

# Agents

# Introduction

# Agents

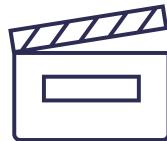
What is an Agent?

Perceives  
Environment

Takes Actions  
autonomously

To achieve a  
defined goal

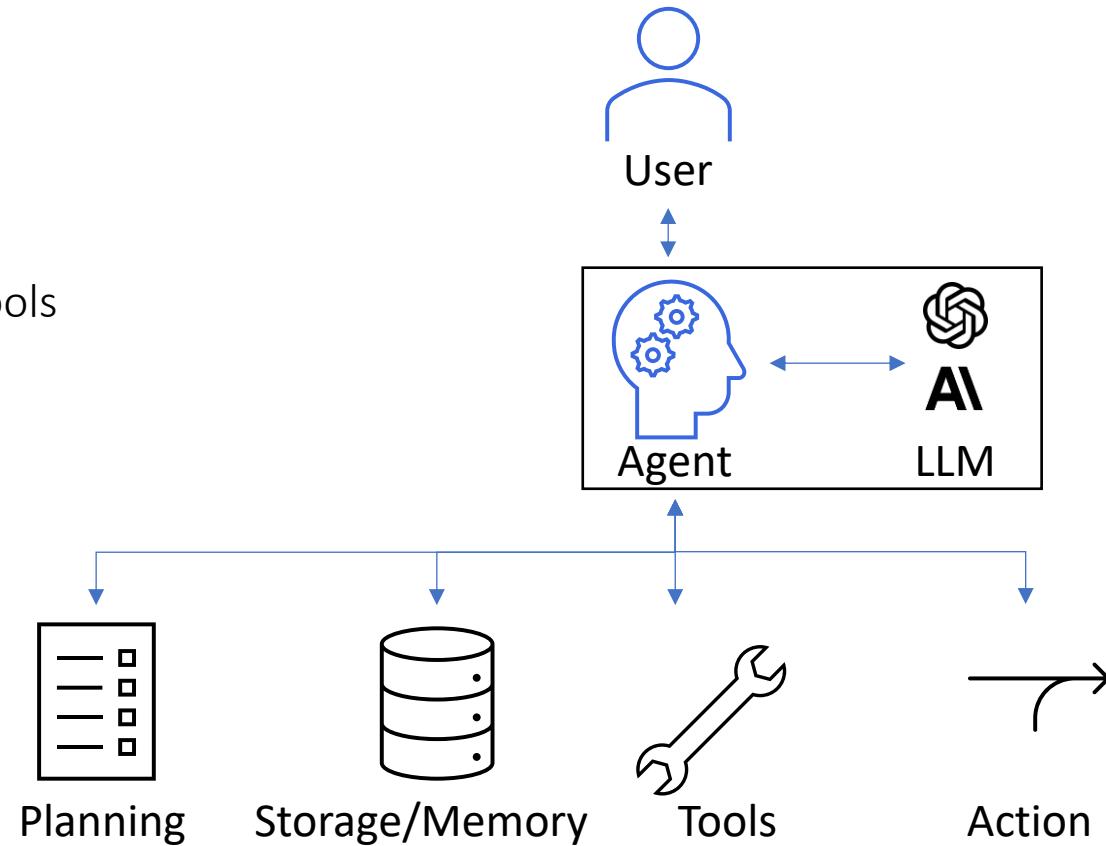
Improve  
through  
learning



# Agents

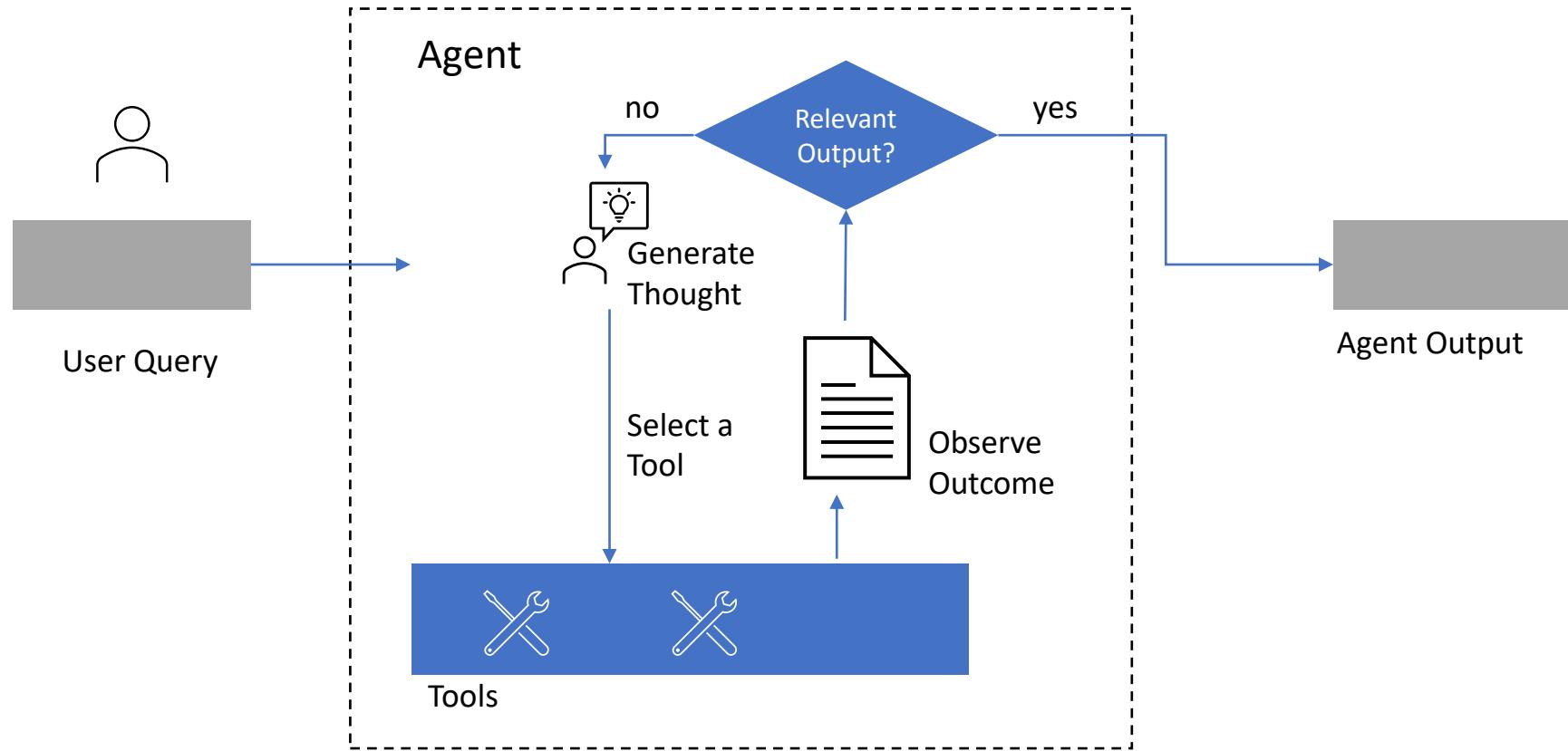
What is an Agent?

- LLM apps execute tasks
- core element: agent
- uses planning, memory, and tools
- can perform actions
- is an expert in its field



# Agents

How does an Agent work?



# Agents

## Levels of AI Agents

**Generality**

Level	Techniques	Performance	Capabilities	Key Characteristics	Use Cases	Narrow Domain	General Wide-Range Domain
5	LLM-based AI + Tools (Intent + Actions + Reasoning & Decision Making + Memory + Reflection + Autonomous Learning + Generalisation + Personality (Emotion + Character) + Collaborative behaviour (Multi-Agents))	Superhuman > 100% of Skilled Adults	True Digital Persona	Agent represents the user in completing affairs, interacts on behalf of user with others, ensuring safety & reliability.	Agent acts on behalf of user to complete tasks, interacting with others while ensuring safety & reliability.	Superhuman Narrow-AI AlphaFold, AlphaZero, StockFish	Artificial Super Intelligence (ASI) Not yet achieved
4	LLM-based AI + Tools (Intent + Actions + Reasoning & Decision Making + Memory & Reflection + Autonomous Learning + Generalisation)	Virtuoso Equal to 99% of Skilled Adults	Memory & Context Awareness	Agent senses user context, understands user memory, and proactively provides personalised services at times.	A personalised virtual assistant enhances UX by understanding context & memory while acting proactively.	Virtuoso Narrow-AI AlphaGo, Deep Blue	Virtuoso AGI Not yet achieved
3	LLM-based AI + Tools (Intent + Actions) + Reasoning & Decision Making + Memory & Reflection	Expert Equal to 90% of Skilled Adults	Strategic task Automation	Using user-defined tasks, agents autonomously plan, execution steps using tools, iterates based on intermediate feedback until completion.	Agents autonomously plan and execute steps based on intermediate feedback.	Expert Narrow-AI Purpose build, specific task orientated Agents	Expert AGI Not yet achieved
2	IL/RL-based AI + Tools (Intent + Actions) + Reasoning & Decision Making	Competent Equal to 50% of Skilled Adults	Deterministic Task Automation of Skilled Adults	Based on user description of deterministic task, agent auto-completes steps in predefine action.	User: "Check the weather in Beijing today".	Competent Narrow-AI Conversational AI build frameworks with LLM, RAG, etc.	Competent AGI Not yet achieved
1	Rule-Based AI + Tools (Intent + Actions)	Emerging Equal to Unskilled Humans	Simple Step Sequence	Agents complete tasks following exact steps, pre-defined by users or developers.	User: "Open Messenger" User: "Open the first unread email in my mailbox and read its content." User: "Call Alice".	Emerging Narrow-AI Single Rule-based systems, SHRDLU, GOFAI	Emerging AGI ChatGPT, Gemini, Llama 2, etc.
0	No AI Tools (Intent + Rules + Actions)	No AI	No AI	No AI	No AI	Narrow Non-AI UI Driven Software	General Non-AI Human-In-The-Loop Computing Mechanical Turk

Adapted From: <https://arxiv.org/pdf/2405.06643.pdf>

Source: <https://cobusgreylings.medium.com/5-levels-of-ai-agents-updated-0ddf8931a1c6>

# Coding Agents



# Agents

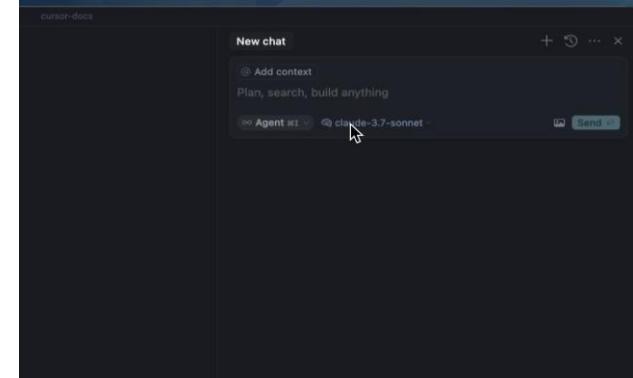
## Coding Agents: Evolution

Auto Completion

Inline Complete

Complete code base

```
-- 33 You, 9 seconds ago - 1 Author (You) 0 implementations
34 pub struct EncodedMessageQueue {
35     pub queue: Vec<(EncodedMessage, ClientFilter)>,
36
37     sender: Arc<Sender<Vec<(EncodedMessage, ClientFilter)>>,
38     receiver: Arc<Receiver<Vec<(EncodedMessage, ClientFilter)>>,
39 }
40
41 impl EncodedMessageQueue { You, 8 seconds ago - Uncommitted changes
42     pub fn new() -> Self {
43         let (sender, receiver) = crossbeam_channel::unbounded();
44         Self {
45             queue: vec![],
46             sender: Arc::new(sender),
47             receiver: Arc::new(receiver),
48         }
49     }
50 }
```



