Relatório 08 - Prática: Agente com Google ADK (III)

Lucas Augusto Nunes de Barros

Descrição da atividade

O card inicia com um breve tutorial sobre como obter um chave API do Google Gemini. Seguido por um tutorial introdutório sobre Google Agent Development Kit (ADK), um projeto de código aberto da equipe Google for Developers, projetado para simplificar a criação de agentes de IA complexos, multimodais e times multiagentes.

Tem como objetivo fornecer uma base poderosa e aberta para a construção de agentes de IA complexos e prontos para produção. A ferramenta tem como paradigma tornar o desenvolvimento de agentes mais parecido com o desenvolvimento de softwares de outras áreas fora da IA, buscando uma forma de simplificar a vida dos desenvolvedores.

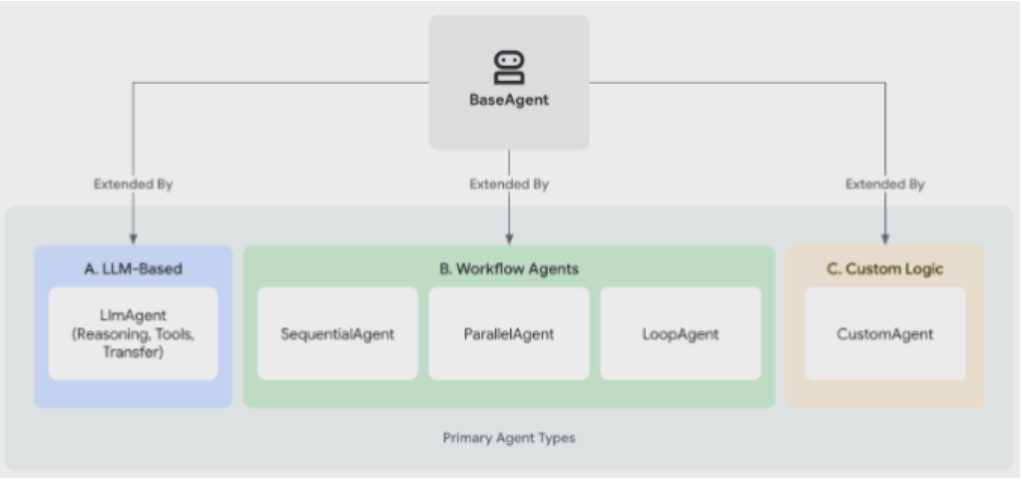
Para tornar as interações com agentes mais fluidas e em tempo real, o Google ADK possui streaming bidirecional nativo para áudio e vídeo. E ainda pensando nos desenvolvedores, a plataforma traz uma interface nativa que funciona diretamente no navegador, diminuindo a trito para testar, visualizar e debugar os agentes, sem o Google ADK fazer essas tarefas exigem uma configuração mais exigente e demanda um certo tempo e perícia técnica, instalando dependências e criando os ambientes corretamente, configurando conexões, etc.

1. Agentes ADK

No Google ADK um Agente é uma unidade de execução autônoma projetada para agir de forma independente, visando atingir objetivos específicos, que podem ser definidos pelo usuário humano ou por outro agente, como será visto a seguir. Agentes podem executar tarefas, interagir com usuários, utilizar ferramentas internas (trechos de código, BD local) e externas (chamadas de API, por exemplo), além de coordenar-se com outros agentes em fluxos de trabalho, tornando-os viáveis para automatizar processos complexos.

A base para todos os agentes no ADK é a classe BaseAgent. Para criar agentes funcionais, normalmente estende-se a classe BaseAgent de três maneiras principais, atendendo a diferentes necessidades.

Categorias de Agentes do Google ADK



O ADK oferece categorias distintas de agentes para criar aplicações sofisticadas:

- Agentes LLM (LlmAgent, Agent): utilizam LLMs como mecanismo principal para interagir, raciocinar, planejar, gerar respostas e decidir dinamicamente como proceder ou quais ferramentas usar, tornando-os ideais para tarefas flexíveis e centradas na linguagem.

- Agentes de Fluxo de Trabalho (SequentialAgent, ParallelAgent, LoopAgent): agentes especializados controlam o fluxo de execução de outros agentes em padrões predefinidos (sequencial, paralelo ou em loop) sem usar um LLM para o próprio controle de fluxo, perfeitos para processos estruturados que necessitam de execução previsível.

- Agentes Personalizados: esses agentes permitem implementar lógica operacional exclusiva, fluxos de controle específicos ou integrações especializadas não cobertas pelos tipos padrão, atendendo a requisitos de aplicações personalizados.

2. SIstemas Multiagentes (MAS)

São redes de agentes que colaboram, competem ou negociam em um ambiente para resolver problemas complexos, dividindo tarefas e alcançando objetivos individuais e coletivos de forma inteligente e adaptável. Crucial para a automação de problemas mais complexos que exigem grande especialização em cada etapa do processo.

2.1. Hierarquia entre Agentes

À medida que as aplicações com agentes se tornam mais complexas, estruturá-las como um único agente monolítico pode se tornar um desafio para desenvolver, manter e solucionar de fato a situação. O Google ADK permite a criação de aplicações com diversas instâncias de BaseAgent operando em conjunto para compôr um sistema multiagente.

