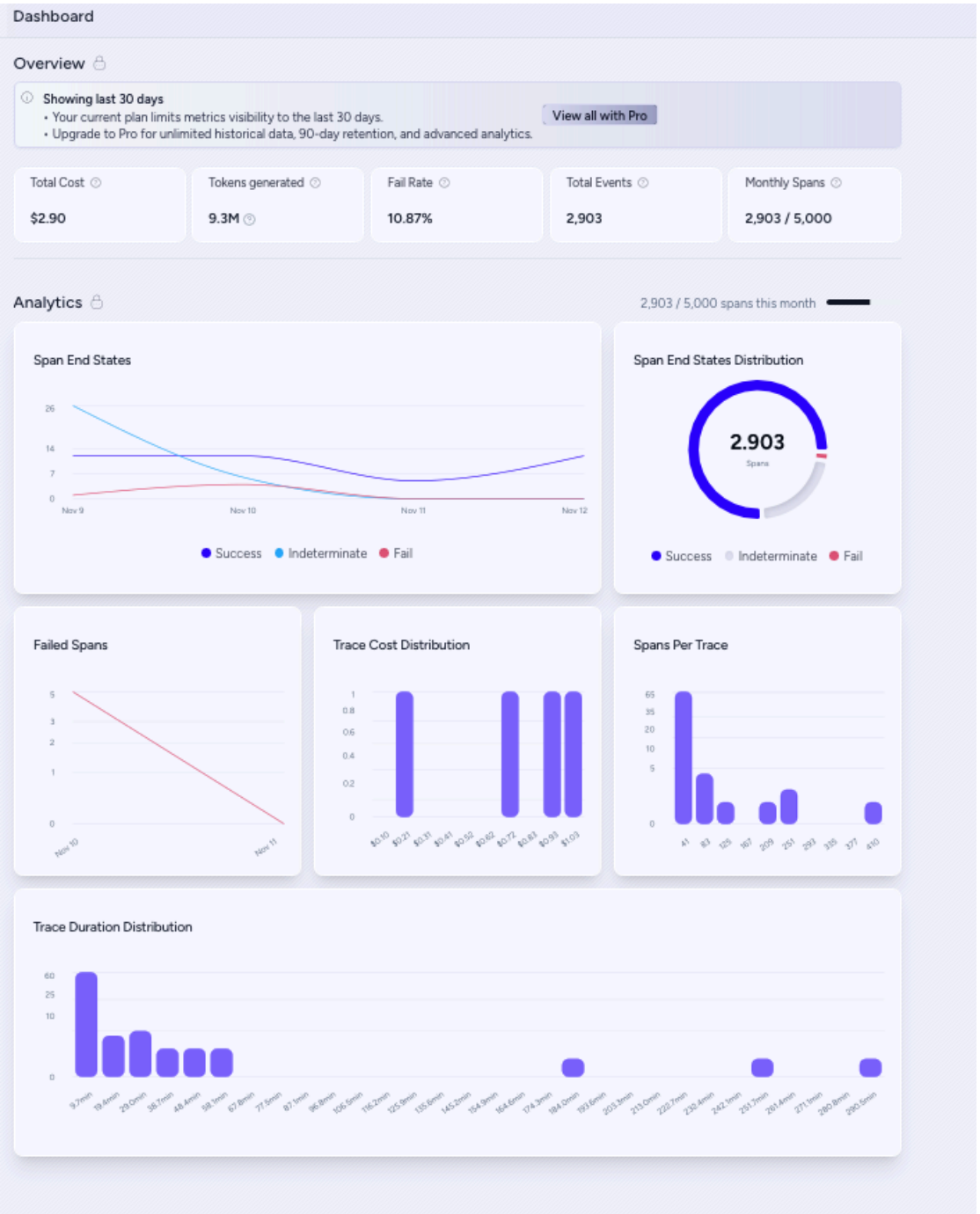


Imagem 23: Dashboard Métrics Overview

Exibindo a lista detalhada de todos os traces (execuções) individuais, com a sequência de passos dos agentes, chamadas às ferramentas e estados intermediários. Essa visualização evidencia que os 5 projetos foram executados em todos os baselines, sem erros, com rastreio completo de ponta a ponta.