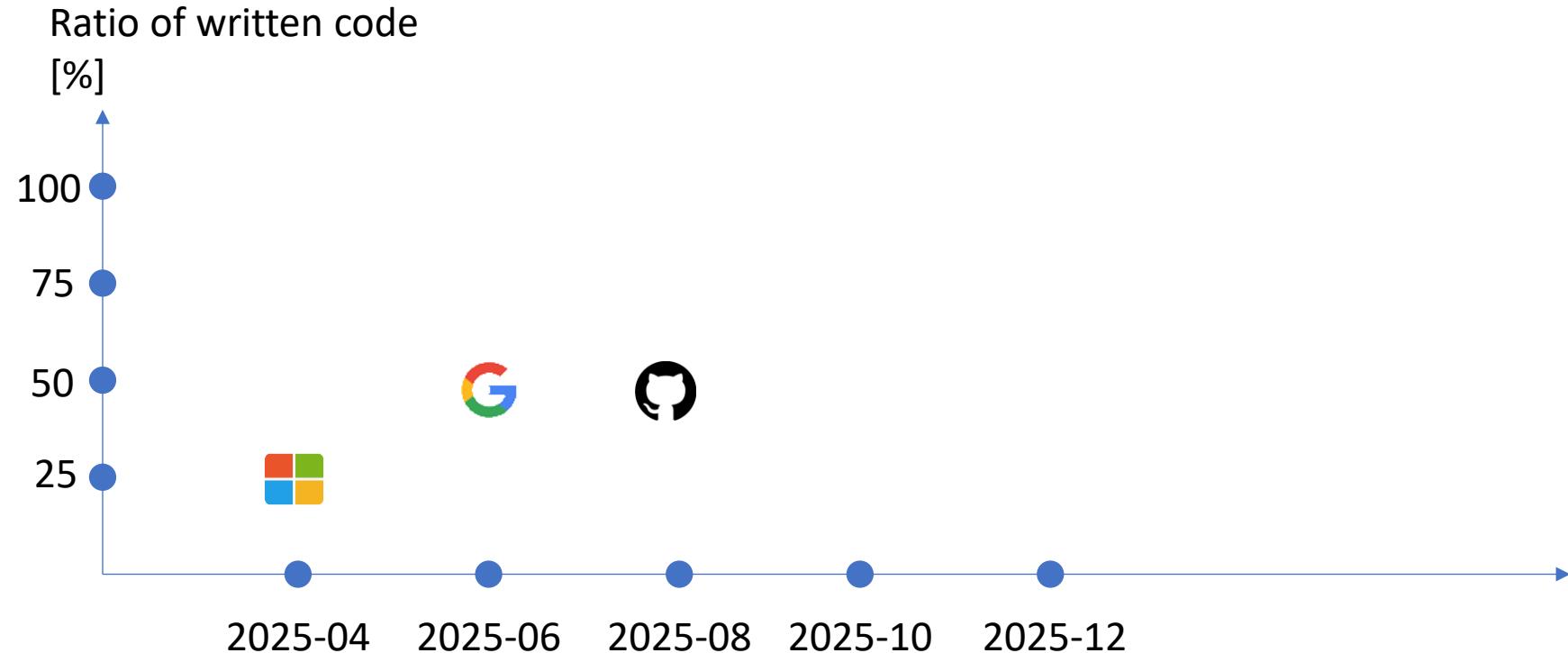
- Proposes automatically next code tokens

- Edits multiple lines
- Available in VS Code, Cursor, Windsurf

- Accesses complete code base
- Performs edits

# Agents

Coding Agents: How much code is written by AI?



# Agents

Coding Agents: Browser-Based

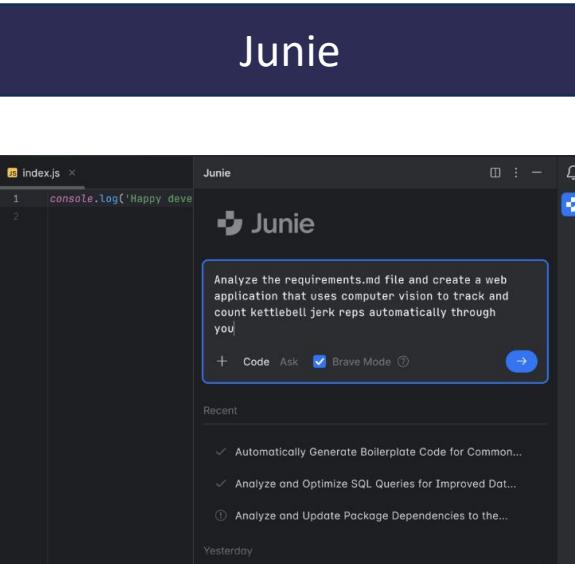


- Browser-based
- freemium
- <https://replit.com/>

- Browser-based
- freemium
- <https://lovable.dev/>

# Agents

## Coding Agents: Browser-Based

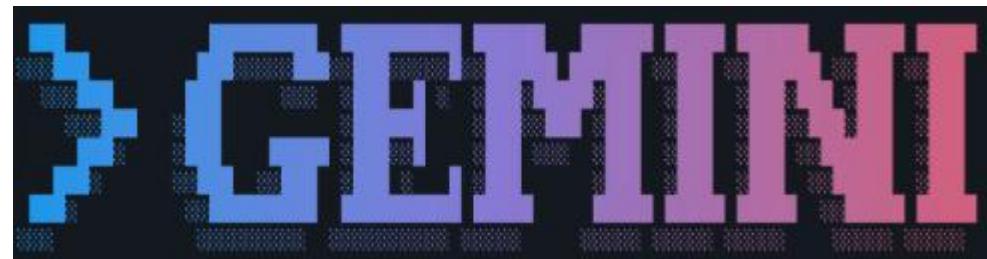
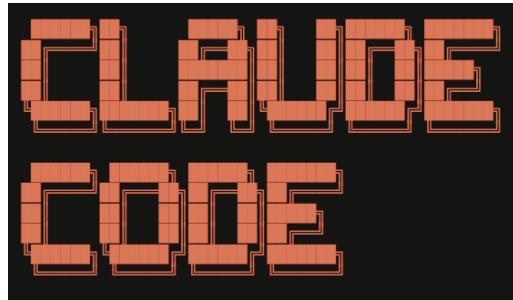


- Developed by Jetbrain
- Available in Jetbrain-IDEs (e.g. PyCharm)

- Browser-based
- Developed by Google
- Integrates with GitHub

# Agents

Coding Agents: Terminal



- Works from command line
- Can work with your complete code base
- Install: `npm install -g @anthropic-ai/clause-code`
- Available for clause subscribers or API-use
- [Source](#)
- Can work with your complete code base
- Install: `npm install -g @google/gemini-cli`
- Available via API key or Google SignIn
- 100 Gemini 2.5 Pro requests free per day!

# Agents

Coding Agents: Terminal

- Works from command line
- Can work with your complete code base
- Install: `npm install -g @anthropic-ai/clause-code`
- Available for clause subscribers or API-use
- [Source](#)
- Can work with your complete code base
- Install: `npm install -g @google/gemini-cli`
- Available via API key or Google SignIn
- 100 Gemini 2.5 Pro requests free per day!

# Agents

Coding Agents: Terminal

## Codex

- Developed by OpenAI
- Included in ChatGPT Plus, Pro, Business, Edu, Enterprise, or via API-key
- Works from command line

# Agents

Coding Agents: IDE



## Cursor

- 20\$/month
- Suitable for
  - Experienced devs, large codebase
- Makes Claude 4 available

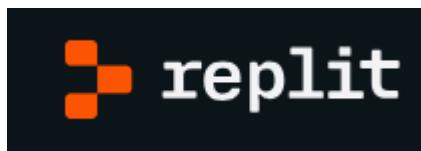


## Windsurf

- 15\$/month
- Suitable for
  - Beginner, vibe-coding, personal projects
- Develop web apps quick and without configuration effort

# Agents

Coding Agents: IDE



Beginner



Intermediate



User Experience Level



# Agents

Coding Agents: Exercise

## User Prompt

Diese Anwendung liest einen kurzen Text ein und bewertet dessen emotionale Stimmung (z. B. positiv, negativ, neutral) mithilfe von Sentiment-Analyse. Basierend auf dem ermittelten Vibe generiert die App eine Webseite. Beispielsweise könnte ein positiver Text eine helle, freundliche Webseite mit vielen leuchtenden Farben und optimistischen Sprüchen erzeugen, während ein negativer Text eine düstere, minimalistische Seite mit gedämpften Farben und melancholischen Zitaten generiert. Der Code wird in Python geschrieben und streamlit für die App verwendet.



- 1 Erstellen der Projektdateien: Ich erstelle die Hauptanwendungsdatei `app.py` und die Abhängigkeitsdatei `requirements.txt`.
- 2 Abhängigkeiten installieren: Ich installiere die in `requirements.txt` aufgeführten Bibliotheken (`streamlit` und `textblob`).
- 3 Implementieren der Kernlogik: Ich schreibe den Python-Code in `app.py`. Dies beinhaltet die Einrichtung der Streamlit-Benutzeroberfläche, die Integration der Sentiment-Analyse mit TextBlob und die Logik zur dynamischen Generierung von Webseiten basierend auf der erkannten Stimmung.
- 4 Testen der Anwendung: Ich führe die Streamlit-Anwendung aus, um sicherzustellen, dass sie wie erwartet funktioniert. Ich werde Texte mit unterschiedlichen Stimmungen eingeben, um die generierten Webseiten zu überprüfen.
- 5 Code einreichen: Sobald die Anwendung voll funktionsfähig ist, reiche ich den Code ein.

Approve plan? →

# Agents

## Coding Agents: Exercise



A /app.py

```
app.py requirements.txt
A /app.py
1 + import streamlit as st
2 + from textblob import TextBlob
3 +
4 + def generate_website(sentiment):
5+     """Generiert eine Webseite basierend auf der Stimmung."""
6+     if sentiment == "positiv":
7+         st.markdown("""
8+             <style>
9+             .container {
10+                 background-color: #ffecb3;
11+                 color: #333;
12+                 padding: 20px;
13+                 border-radius: 10px;
14+                 text-align: center;
15+                 font-family: 'Arial', sans-serif;
16+             }
17+             h1 {
18+                 color: #ff8f00;
19+             }
20+             </style>
21+             <div class="container">
22+                 <h1>Ein sonniger Tag! ☀</h1>
23+                 <p>Die Stimmung ist wunderbar positiv. Hier ist ein Lächeln für dich! 😊</p>
24+                 <p><i>"Behalte dein Gesicht immer im Sonnenschein - und Schatten werden hinter dir"</i></p>
25+             </div>
26+             """, unsafe_allow_html=True)
27+     elif sentiment == "negativ":
28+         st.markdown("""
29+             <style>
30+             .container {
31+                 background-color: #424242;
32+                 color: #f5f5f5;
33+                 padding: 20px;
34+                 border-radius: 10px;
35+                 text-align: center;
36+                 font-family: 'Georgia', serif;
37+             }
38+             h1 {
39+                 color: #9e9e9e;
40+             }
41+             </style>
42+             <div class="container">
43+                 <h1>Es tut mir leid, du siehst traurig aus! ☹</h1>
44+                 <p>Die Stimmung ist leider negativ. Ich hoffe, du wirst bald wieder glücklich sein! 😊</p>
45+             </div>
46+             """, unsafe_allow_html=True)
```