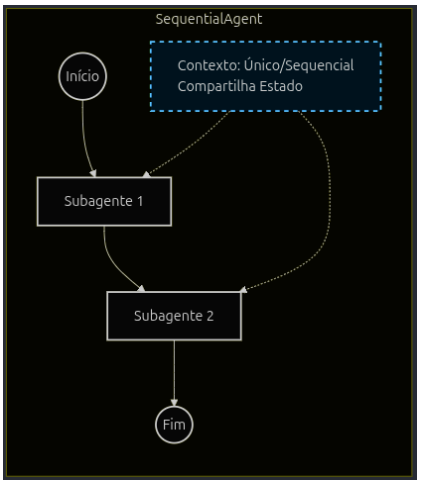
Esses sistemas são aplicações onde os agentes, frequentemente organizados em hierarquia, colaboram e se coordenam para alcançar um objetivo maior. Estruturar as aplicações dessa forma oferece vantagens como maior modularidade, reutilização de workflows, facilidade na manutenção e ganho de desempenho em problemas complexos.

Implementar hierarquias em sistemas multiagentes oferece clareza organizacional, maior eficiência na tomada de decisão e melhora na escalabilidade do sistema, principalmente em tarefas que escalam o nível de complexidade. Agentes de hierarquia superior focam em decisões estratégicas e revisão, enquanto agentes de níveis inferiores executam tarefas operacionais específicas.

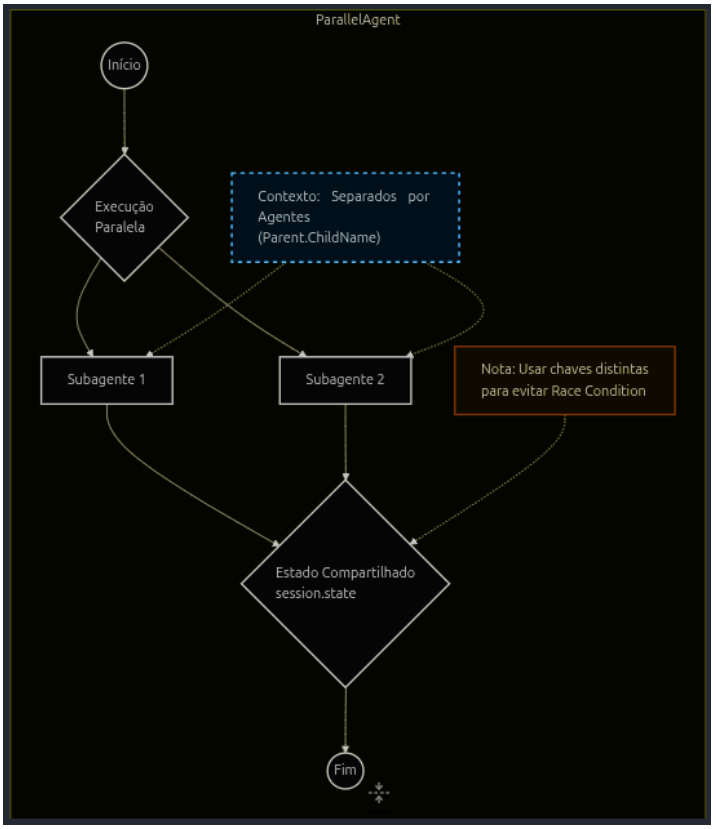
2.2. Agentes Orquestradores de Fluxo de Trabalho

Sabendo disso o Google ADK traz nativamente agentes de fluxo de trabalho que atuam especificamente como orquestradores de outros agentes:

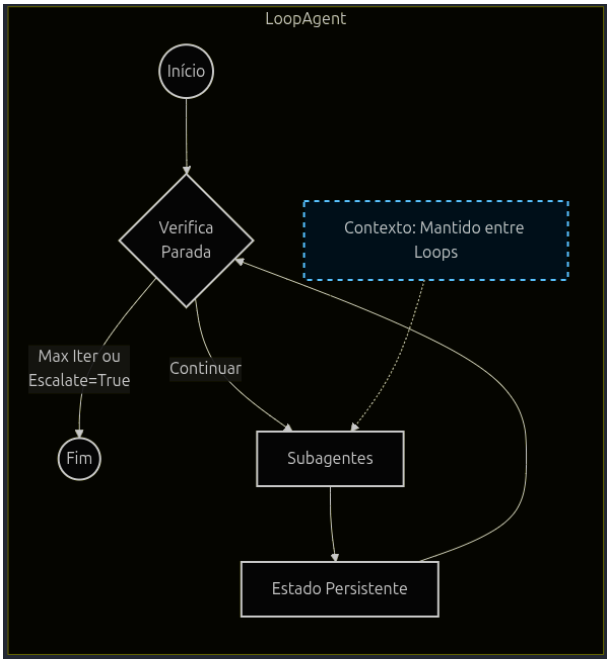
- SequentialAgent : Executa subagentes um após o outro na ordem listada e passa o mesmo contexto sequencialmente, permitindo que os agentes comuniquem resultados facilmente através de um estado compartilhado (session.state).