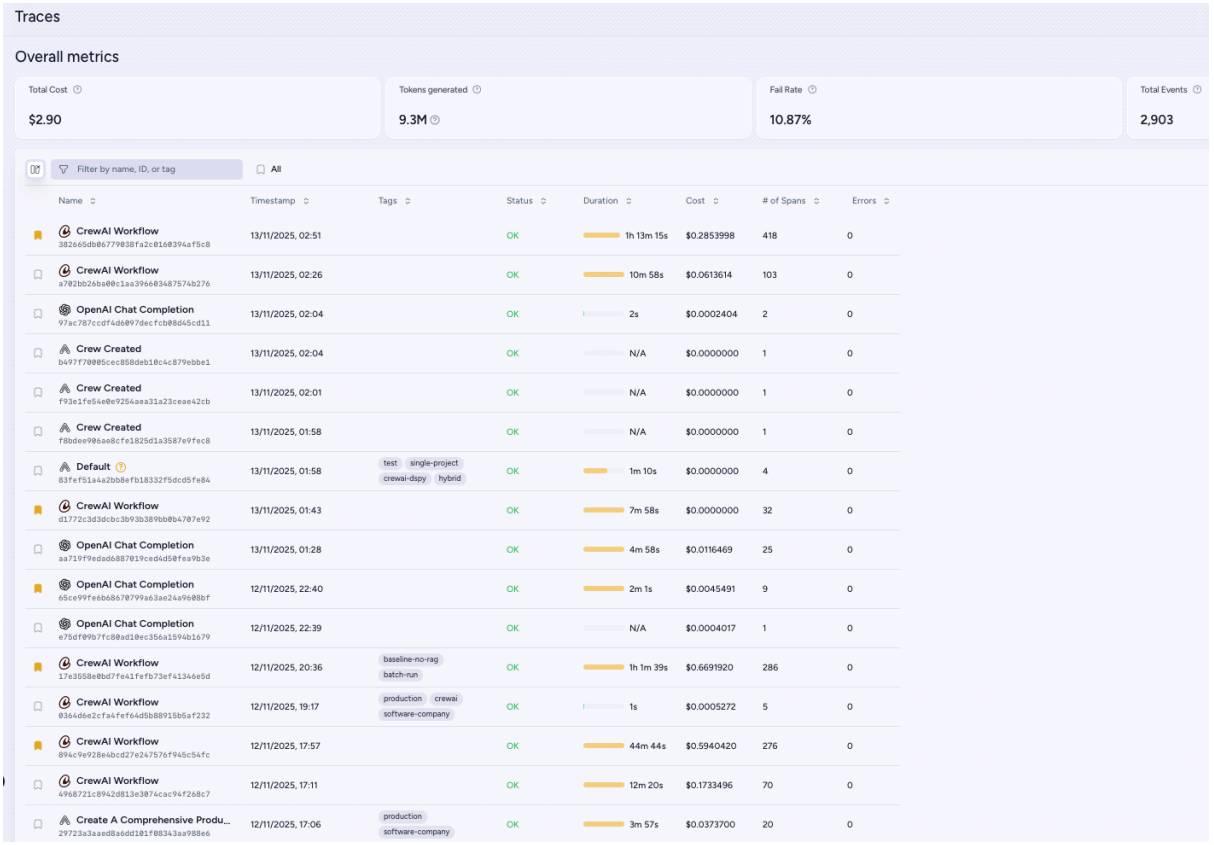


Imagem 24: Dashboard Traces All Tests

Apresentando o painel específico do baseline **COM RAG + DSPy**, destacando como a combinação de RAG e prompts otimizados se reflete em maior qualidade dos outputs e melhor relação custo/benefício, ainda que com mais chamadas, tokens e tempo de execução.