A /requirements.txt

```
app.py requirements.txt
A /requirements.txt
1 -
1 + streamlit
2 + textblob
3 +
```

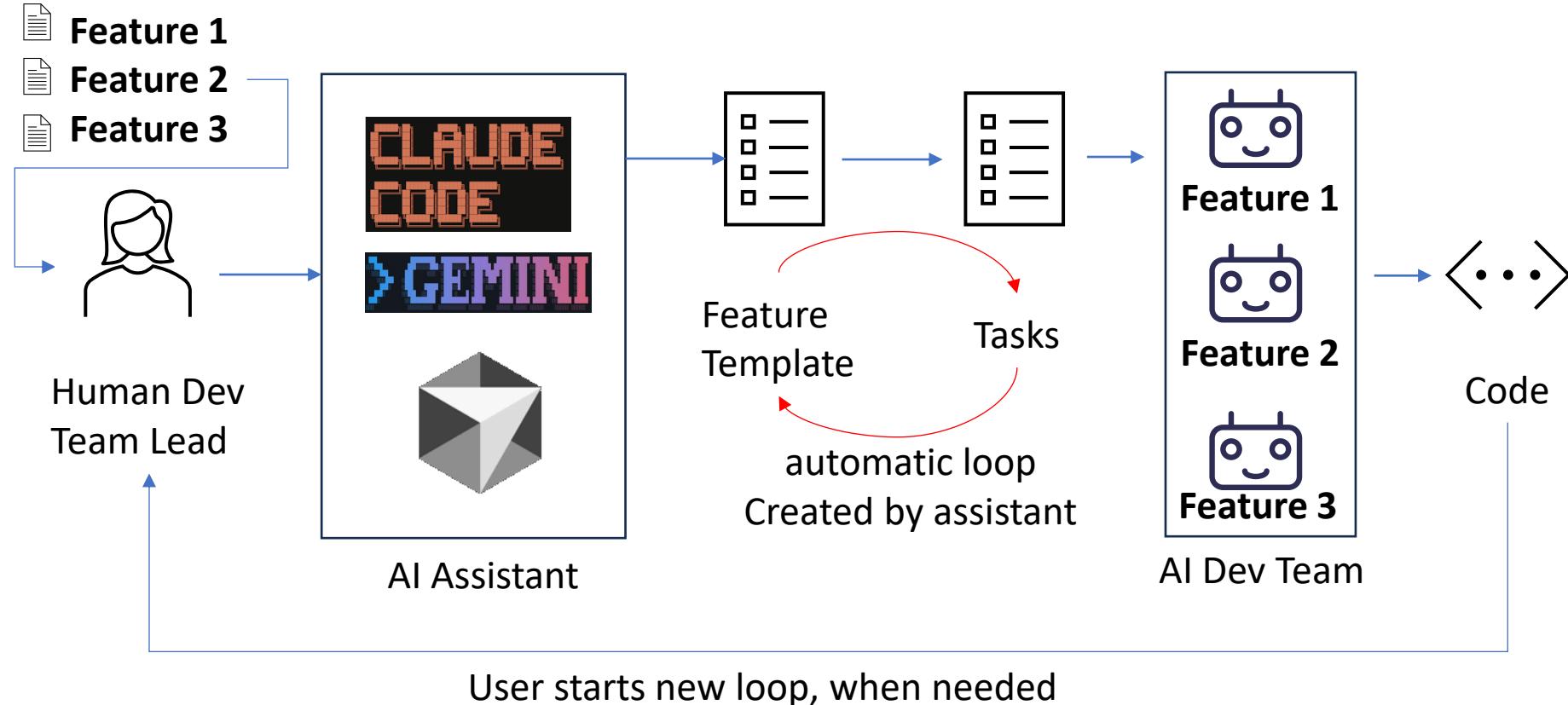
# Agents

Coding Agents: World before Agents



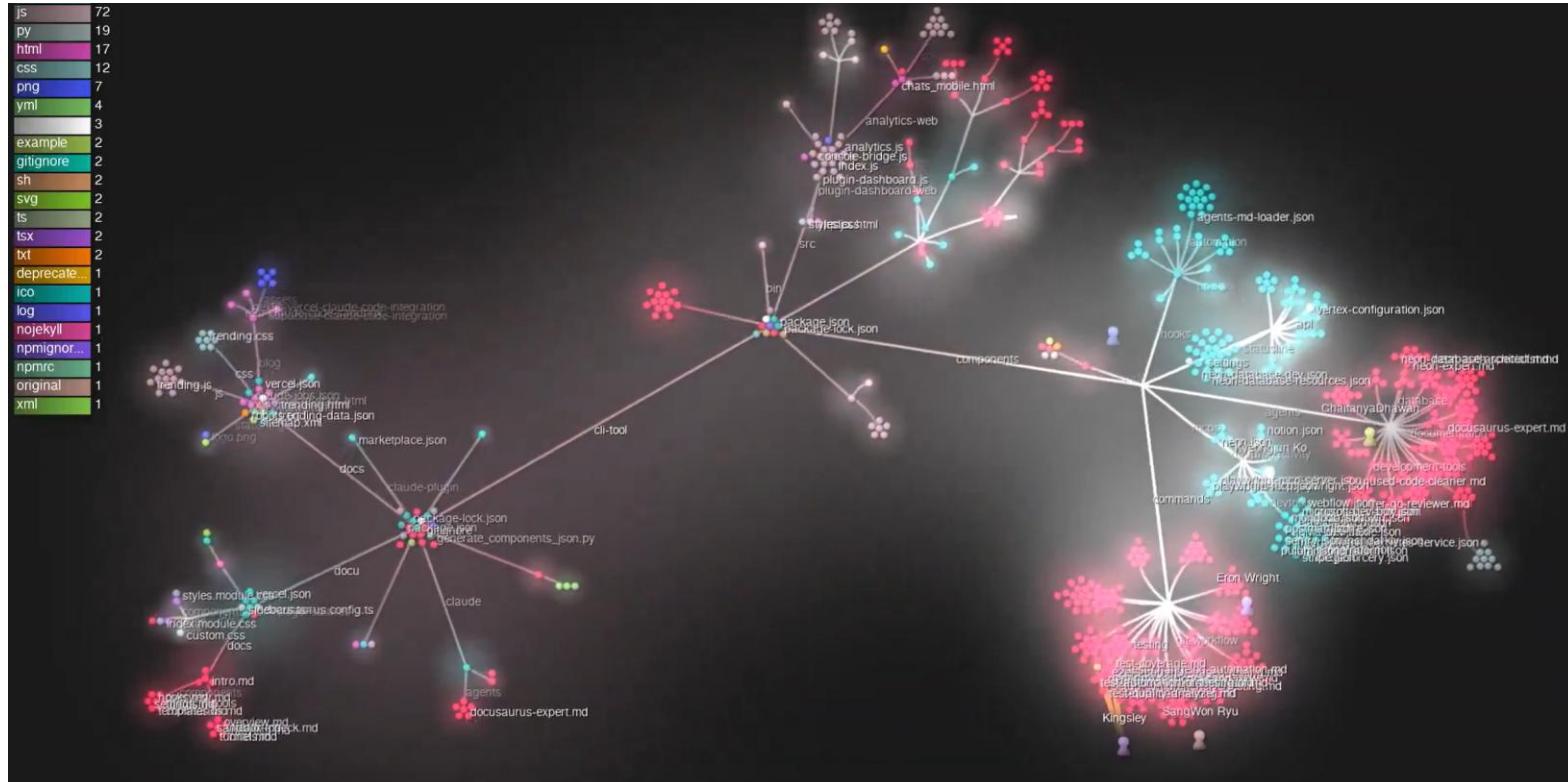
# Agents

## Coding Agents: Workflow



# Agents

# Coding Agents: Claude Code Visualisation



Source: [https://x.com/dani\\_avila7/status/1978510902740877344](https://x.com/dani_avila7/status/1978510902740877344)

# Available Frameworks

# Agents

Which Frameworks are available?



## LangGraph

LangGraph

- built on top of LangChain (same team)
- flexible, customizable
- works with any LLM
- More complex

## CrewAI

CrewAI

- very intuitive
- suitable for many agents
- supports many LLM providers
- not ideal for very complex tasks



## OpenAI

Agents

- very easy to use
- suitable for beginners
- Works with any LLM

# Agents

Which Frameworks are available?



Microsoft

AG2 (formerly: AutoGen)

- mostly for two agents
- good for code generation
- very powerful



Google ADK

- Agent development kit
- Framework for development and deployment of AI agents
- Easy to learn
- Multi-agent architecture



Pydantic AI

Pydantic AI

- Claims to provide production grade applications and workflows
- Model-agnostic

# Agents

Which Frameworks are available?



Magnetic-One

- suitable for beginners
- pre-defined 5 agents: manager, web-surfer, file-surfer, coder, terminal
- built on top of AutoGen
- limited support and documentation

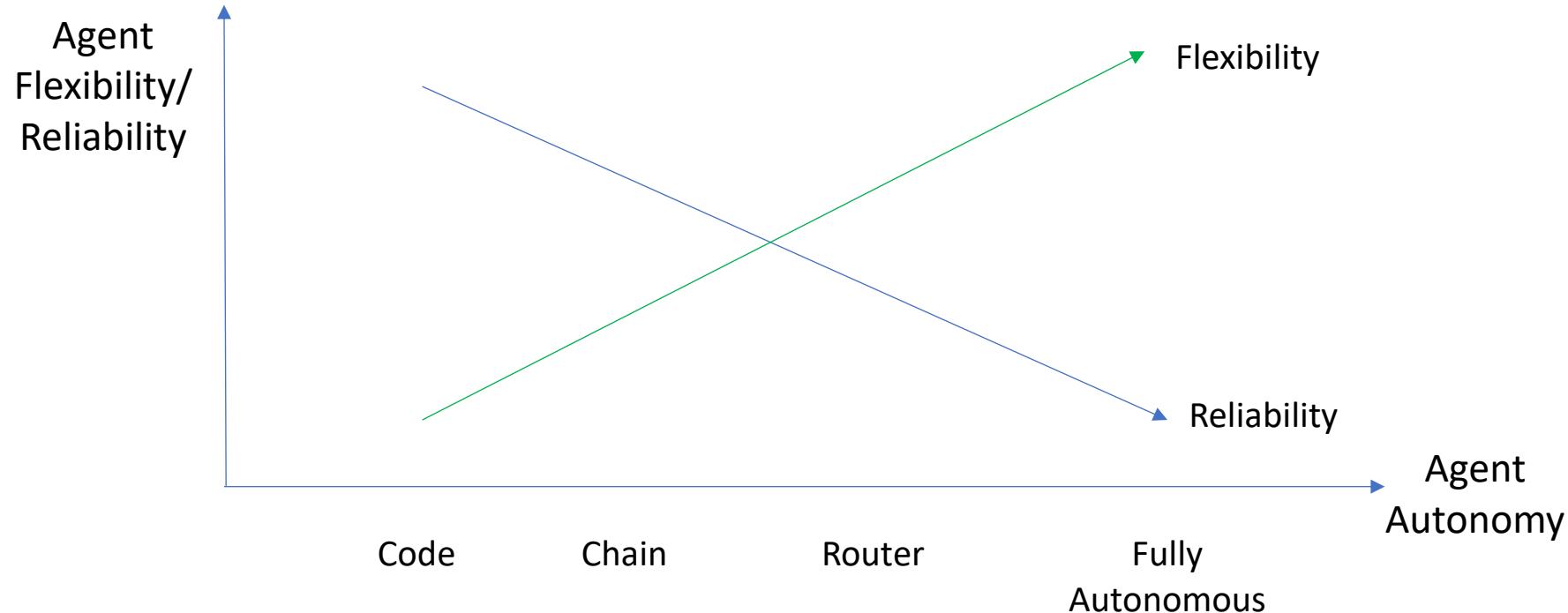


tinytroupe

- multiagent persona simulation for imagination enhancement and business insights
- only works with GPT-4o
- [Link](#)