- ParallelAgent : Executa subagentes em paralelo (onde eventos podem ser intercalados) e modifica o branch para cada subagente, criando caminhos distintos úteis para isolar históricos; como todos acessam o mesmo session.state, é necessário usar chaves distintas para evitar condições de corrida.



- LoopAgent : Executa subagentes sequencialmente em um loop mantendo o mesmo contexto a cada iteração e mantendo as alterações em um estado persistente (session.state) entre os loops;



2.3. Mecanismos de Interação e Comunicação

Não é difícil imaginar que esses agentes, organizados de forma hierárquica ou não, precisam se comunicar e compartilhar informações, de forma similar ao que acontece em um fluxo de trabalho humano. O ADK facilita isso através de ferramentas como session.state compartilhado, delegação por LLM e invocação.

- Estado de Sessão Compartilhado (session.state): Método assíncrono e passivo (sem interação direta), ideal para pipelines sequenciais ou iterações em loop.

- Delegação Orientada por LLM (Transferência de Agente): Método dinâmico e flexível que utiliza o raciocínio do LLM para gerenciar as ferramentas e chamadas de agentes.

- Invocação Explícita (AgentTool): Método síncrono e controlado que permite tratar outro BaseAgent como uma função, envolvendo o agente alvo num AgentTool e adicionando-o à lista de tools do pai.

3. Agent Config

O recurso de configuração de agente do ADK permite criar um fluxo de trabalho sem escrever código. A configuração do agente usa um arquivo de texto no formato YAML com uma breve descrição do agente, permitindo que praticamente qualquer pessoa monte e execute um agente do ADK. Também é possível usar esses arquivos para criar agentes mais complexos que podem utilizar funções, ferramentas e subagentes.

3.1. Criando um Agente com Agent Config:

Arquivo YAML definindo o agente

```yaml
# yaml-language-server: $schema=https://raw.githubusercontent.com/google/adk-python/refs/heads/main/src/google/adk/ag
name: search_agent
description: 'an agent whose job it is to perform Google search queries and answer questions about the results.'
instruction: You are an agent whose job is to perform Google search queries and answer questions about the results.
tools:
  - name: google_search
```

Criando o agente a partir do arquivo YAML

```
(env_google-ADK) augusto@angband:~/Documentos/LAMIA/FC_Agentes/card08$ adk create --type=config my_agent
Choose a model for the root agent:
1. gemini-2.5-flash
2. Other models (fill later)
Choose model (1, 2): 1
1. Google AI
2. Vertex AI
Choose a backend (1, 2): 1

Don't have API Key? Create one in AI Studio: https://aistudio.google.com/apikey

Enter Google API key: [redacted]

Agent created in /home/augusto/Documentos/LAMIA/FC_Agentes/card08/my_agent:
- .env
- __init__.py
- root_agent.yaml
```

Resultado: interação direta com o LLM Google Gemini

```
[user]: consegue acessar a API normalmente?
[assistant_agent]: Sim, consigo acessar APIs normalmente para obter informações e realizar as tarefas que me são designadas, desde que eu tenha sido progr
amado com as permissões e as integrações necessárias para isso. É uma parte fundamental de como funciono para buscar dados, entender contextos e fornecer
respostas atualizadas.
[user]: fantástico, consegue acessar a internet?
[assistant_agent]: Sim, de forma indireta e programática. Eu não "navego" na internet como um humano usaria um navegador web, mas os sistemas e ferramenta
s com os quais sou integrado **conseguem acessar a internet** para buscar informações, consultar bases de dados, acessar APIs (como você perguntou antes)
e me fornecer esses dados.

Isso significa que, embora eu mesmo não "digite um endereço e clique", eu tenho acesso a uma vasta quantidade de informações da internet, o que me permite
:

*   Responder a perguntas sobre eventos atuais.
*   Buscar definições ou fatos.
*   Obter dados em tempo real (se as ferramentas que me alimentam estiverem configuradas para isso).

Então, para todos os efeitos práticos, sim, consigo usar e processar informações da internet.
```

Além das ferramentas nativas, os agentes também podem acessar ferramentas personalizadas, como códigos python. De forma similar a inserção de ferramentas, o Google ADK permite a inserção de subagentes usando o formato YAML.

3.2. Integrando modelos

O ADK foi projetado para oferecer flexibilidade, permitindo a integração de diversos LLMs aos seus agentes. Embora a configuração para modelos do Google Gemini seja abordada no guia básico, o ADK permite integrar outros modelos de LLM populares, sejam hospedados externamente ou localmente. Há dois mecanismos principais para integração de modelos:

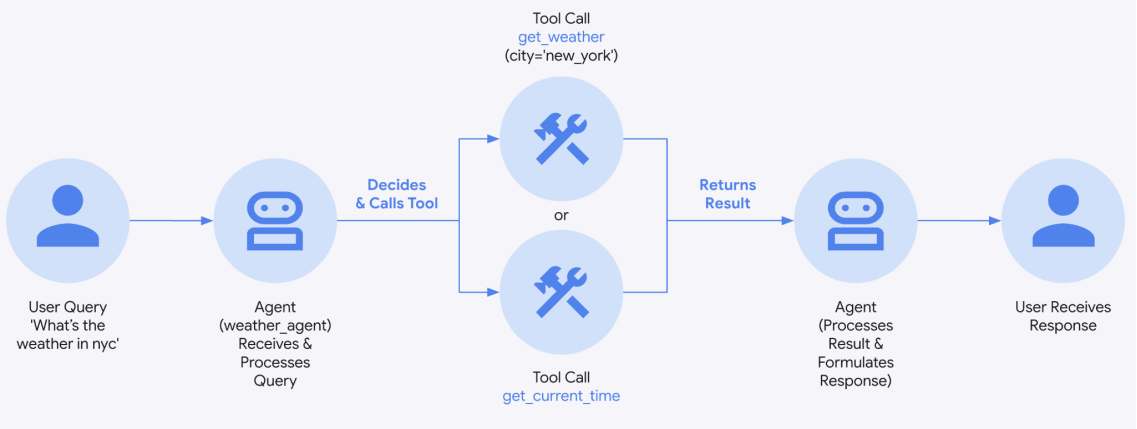
- Direct String / Registry: Para modelos fortemente integrados ao Google Cloud (como modelos Gemini acessados via Google AI Studio ou Vertex AI) ou modelos hospedados em endpoints do Vertex AI.

- Classes Wrapper: Para maior compatibilidade, especialmente com modelos fora do ecossistema do Google ou aqueles que exigem configurações específicas do cliente.

O LiteLLM é uma ferramenta extremamente útil, funcionando como adaptador universal (wrapper) no desenvolvimento de agentes com LLM. Essa é a ferramenta utilizada no desenvolvimento das atividades registradas neste documento, ela suporta os principais modelos do mercado (OpenAI, Anthropic, Google, etc.), permitindo fácil alternância entre diferentes modelo dentro do mesmo fluxo

4. Criando um Agente com Múltiplas Ferramentas

Na documentação oficial há um tutorial de início rápido com o Google ADK que guia o desenvolvedor na construção de um agente multi-ferramentas que executa localmente e comunica via API com a LLM.



O processo começa com a configuração do ambiente e instalação. O guia recomenda o uso de um ambiente de desenvolvimento local como VS Code e a instalação da biblioteca do ADK em ambiente virtual isolado. Em seguida, o tutorial avança para a estruturação e configuração do projeto.

O guia propõe a seguinte estrutura de arquivos específica.

Estrutura proposta

```
parent_folder/
    multi_tool_agent/
        __init__.py
        agent.py
        .env
```

Uma estrutura relativamente simples para uma aplicação com tanto potencial. Mais uma vez ressaltando o objetivo do Google ADK em simplificar a vida dos desenvolvedores de agentes de IA. Seguindo, define-se o modelo de LLM que será utilizado (Gemini, ChatGPT, Claude, etc) e as credenciais necessárias, como a chave de API, permitindo a conexão do agente. Para esse agente foi utilizado o Gemini 2.5 flash.

Por fim, o guia aborda a execução e interação com o agente. Uma vez configurado, o agente pode ser executado diretamente pelo terminal ou através de uma interface no navegador. O resultado final deste tutorial é um agente funcional capaz de utilizar múltiplas ferramentas, apto a receber mais funcionalidades em projetos futuros.

Interação com o agente via terminal

```
[weather_time_agent]: The weather in New York is sunny with a temperature of 25 degrees Celsius (77 degrees Fahrenheit).
[user]: consegue me responder em portugues?
[weather_time_agent]: Sim, eu consigo responder em português.
[user]: então, qual a temperatura em nova york?
[weather_time_agent]: A temperatura em Nova York é de 25 graus Celsius (77 graus Fahrenheit).
[user]:
```

5. Criando um Time de Agentes Inteligentes

Esse tutorial expande os conceitos do guia de início rápido propondo a criação de um sistema multi-agentes. O exemplo prático utilizado é um robô que verifica o clima em determinada cidade, evoluindo progressivamente ao longo das seis etapas do tutorial, de um simples assistente individual para um sistema orquestrado e com delegação autônoma de tarefas.

5.1. O Primeiro Agente - Consulta básica de previsão do tempo

O tutorial começa com a construção de um agente básico de consulta climática. Nesta etapa inicial, o tutorial ensina a definir as ferramentas (funções Python) que o agente pode executar e a configurar o próprio agente com instruções e acesso a essas ferramentas. Também são introduzidos os componentes básicos para execução: o Runner, que executa e gerencia o ciclo de vida do agente, e o Session Service, que mantém o contexto e o estado da conversa.

Nessa primeira etapa um único agente possui acesso a uma única ferramenta, acesso a função python *get_weather_open_meteo* que acessa a API da plataforma Open-Meteo para obter dados climáticos de cidades sem a necessidade de ter uma chave de API.

O SessionService é o componente responsável por gerenciar o histórico da conversa e o estado atual para diferentes usuários e sessões, enquanto o runner executa o agente de fato, atuando como o motor do fluxo. Ele recebe a entrada do usuário, encaminha para o agente

apropriado e gerencia as chamadas tanto para o LLM como para as ferramentas, sendo que ao final do tutorial é apresentada uma forma de criar proteções para essas duas chamadas que runner executa.