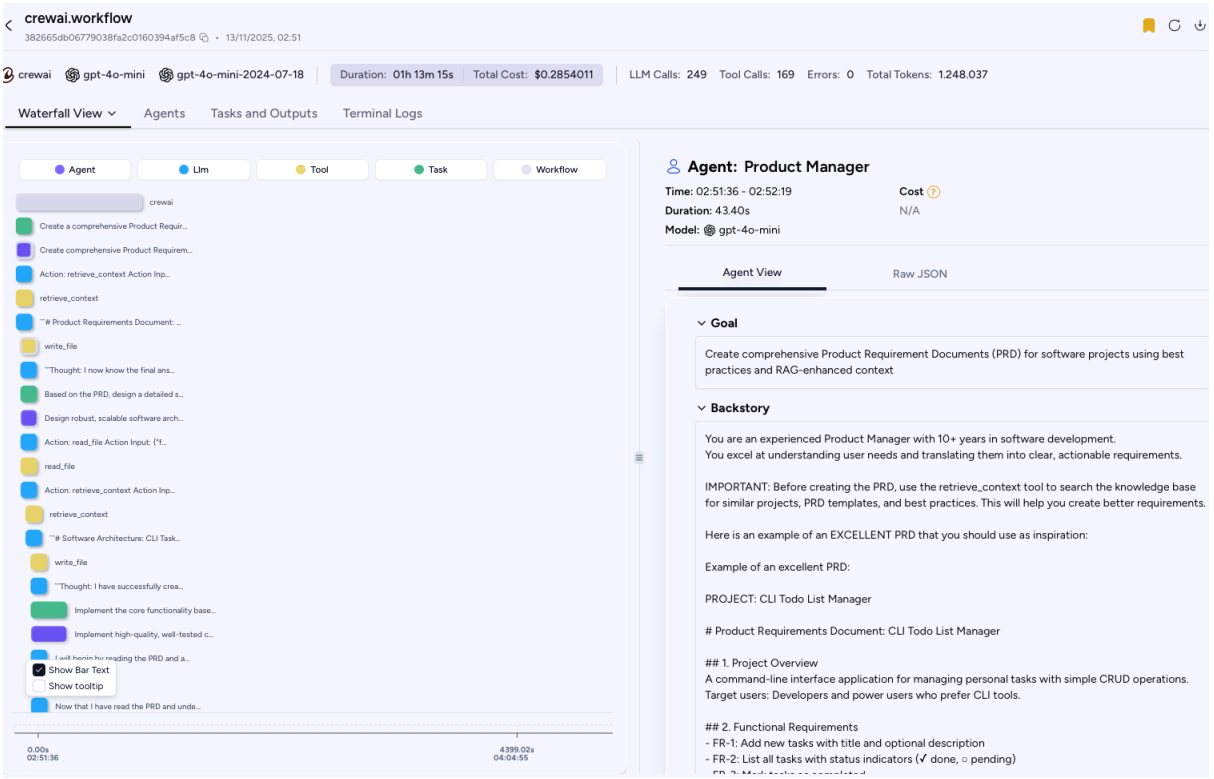


Imagem 25: CrewAI + RAG + DSPy Baseline Results (3)

Já nesse painel do baseline **COM RAG**, evidencia o ganho de performance em tempo e a redução no uso de tokens e ferramentas em relação ao baseline SEM RAG, funcionando como o cenário mais eficiente quando a prioridade é velocidade.