# Agents

Flexibility / Reliability vs. Autonomy



own graph; adapted from: LangChain Academy „Introduction to LangGraph“

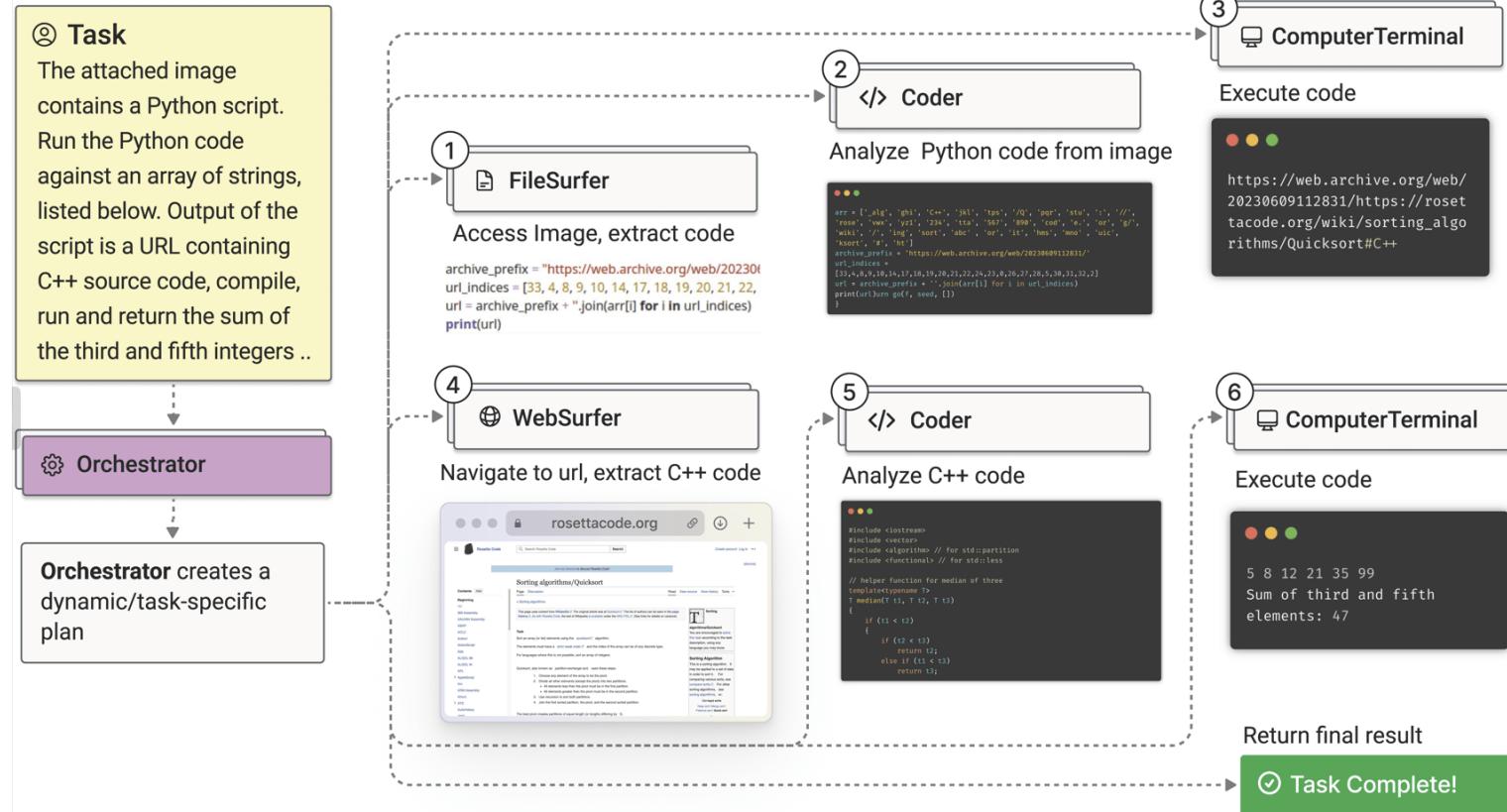
# Agents

Flexibility vs.



# Agents

## Magnetic One



# Agents

Tine Troupe



```
1 factory = TinyPersonFactory("One of the largest banks in Brazil, full of bureaucracy and legacy systems.")
2
3 customer = factory.generate_person(
4     """
5         The vice-president of one product innovation. Has a degree in engineering and a MBA in finance.
6         Is facing a lot of pressure from the board of directors to fight off the competition from the fintechs.
7     """
8 )
✓ 10.1s
```

Python

```
1 customer.minibio()
✓ 0.0s
```

Python

'Lucas Almeida is a 42 year old Vice-President of Product Innovation, Brazilian, currently living in Brazil.'

We can now perform the interview.

```
1 customer.think("I am now talking to a business and technology consultant to help me with my professional problems.")
✓ 0.0s
```

Python

*Lucas Almeida --> Lucas Almeida: [THOUGHT]*  
    > I am now talking to a business and technology consultant to help me with my  
    > professional problems.

TinyPerson(name='Lucas Almeida')

```
1 customer.listen_and_act("What would you say are your main problems today? Please be as specific as possible.",
2                           max_content_length=3000)
✓ 10.9s
```

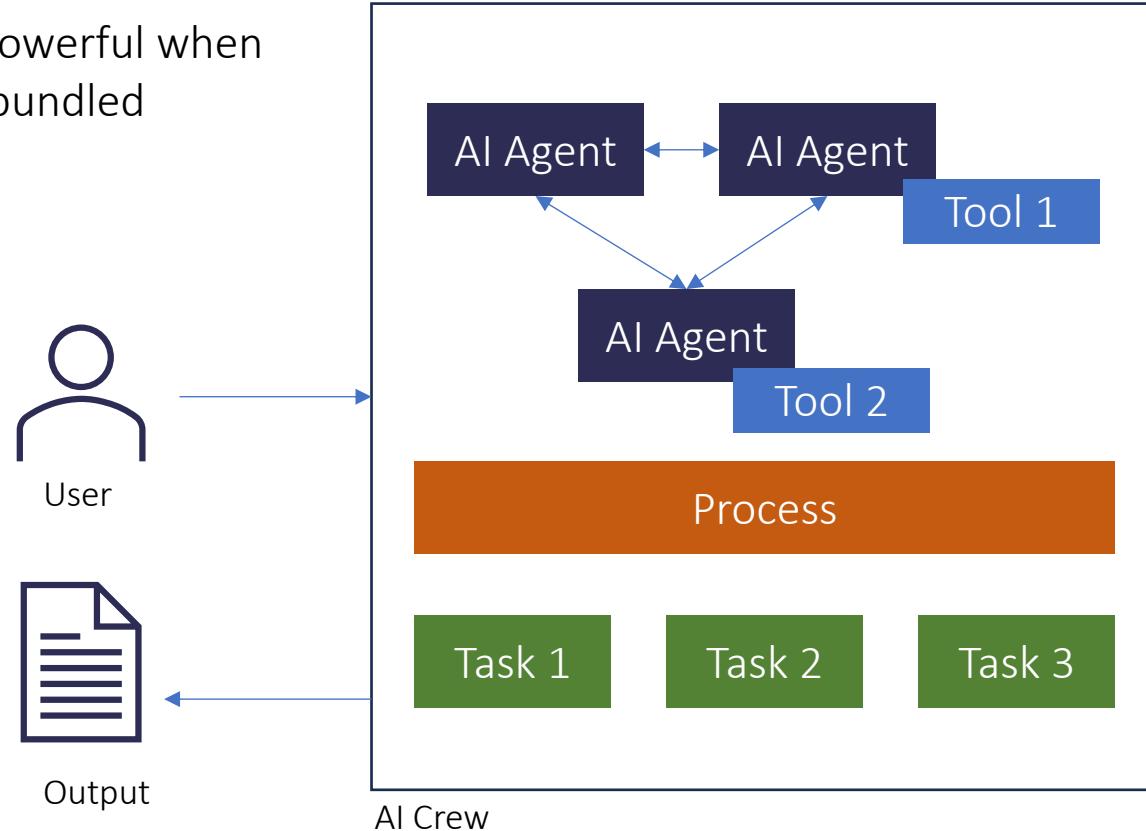
Python

crewAI

# crewAI

What is an AI Crew?

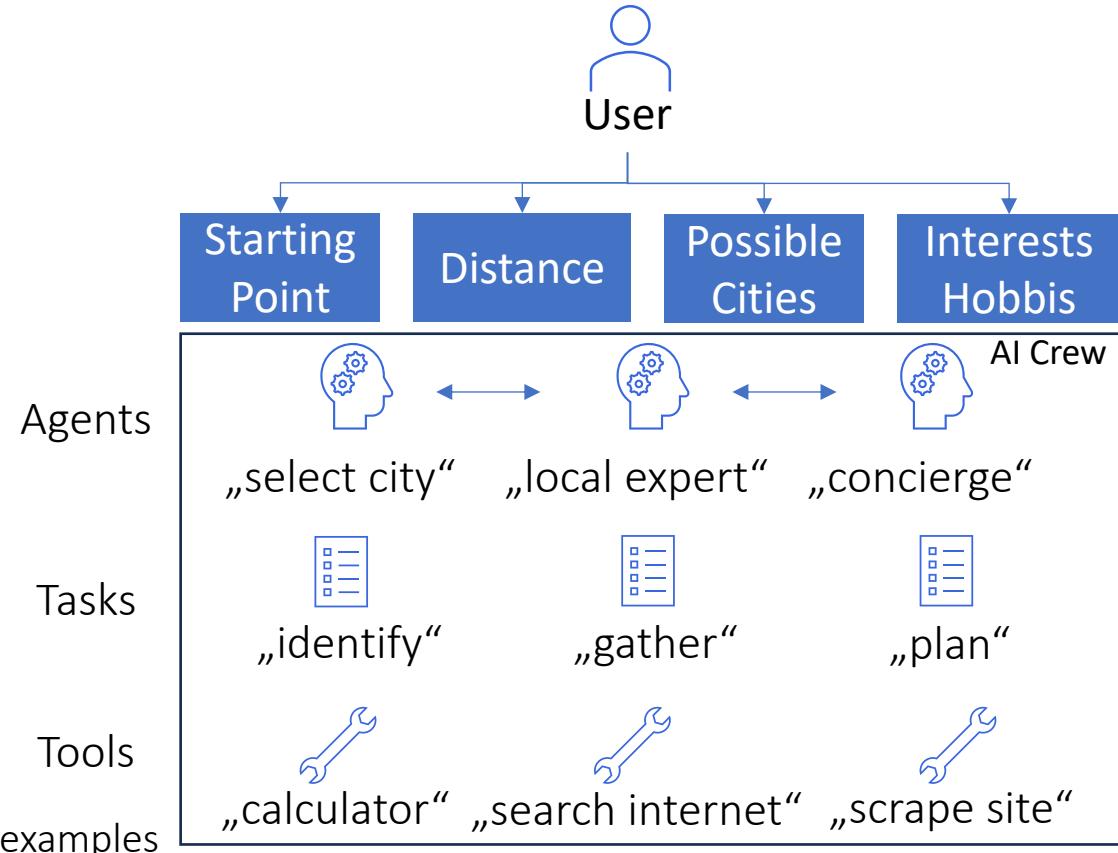
- concept extremely powerful when multiple agents are bundled
- system of experts



# crewAI

Example – Plan your vacation

- 1. Define Goal
- 2. User Inputs
- 3. Set up
  - agents
  - tasks
  - if needed:
    - tools
    - process
    - ...