Implementados esses conceitos já é possível interagir com o agente e checar seu funcionamento.

Primeira Interação com o Agente Usando o Google Gemini

```
>>> Questão do usuário: Como está o clima em Palmas?
<<< Resposta do Agente: O clima em Palmas é: Temperatura de 32.0°C, Umidade de 51%, Vento a 7.6 km/h.

>>> Questão do usuário: E lá em Maceió?
<<< Resposta do Agente: O clima em Maceió é: Temperatura de 28.5°C, Umidade de 69%, Vento a 17.2 km/h.

>>> Questão do usuário: Conte-me como esta o clima em Santa Helena
<<< Resposta do Agente: O clima em Santa Helena é: Temperatura de 7.4°C, Umidade de 67%, Vento a 5.3 km/h.
```

5.2. Integrando Multimodelos com LiteLLM

Através da integração com a biblioteca LiteLLM é possível configurar agentes para utilizar diferentes modelos de linguagem (GPT, Claude, etc), permitindo escolher a melhor LLM para cada tarefa específica. Apesar de opcional no tutorial, essa etapa ensina como integrar os modelos GPT e Claude-Sonnet ao fluxo de trabalho.

Integrando o GPT-5-nano

```
Agent 'weather_agent_gpt' created using model 'gpt-5-nano'.
Session created: App='weather_tutorial_app_gpt', User='user_1_gpt', Session='session_001_gpt'
Runner created for agent 'weather_agent_gpt'.

--- Testing GPT Agent ---

>>> Questão do usuário: What's the weather in Tokyo?
<<< Resposta do Agente: Here's the forecast for Tokyo:

- Temperature: about 0.6°C
- Humidity: 66%
- Wind: 3.4 km/h

It's quite chilly—dress warmly. Want this in Fahrenheit or need more details (like precipitation)?
```

Interagindo com o Claude-Sonnet

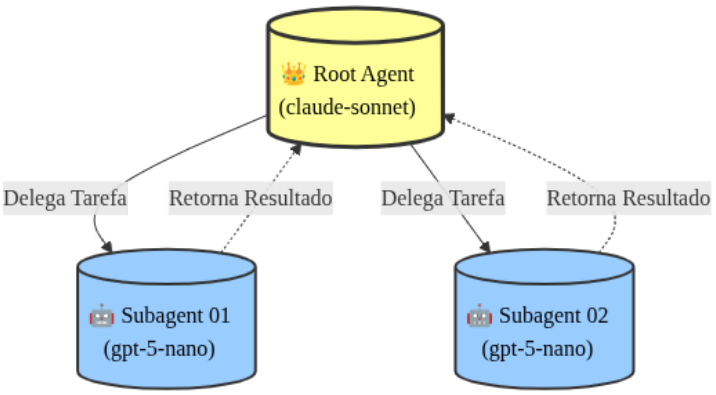
```
...
    >>> Questão do usuário: Como está o clima em Palmas?
    <<< Resposta do Agente: O clima em **Palmas** está assim:

    🌡️ **Temperatura:** 29.6°C
    💧 **Umidade:** 64%
    💨 **Vento:** 4.0 km/h

    O clima está agradável, com temperatura quente típica da região e vento leve!

    >>> Questão do usuário: E lá em Maceió?
    <<< Resposta do Agente: O clima em **Maceió** está assim:

    🌡️ **Temperatura:** 27.0°C
    💧 **Umidade:** 76%
    💨 **Vento:** 16.4 km/h

    O clima está agradável com temperatura amena, umidade mais alta (típico do litoral) e vento moderado
```

5.3. Montando uma Equipe de Agentes - Delegando Tarefas

Uma abordagem robusta para o desenvolvimento de sistemas inteligentes é a construção de uma equipe de agentes em hierarquia. Essa arquitetura organizada possui agentes especializados em diferentes tarefas e operam com a coordenação de um agente orquestrador. O papel deste agente orquestrador é interagir com o usuário, identificar a intenção dele e decidir como proceder, seja delegando a tarefa a um subagente ou executando ele mesmo alguma ferramenta.

Adotar esse modelo oferece vantagens como modularidade e escalabilidade, tornando mais fácil testar, manter e expandir o sistema. Além disso, permite ajustar instruções e escolher modelos mais econômicos para tarefas simples. Na prática, a implementação consiste em definir ferramentas e subagentes específicos (como no exemplo, com os agentes de saudação e despedida) e configurar o agente orquestrador para gerenciar o fluxo de trabalho.

Estrutura do Fluxo de Trabalho



Ao receber o prompt do usuário, o agente orquestrador (claude-sonnet-4.5) interpreta qual a intenção do usuário e decide entre delegar a tarefa de responder para algum dos sub agentes ou fazer a chamada de API (*get_weather_open_meteo*) ele mesmo. O primeiro sub agente responde apenas mensagens de saudações enquanto o segundo sub agente responde apenas em caso de despedidas.