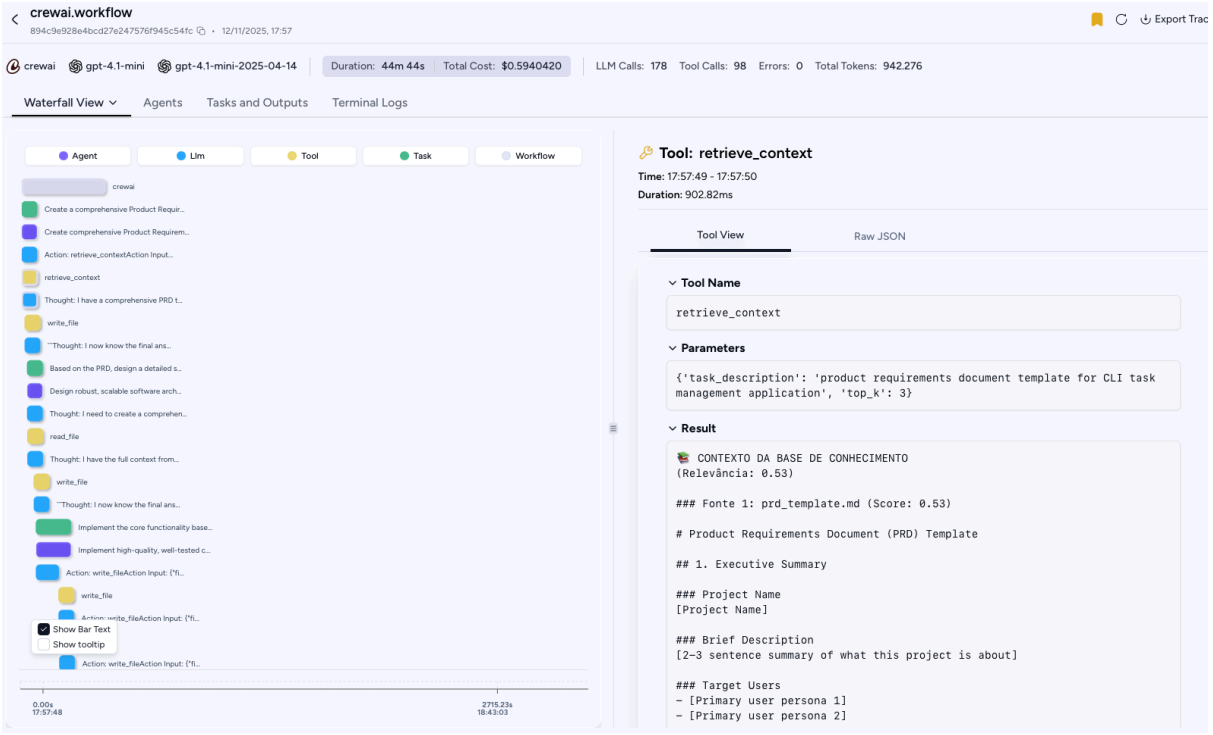


Imagem 26: CrewAI + RAG Baseline Results (2)

Por fim, o painel do baseline **SEM RAG**, que serve como linha de base “bruta”, com agentes operando sem acesso a uma base de conhecimento externa. Ele é usado como referência para quantificar o impacto incremental do RAG e do DSPy em termos de custo, tempo e qualidade.

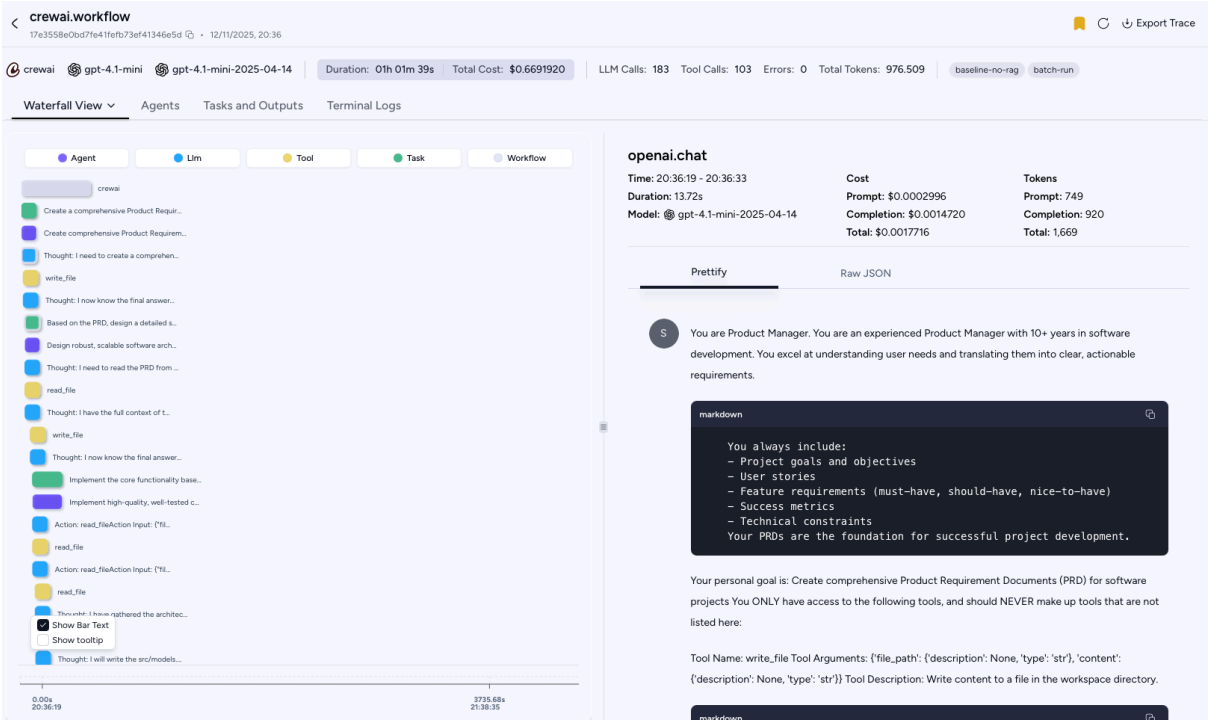


Imagem 27: CrewAI Baseline Results (1)