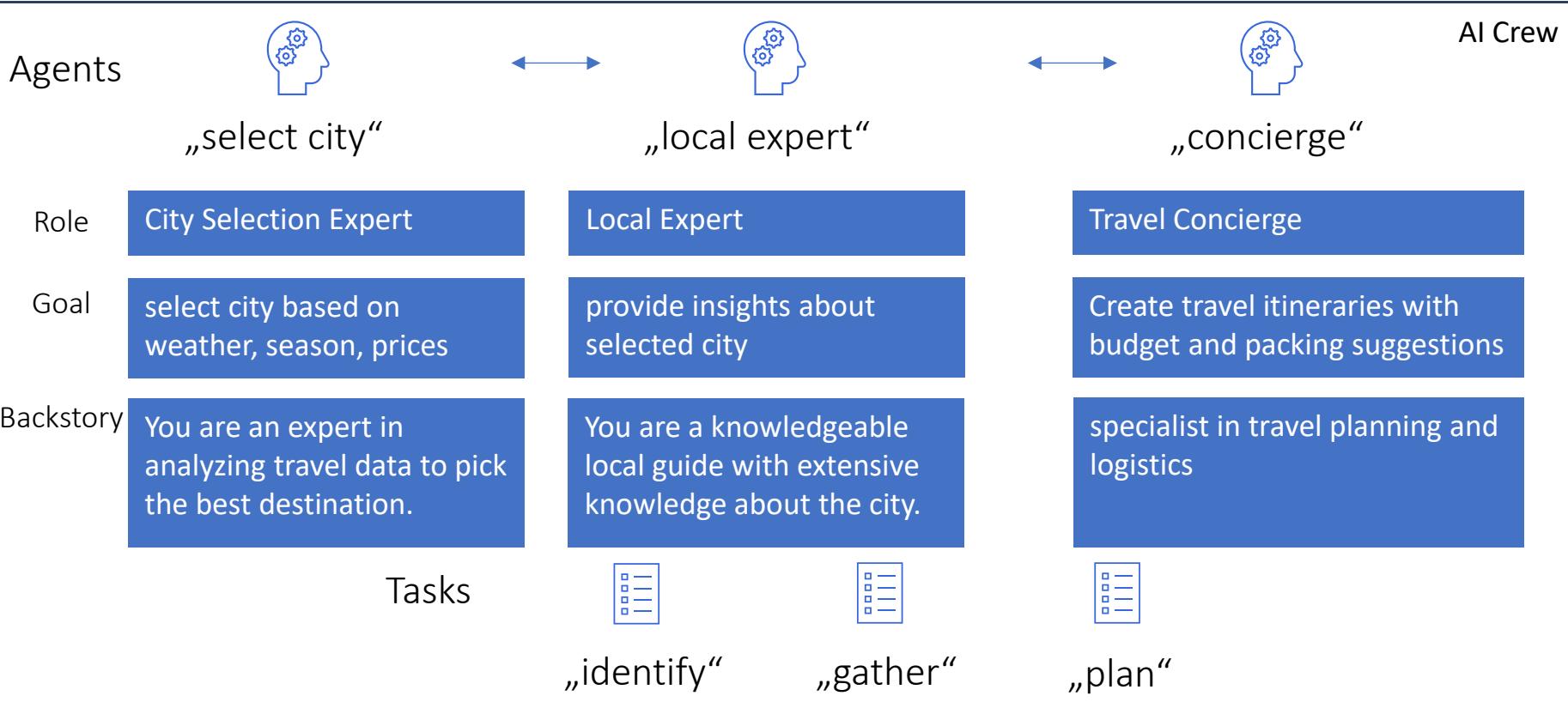


Idea found at

<https://github.com/joaomdmoura/crewAI-examples>

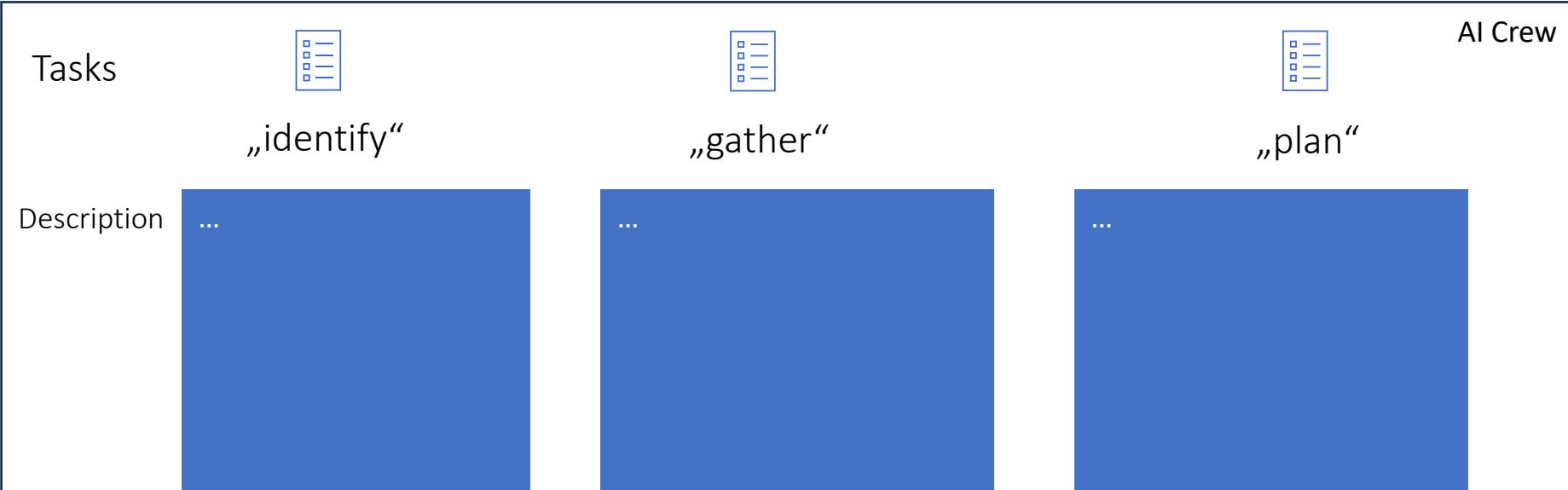
# crewAI

Example – Plan your vacation: Agents



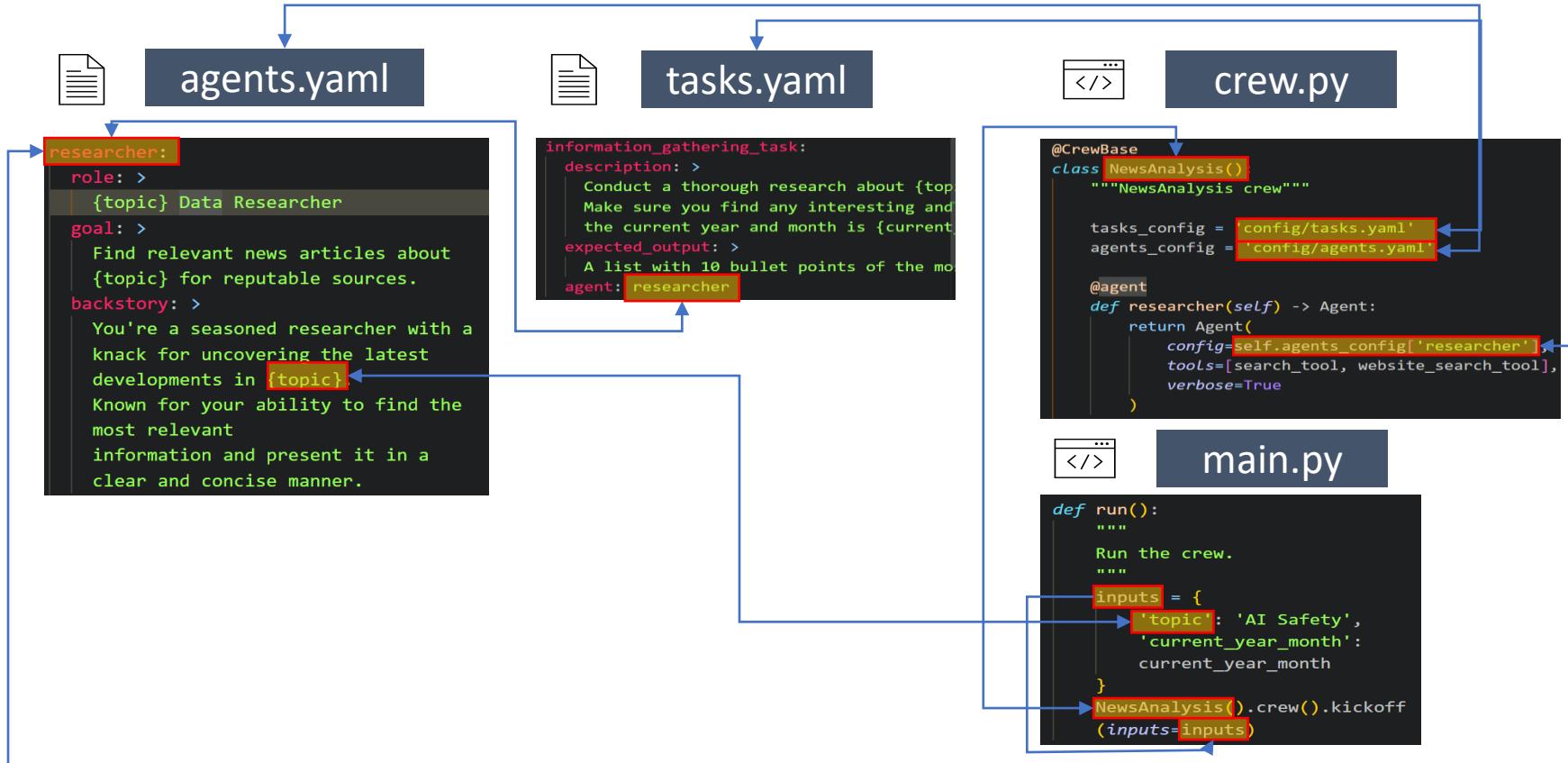
# crewAI

Example – Plan your vacation: Tasks



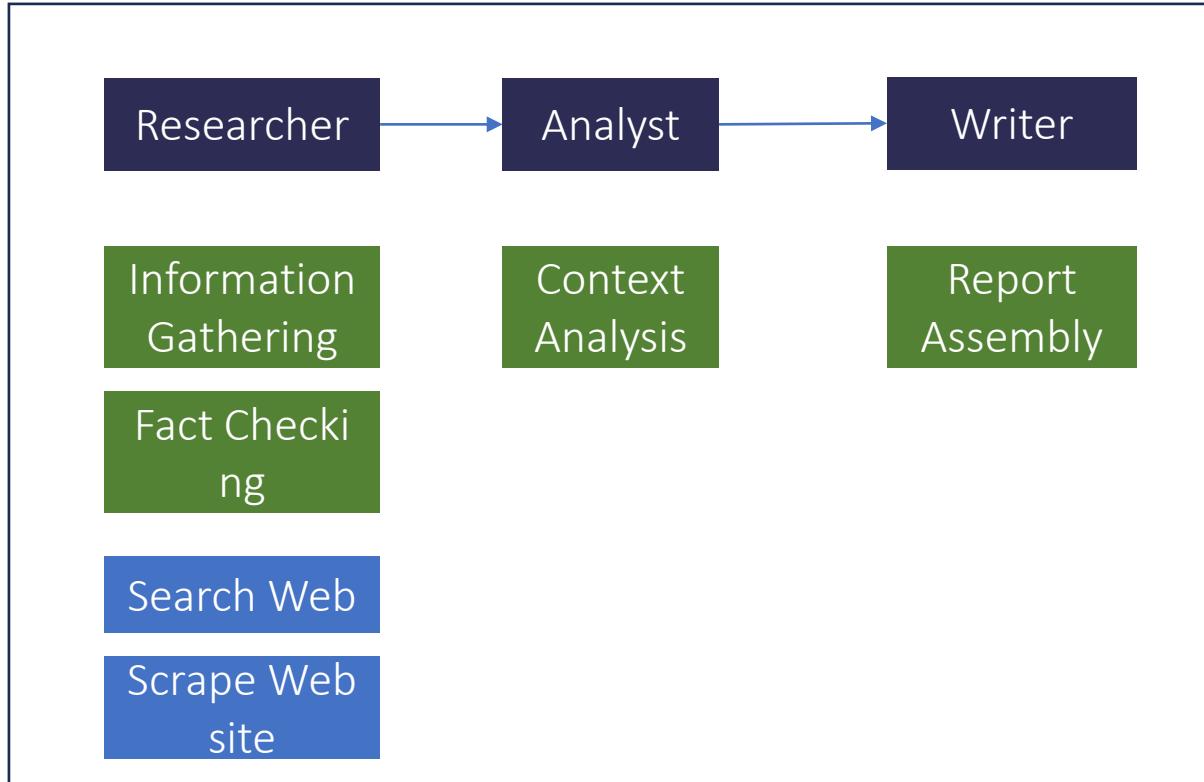
# crewAI

## File Dependencies



# crewAI

## Example – Crew Setup



Legend

Agent

Task

Tool

# crewAI

## Tools

- can be used by Agents for



Searching the Internet

Scraping Websites



Reading Files

Tool	Description
CodeDocsSearchTool	A RAG tool optimized for searching through code documentation and related technical documents.
CSVSearchTool	A RAG tool designed for searching within CSV files, tailored to handle structured data.
DirectorySearchTool	A RAG tool for searching within directories, useful for navigating through file systems.
DOCXSearchTool	A RAG tool aimed at searching within DOCX documents, ideal for processing Word files.
DirectoryReadTool	Facilitates reading and processing of directory structures and their contents.
FileReadTool	Enables reading and extracting data from files, supporting various file formats.
GithubSearchTool	A RAG tool for searching within GitHub repositories, useful for code and documentation search.
SerpentDevTool	A specialized tool for development purposes, with specific functionalities under development.
TXTSearchTool	A RAG tool focused on searching within text (.txt) files, suitable for unstructured data.

...

Source: <https://docs.crewai.com/core-concepts/Tools/#available-crewai-tools>

# crewAI

## Collaboration

Agents can collaborate on a task to

- share information
- assist on a task
- allocate and optimize resources

Process.sequential

Process.hierarchical



```
from crewai import Agent, Task, Crew,  
Process  
  
crew = Crew(  
    agents=[planner, writer, editor  
],  
    tasks=[plan, write, edit],  
    verbose=2,  
    manager_llm=llm,  
    process= Process.hierarchical  
)
```

# crewAI

## Expected Task Outcome

- output formats can be defined in detail

```
class OutputFormat(BaseModel):  
    chapter_title: str  
    bullet_points: list[str]  
  
Task(  
    description=(“...”),  
    expected_output="A well-written slideset ...“,  
    agent=editor,  
    output_format="markdown",  