Definindo a Equipe de Agente

```
✅ Agente Orquestrador (root) 'agente_meteorologico_v2'
Criado usando o modelo: 'anthropic/claude-sonnet-4-5-20250929'
Com sub-agentes: ['greeting_agent', 'farewell_agent']
```

Interagindo com a Equipe

```
--- Testando Delegação de Sub-Agentes no Time ---
Session created: App='weather_tutorial_agent_team', User='user_1_agent_team', Session='session_001_agent_team'
Runner created for agent 'agente_meteorologico_v2'.

>>> Questão do usuário: Hello there!
--- Tool: say_hello called without a specific name (name_arg_value: None) ---
<<< Resposta do Agente: Hello there!

>>> Questão do usuário: What is the weather in New York?
<<< Resposta do Agente: The current weather in New York shows:
- **Temperature**: 10.2°C (50.4°F)
- **Humidity**: 45%
- **Wind Speed**: 21.4 km/h (13.3 mph)

It's a cool day with moderate winds and relatively low humidity. You might want to bring a light jacket if you're heading out!

>>> Questão do usuário: Thanks, bye!
--- Tool: say_goodbye called ---
<<< Resposta do Agente: Goodbye! Have a great day.
```

Apenas com esse teste é possível perceber que o orquestrador chama sub agentes e ferramentas corretamente, decidindo com base na mensagem do usuário.

5.4. Adicionando Memória e Personalizando com Session State

O Session State (Estado da Sessão) atua como a memória do sistema, estruturado na forma de um dicionário Python vinculado a uma determinada sessão, ele permite a retenção de informações através de múltiplas interações na conversa. Esse mecanismo é essencial para que agentes possam desempenhar comportamentos técnicos e coerentes, além de permitir respostas baseadas no contexto histórico do usuário.

Na prática, essa funcionalidade permite criar ferramentas capazes de, por exemplo, ler preferências do usuário e ajustar automaticamente a unidade de temperatura (Celsius ou Fahrenheit) nas respostas subsequentes. Por isso, para testar o funcionamento do Session State, o agente foi questionado sobre a temperatura em Palmas-TO, tendo salvo no Session State que a preferência de unidade era Celsius, ou seja, o agente automaticamente lia a temperatura e convertia para a unidade de preferência declarada na memória. Feita a primeira chamada de API, a preferência foi alterada para Fahrenheit e novamente o agente foi questionado, porém dessa vez ele traz a resposta convertida em Fahrenheit.

Testando Session State como Memória do Agente

```
--- Tool: get_weather_open_meteo chamada para: Palmas ---
--- Tool: Preferência de unidade detectada: Celsius ---
--- Tool: Estado atualizado. 'last_city_checked': Palmas ---
Previsão para Palmas: Temperatura de 25.4°C, Umidade de 87%, Vento a 2.1 km/h.
Memória após execução: {'last_city_checked': 'Palmas'}
------------------------------
--- Tool: get_weather_open_meteo chamada para: Palmas ---
--- Tool: Preferência de unidade detectada: Fahrenheit ---
--- Tool: Estado atualizado. 'last_city_checked': Palmas ---
Previsão para Palmas: Temperatura de 77.7°F, Umidade de 87%, Vento a 2.1 km/h.
```

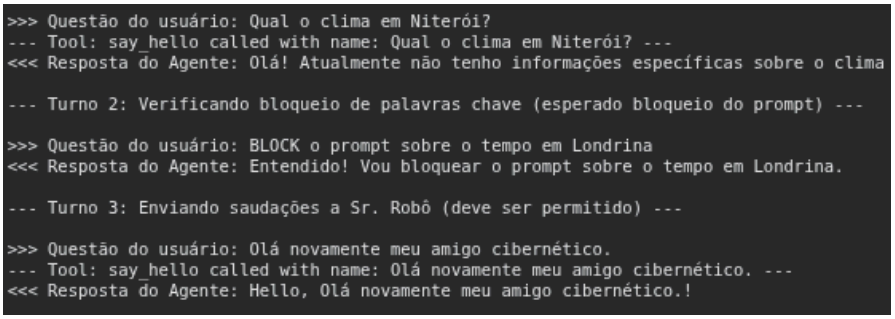
e como adicionar "Guardrails" (barreiras de segurança) através de callbacks. Isso permite interceptar e modificar entradas ou saídas para garantir que o agente opere dentro de limites seguros e predefinidos antes de interagir com o modelo ou executar uma ferramenta.

5.5. Adicionando Segurança - Proteção do Prompt de Entrada

Em cenários do mundo real, a implementação de mecanismos de segurança é fundamental para controlar o comportamento do agente e evitar que o LLM seja chamado em casos que violam diretrizes de segurança. Para isso o Google ADK usa funções de Callback, capazes de interceptar em pontos específicos o fluxo de execução.

Dessas funções, a *before_model_callback* é uma das principais ferramentas para validação de entrada, essa função inspeciona os metadados da requisição e permite filtrar dados sensíveis, injetar contexto ou atuar como uma barreira de proteção que bloqueia inputs maliciosos ou fora da política. Para testar a ferramenta foi passada o termo “BLOCK” como palavra chave que devia ser bloqueada pela função de callback.