    output_format_model=OutputFormat,  
    output_format_description=(  
        "The output format is a markdown file ..."  
    ),  
    output_file = "slideset.md"  
)
```

# crewAI

Use of other LLMs

- set up an llm-object
- pass it as a parameter

```
from langchain_groq import ChatGroq
llm=ChatGroq(temperature=0,
              model_name=MODEL,
              api_key=os.environ["GR
OQ_API_KEY"])
planner = Agent(
    role="...",
    goal="...",
    backstory="...",
    allow_delegation=False,
    Llm=llm,
    verbose=True
)
```

# crewAI

## Installation

- Problem:
  - crewAI is a huge framework with many dependencies
  - Does currently have issues with Python 3.13



```
C:\Temp\crewai>where python
C:\Python313\python.exe
C:\Users\BertGollnick\AppData\Local\Programs\Python\Python312\python.exe
C:\Users\BertGollnick\AppData\Local\Microsoft\WindowsApps\python.exe
```

- Start from totally clean environment:

```
C:\Temp\crewai>C:\Users\BertGollnick\AppData\Local\Programs\Python\Python312\python.exe -m venv .venv
```

```
C:\Temp\crewai>.venv\Scripts\activate
```

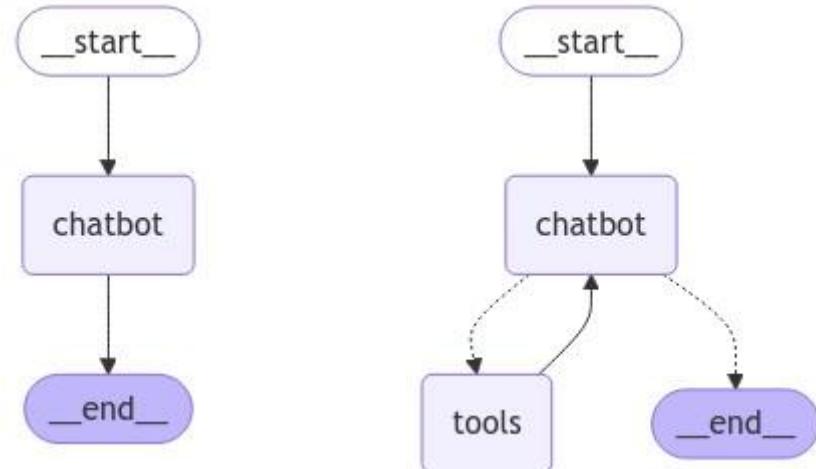
```
(.venv) C:\Temp\crewai>pip install crewai[tools]
```

# LangGraph

# LangGraph

## Introduction

- agentic system
- graph-based representation (directed acyclical graph DAG)
- integrates well with LangChain ecosystem
- focuses on complex workflows
- based on nodes (tasks), and edges (dependencies)



Typical LangGraph graphs

# LangGraph

## State

- Shared data structure
- Nodes read from and write to
- TypedDict...for type hinting
- Annotated...new messages are appended to existing list

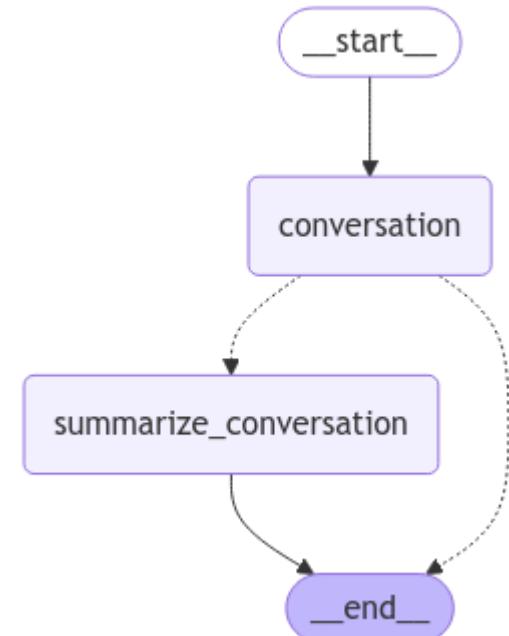
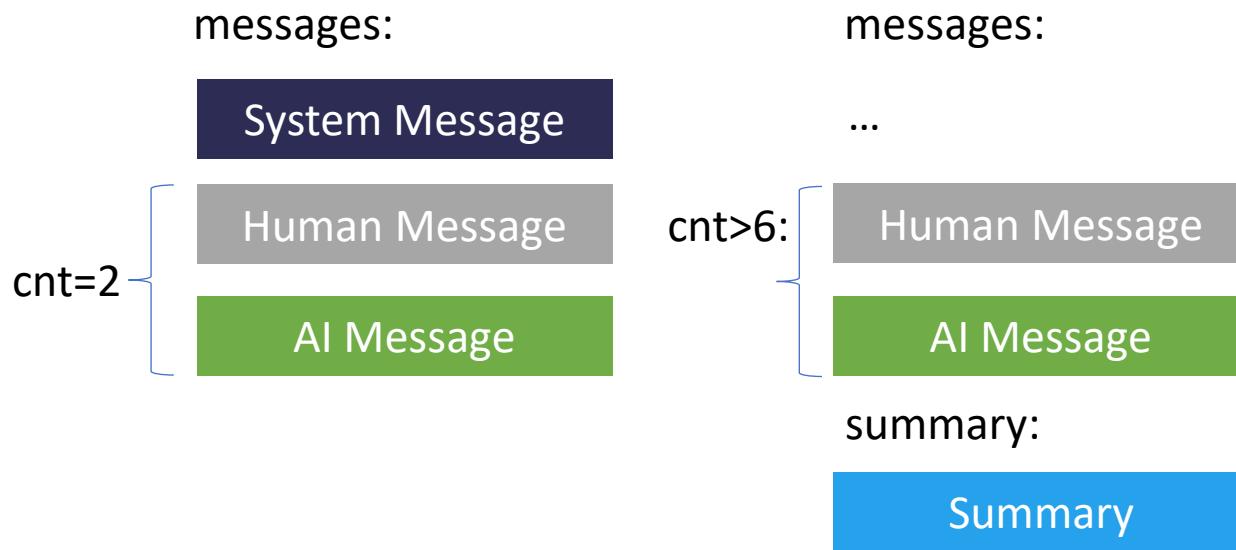
```
class ConversationState(TypedDict):
    """
    Represents the state of our conversation graph.

    Attributes:
        messages: A list of BaseMessage objects, annotated to
                  append new messages.
        summary: An optional string to store the conversation
                 summary.
    """

    messages: Annotated[List[BaseMessage], operator.add]
    summary: str = ""
```

# LangGraph

Chat with Summarization

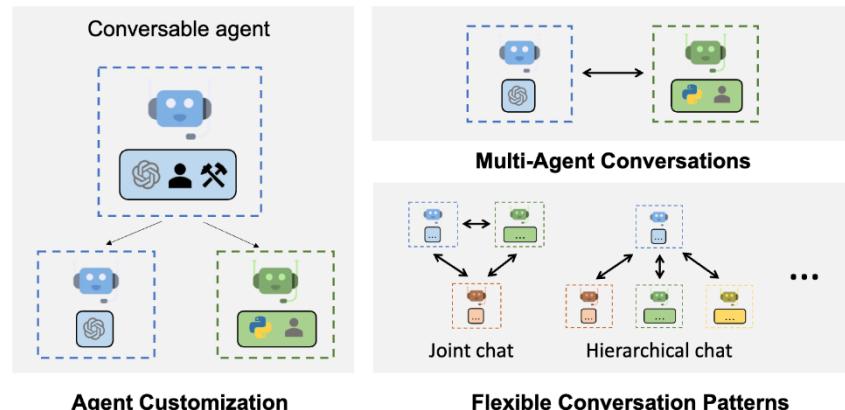


AG2 (formerly autogen)

# AG2

## Introduction

- Open-Source framework for building AI agents
- Installation: pip install ag2
- Docker: optional