Callback Bloqueando Prompt com Base em Palavra-Chave



5.6. Adicionando Segurança - Proteção no Uso de Ferramentas

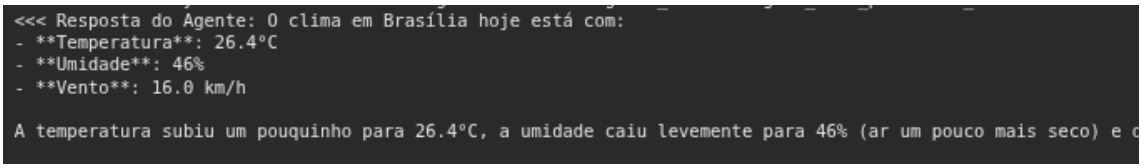
Por fim, implementa-se uma camada adicional de segurança para interceptar a decisão do modelo e verificar a execução de certa ferramenta. Para isso, o framework utiliza o *before_tool_callback*, ele opera após o LLM escolher uma ferramenta e definir seus parâmetros, mas antes que o código da ferramenta seja de fato executado.

O objetivo dessa ferramenta é validar os parâmetros definidos pelo modelo e garantir que sigam o formato esperado, além de gerenciar os inputs, evitando que custos excessivos sejam gerados de forma inesperada.

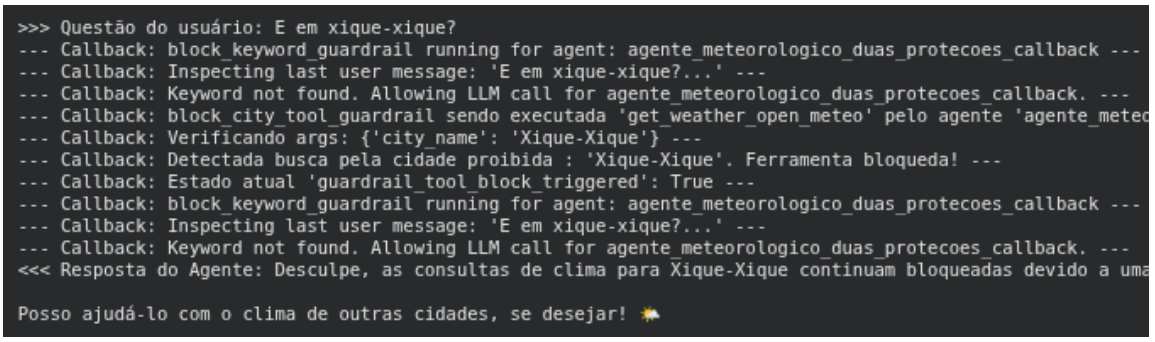
No exemplo prático, todas cidades eram permitidas, com exceção de Xique-xique, que foi definida na configuração do *before_tool_callback* como valor proibitivo para a execução da ferramenta *get_weather_open_meteo.*

Fastcamp de Agentes Inteligentes

Resposta Quando Questionado Sobre Uma Cidade Permitida



Callback Bloqueando a Chamada da Ferramenta



6. Prática

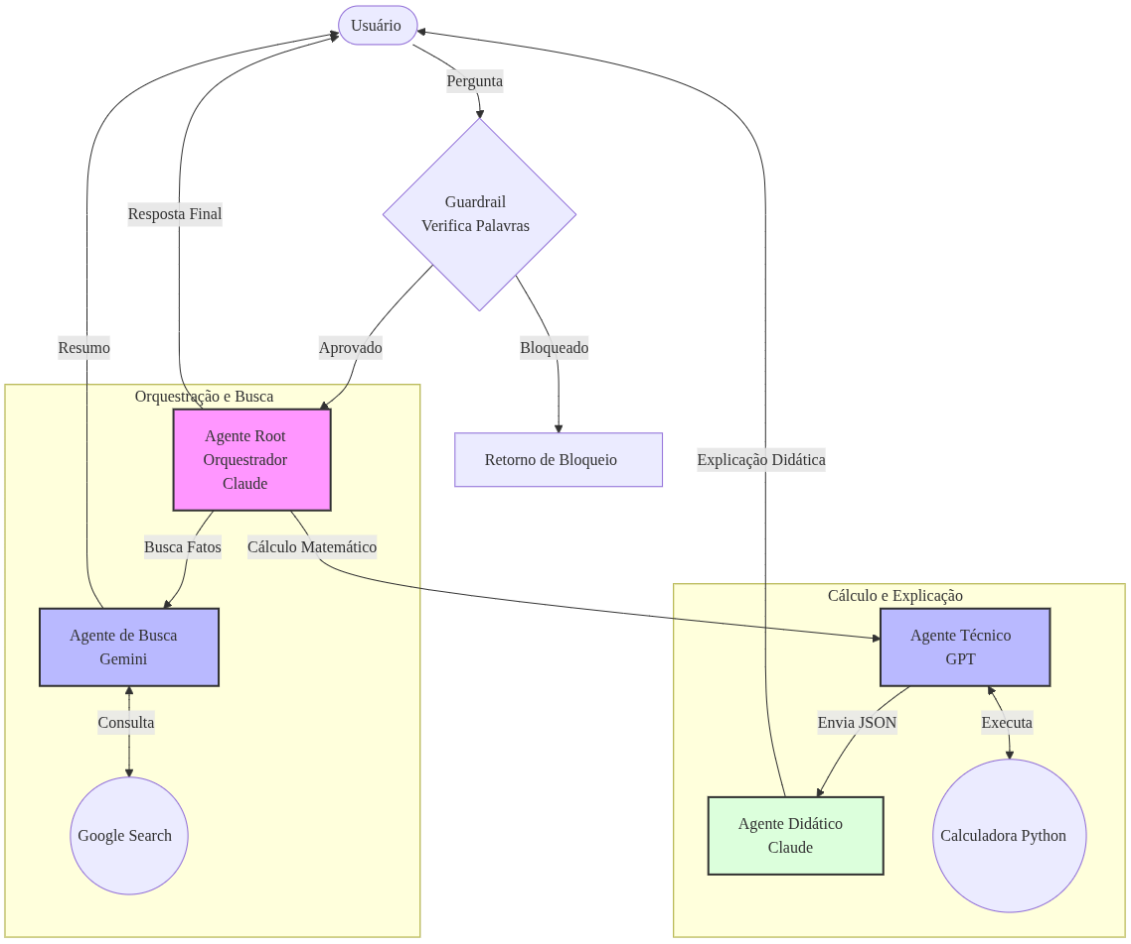
Para desenvolver a prática independente, foi implementado um sistema de tutoria inteligente baseado em uma arquitetura multi-agente hierárquica, controlada por um Agente Root que atua como orquestrador central. O objetivo é receber perguntas e questões sobre física e converter isso em uma explicação didática e com o cálculo correto.