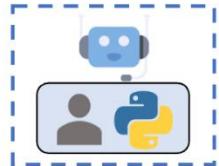
Source: <https://github.com/ag2ai/ag2>

# AG2

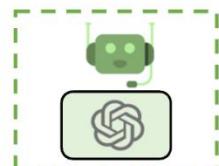
## Introduction

Uses shell with  
human-in-the-loop

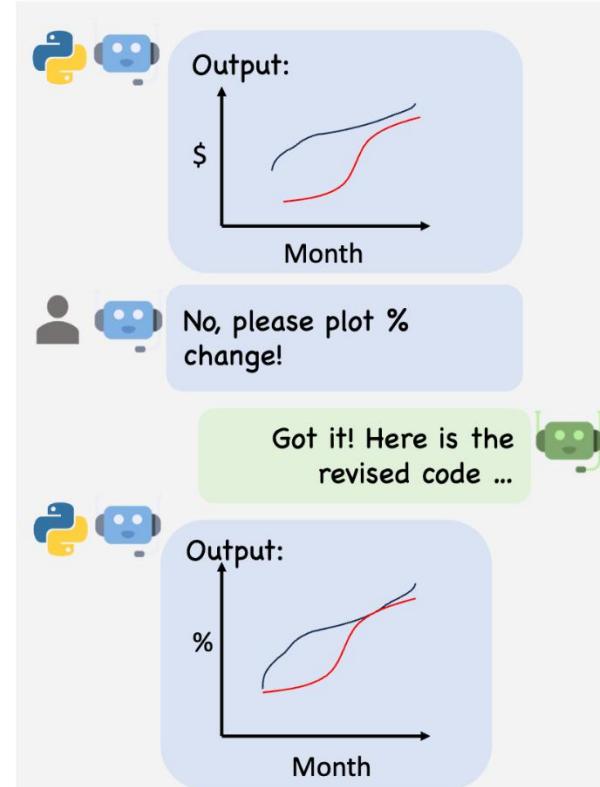
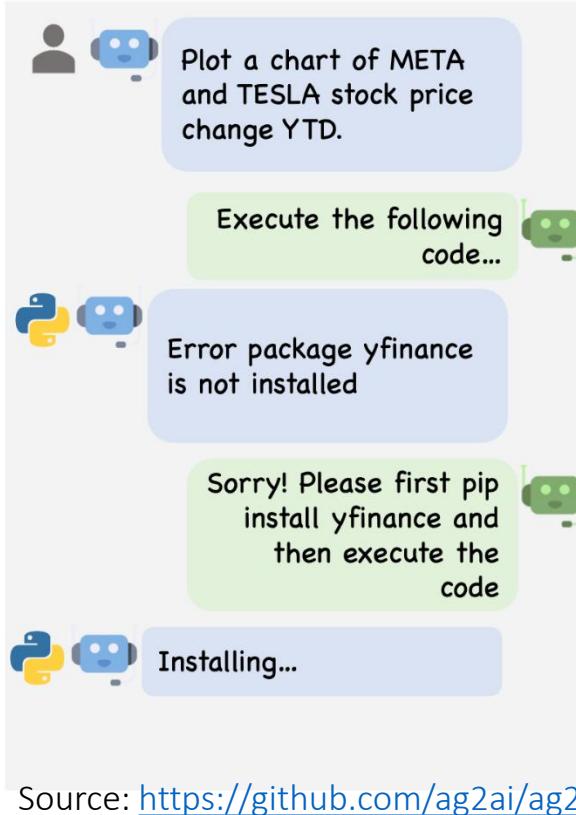
User Proxy Agent



Assistant Agent



LLM configured to  
write python code

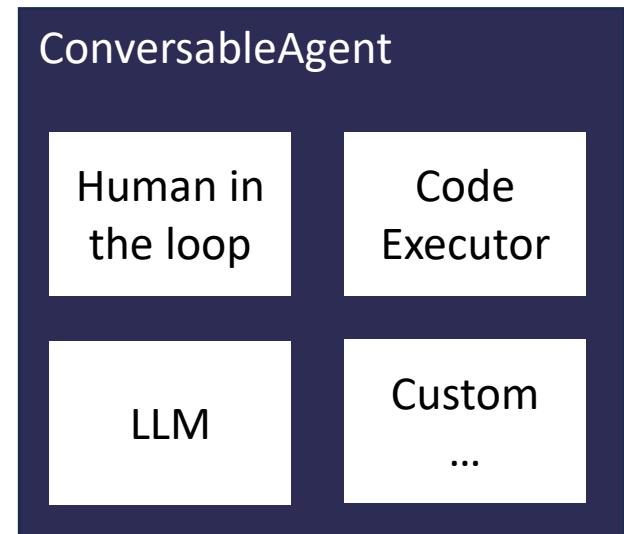


Source: <https://github.com/ag2ai/ag2>

# AG2

## Agent

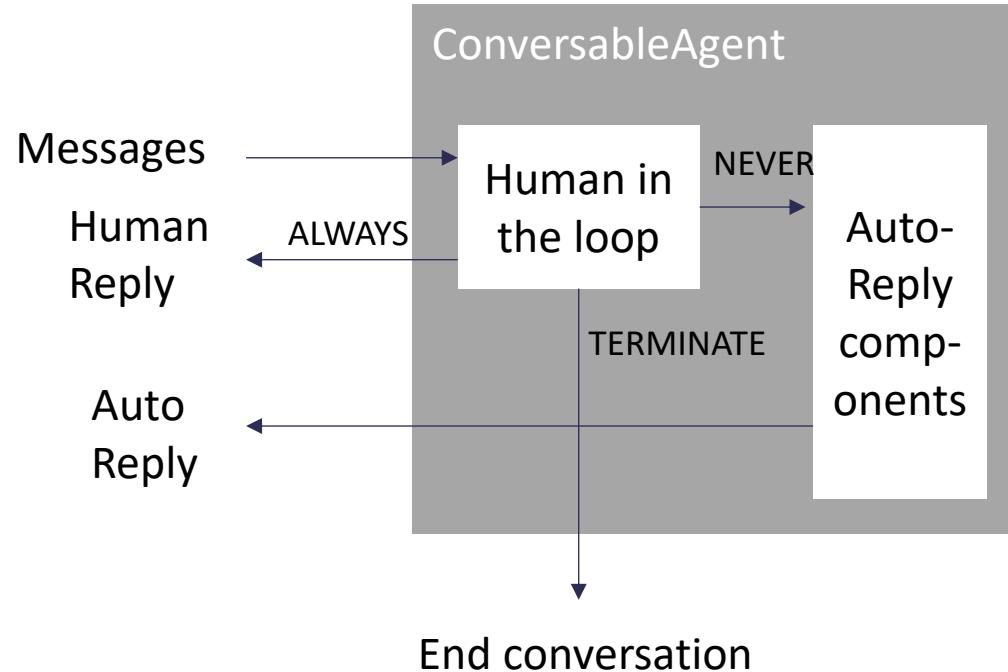
- AG2 agent
  - entity that can send and receive messages to and from other agents
  - agent can be run by
    - models,
    - code executors,
    - human, or
    - a combination of above



# AG2

## Human in the loop

- HITL in front of Auto reply, intercepts incoming messages, and decides to pass to auto-reply, or to human feedback
- customizable through *human\_input\_mode* parameter
- modes:
  - NEVER, TERMINATE (default), ALWAYS



# AG2

## Tools

- agents can use tools
- register\_for\_llm
  - exposes tool to LLM
  - allows LLM to reason about tool, decide when to call
- register\_for\_execution
  - handles execution of tool when LLM decides to call it
  - connects logical request generated by LLM to process
  - without it, even if LLM decides to use a tool, there would be no backend to execute tool's functionality

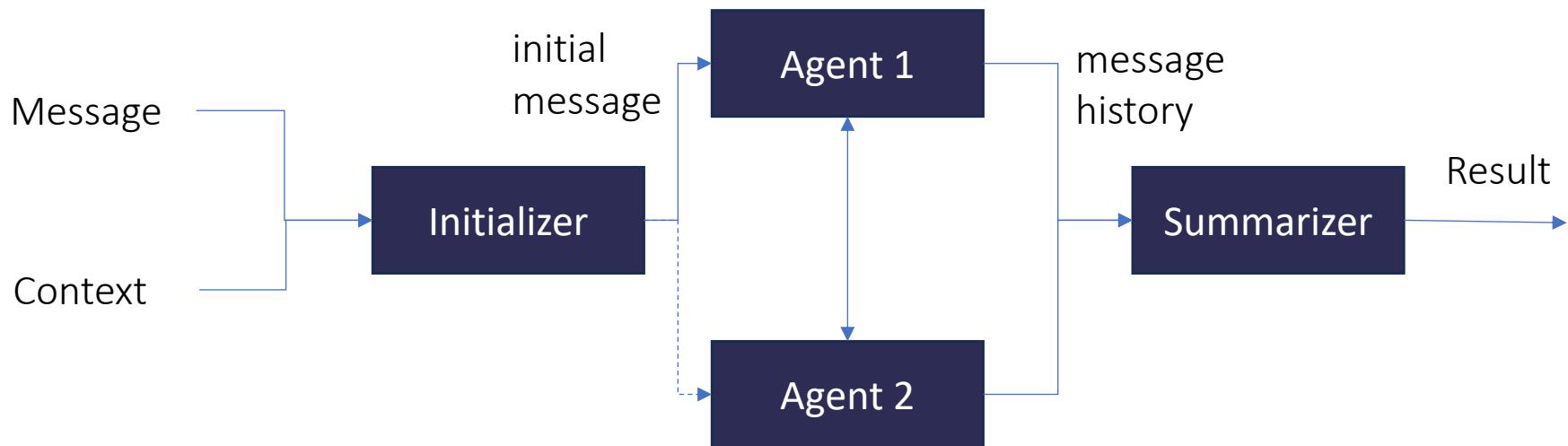
```
# %% create an agent with a tool
my_assistant = ConversableAgent(
    name="my_assistant",
    system_message="You are a helpful AI assistant.",
    llm_config=config_list
)

# register the tool signature at agent level
my_assistant.register_for_llm(
    name="get_current_date",
    description="Returns the current date in the form
at YYYY-MM-DD."
)(get_current_date)

# register the tool function at execution level
my_assistant.register_for_execution(name="get_current
_date")(get_current_date)
```

# AG2

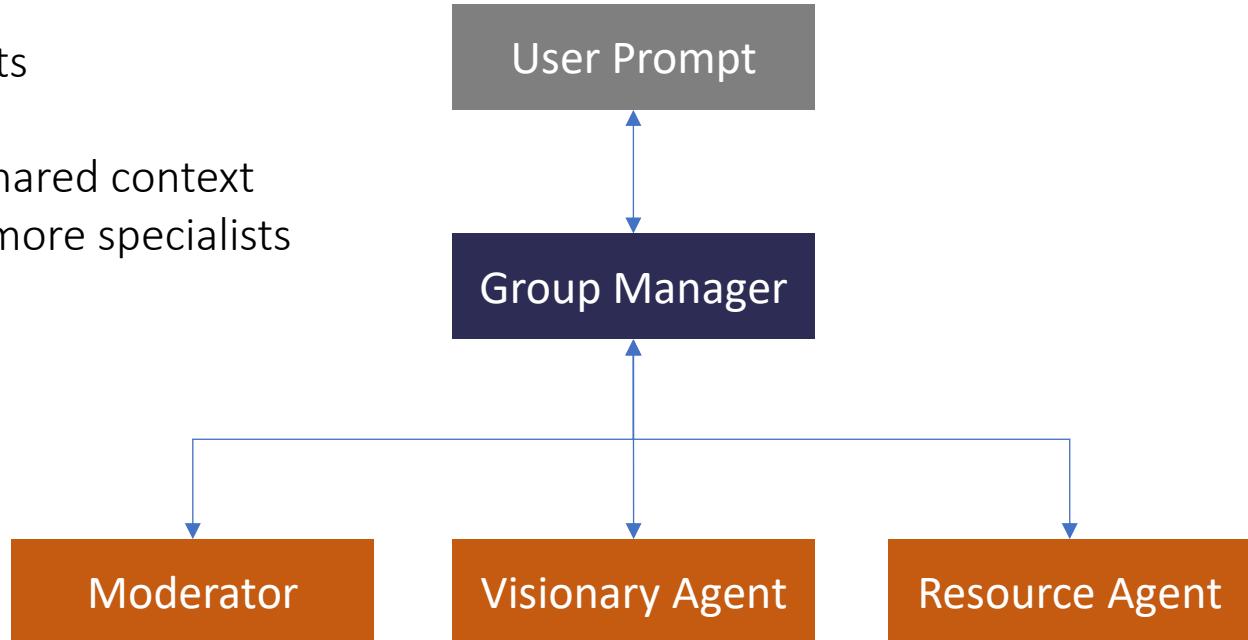
## Conversation Patterns: Two Agents Chatting