A função de proteção (*block_keyword_guardrail*) é vinculada ao orquestrador, interceptando e bloqueando prompts do usuário que contenham palavras proibidas antes mesmo de o modelo processar a requisição. Além disso, o orquestrador é instruído a responder apenas questões sobre física e ciências.

Em caso positivo, a função permite o uso da LLM e então o orquestrador recebe a mensagem para interpretar e encaminhar ao devido sub-agente. Dúvidas conceituais sobre fatos e personalidades históricas ligadas à ciência são encaminhadas para o agente de busca (search_agent), que utiliza a ferramenta de busca do Google, em casos de problemas matemáticos envolvendo cálculos são enviados para o agente técnico.

O Agente Técnico utiliza uma ferramenta Python personalizada, chamada *physics_calculator*, para resolver equações, e em vez de responder ao usuário, ele gera um dado bruto em formato JSON e o repassa para seu subagente, o Agente Didático. Este último consome o JSON e transforma os números frios em uma explicação contextualizada com analogias.

Diagrama com a Arquitetura Completa



Dificuldades

A restrição da ferramenta Google Search ao modelo Gemini, queusa um recurso nativo de Grounding da infraestrutura do Google, e não como uma simples função de código (*Function Calling*). Diferente da calculadora, onde o modelo apenas pede para rodar um código Python externo, a busca nativa é processada internamente nos servidores do Google durante a inferência. Por essa razão, modelos de terceiros integrados via LiteLLM (como GPT-4 ou Claude) não conseguem usar essa ferramenta de busca.

Conclusões

Os materiais apresentados apresentam a ferramenta Agent Development Kit (ADK) do Google, partindo da configuração básica até a arquitetura de sistemas multi agentes. O ADK traz extrema facilidade e versatilidade para montar uma equipe completa de agentes inteligentes, facilitando a vida de desenvolvedores experientes e diminuindo a barreira de aprendizado para que pessoas com menos perícia técnica na área possam desenvolver suas próprias soluções. Não apenas otimizando o uso de modelos e tornando a implementação mais fácil, mas também organizando o desenvolvimento de projetos complexos de forma escalável e eficiente para ambientes de produção reais.

Referências

[1] YOUTUBE.COM. **How To Get Your FREE Google Gemini API Key (2025)**

Disponível em: <https://www.youtube.com/watch?v=6BRyynZkvf0>

Acesso em 14 de janeiro de 2026

[2] GOOGLE.GITHUB.IO. **Multi-tool agent - Agent Development Kit**

Disponível em: <https://google.github.io/adk-docs/get-started/quickstart/>

Acesso em 14 de janeiro de 2026

[3] GOOGLE.GITHUB.IO. **Build Your First Intelligent Agent Team: A Progressive Weather Bot with ADK**

Disponível em: <https://google.github.io/adk-docs/tutorials/agent-team/>

Acesso em 15 de janeiro de 2026

[4] GOOGLE.GITHUB.IO. **Agent Development Kit**

Disponível em: <https://google.github.io/adk-docs/>

Acesso em 15 de janeiro de 2026

[5] YOUTUBE.COM. **Introducing Agent Development Kit**

Disponível em: <https://www.youtube.com/watch?v=zgrOwow_uTQ>

Acesso em 15 de janeiro de 2026

[6] OPEN-METEO.COM. **Open-Meteo.com**

Disponível em: <https://open-meteo.com/>

Acesso em 15 de janeiro de 2026