# AG2

## Group Chat

- Multiple agents interact
- Allows a team of specialists
- Dynamic collaboration
- All agents use the same shared context
- Easily scalable by adding more specialists



# OpenAI Agents SDK

# OpenAI Agents SDK

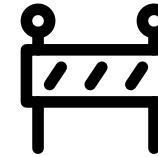
## Introduction



Agent



Handoffs

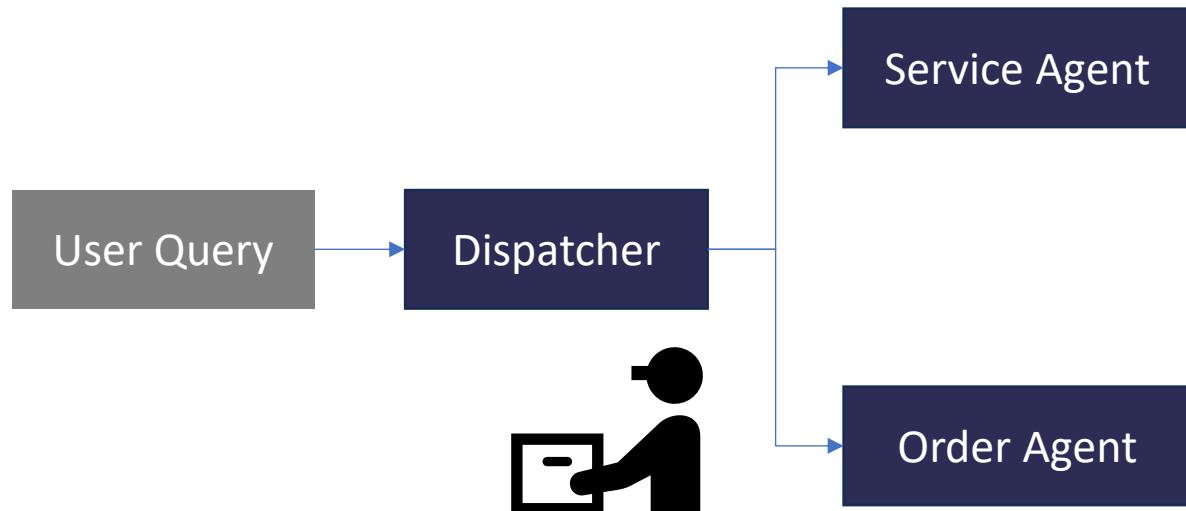


Guardrails

- LLM
- Instructions
- tools
- Agents delegate tasks to other agents
- Enables input validation

# OpenAI Agents SDK

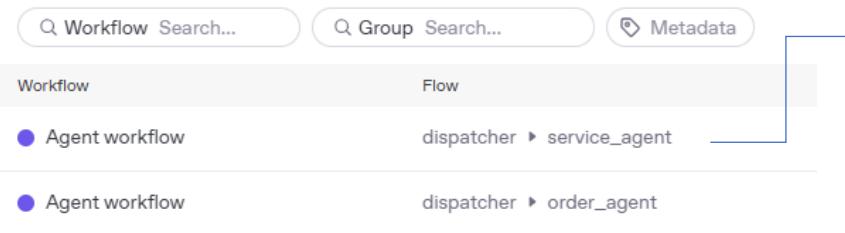
## Handoff Implementation



# OpenAI Agents SDK

## Traces

### Traces



Traces / Agent workflow trace\_b000faa931714b...

dispatcher  
POST /v1/responses  
Handoff ➔ service\_agent

service\_agent  
POST /v1/responses

**Span details**

Properties  
Created Jul 7, 2025, 8:04 AM

ID resp\_686b6379e518819abf16... ▾

Model gpt-4o-2024-08-06

Tokens 114 total

Functions transfer\_to\_order\_agent()  
transfer\_to\_service\_agent()

Configuration

Response text

**Instructions**

System Instructions  
You are a dispatcher. You determine which agent should handle the user's r

**Input** 101t ▾  
User  
I ordered a while ago, but I haven't received my pizza yet. What's the status?

**Output** 13t ▾  
Function Call

This screenshot shows a detailed view of a trace for an 'Agent workflow'. The trace starts with a 'dispatcher' sending a 'POST /v1/responses' request. This is followed by a 'Handoff' event where the request is passed to a 'service\_agent'. The 'service\_agent' then sends its own 'POST /v1/responses' request. The right side of the screen displays the 'Span details' for this trace, including properties like ID and creation time, tokens processed, and functions called. It also shows the configuration, response, system instructions, input from the user, and the final output. The input message from the user is: 'I ordered a while ago, but I haven't received my pizza yet. What's the status?'

# OpenAI Agents SDK

## Running Options

`Runner.run()`

- Asynchronous
- Returns RunResult

`Runner.run_sync()`

- Synchronous
- Returns RunResult

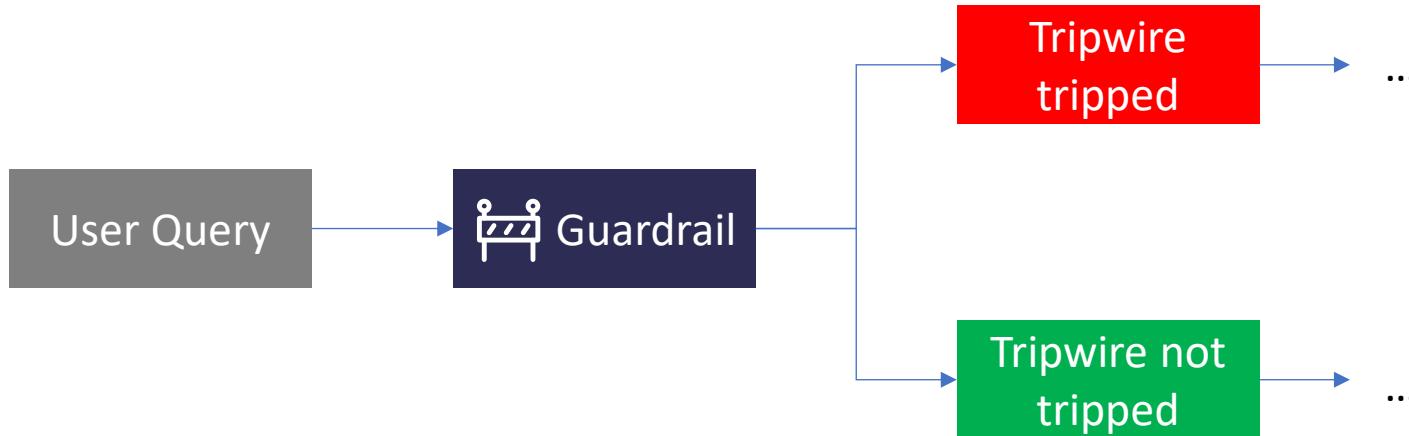
`Runner.run_streamed()`

- asynchronous
- Returns RunResultStreaming

# OpenAI Agents SDK

## Guardrails

- Run parallel to agents
- Allow for input validation



# Google ADK

# Google ADK

Agent Development Kit (ADK)

- Framework for developing AND deploying AI agents
- model-agnostic
- deployment-agnostic
- Many pre-built tools
- Modular and scalable

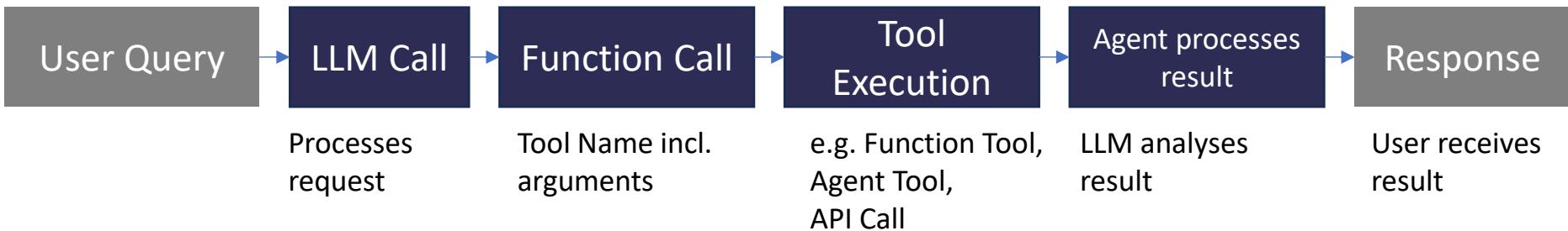


Agent Development Kit

# Google ADK

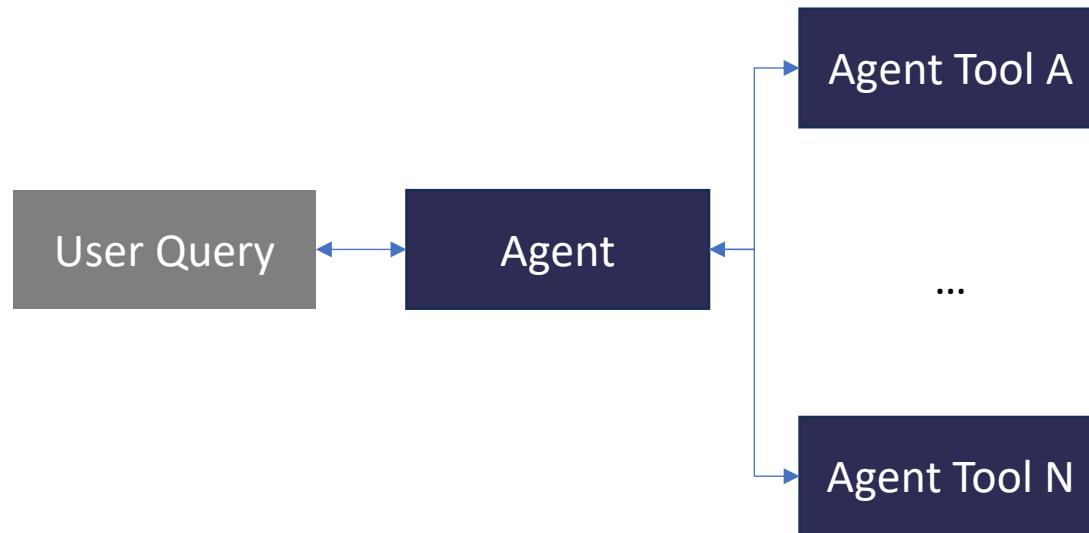
## Tool Use

- tool is a feature provided to agent
- Enables to perform actions and interact with the world



# Google ADK

## Agent Tool Use



# Google ADK

Run Agent



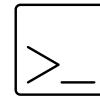
parent\_folder



navigate here



agent\_project



adk run agent\_project



\_\_init\_\_.py



adk web



.env.py



agent.py