Agents

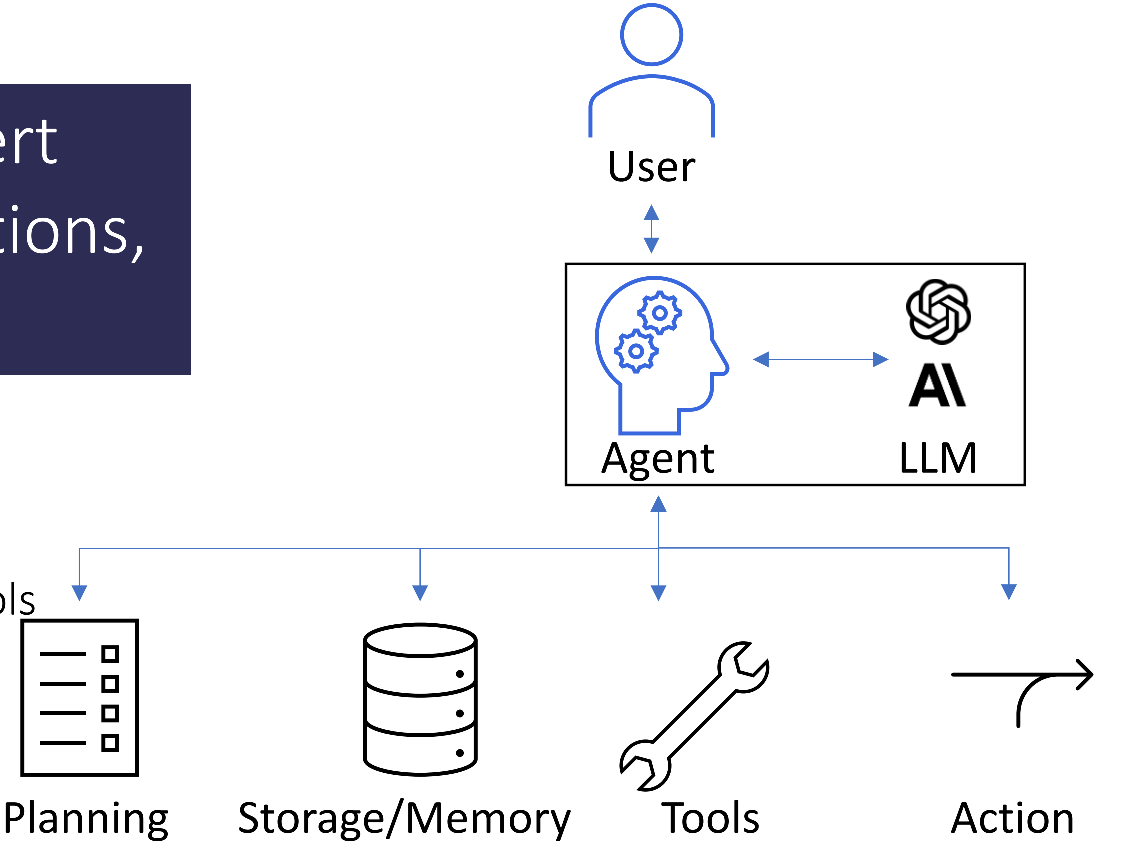
Introduction

Agents

What is an Agent?

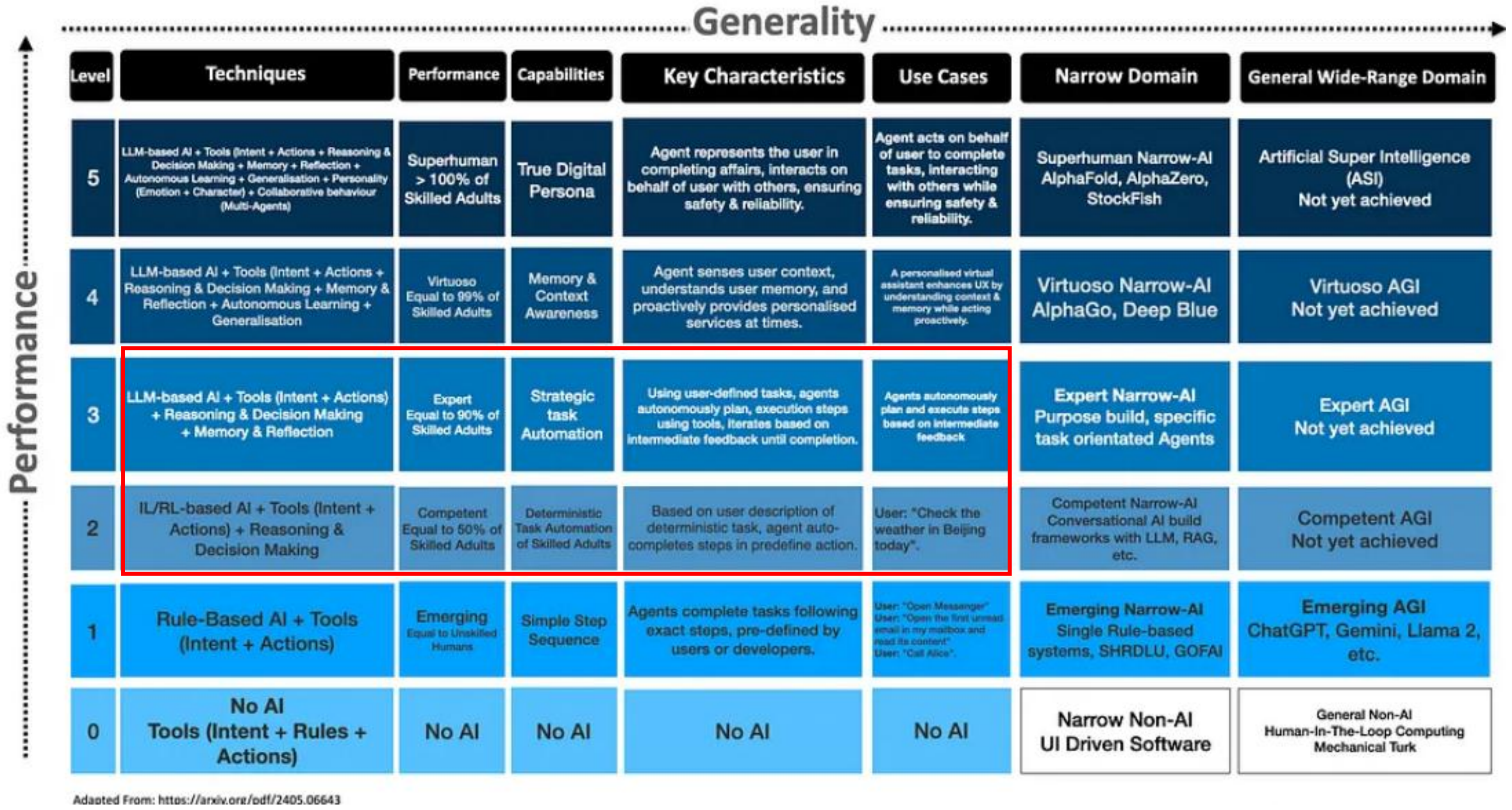
An AI Agent is an expert that can answer questions, and help with tasks.

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



Agents

Levels of AI Agents



Generality
Performance

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

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

Available Frameworks

Agents

Which Frameworks are available?

LangGraph

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

- not intuitive for non-programmers

CrewAI

- very intuitive
- suitable for many agents
- supports many LLM providers

- not ideal for very complex tasks

Swarm

- very easy to use
- suitable for beginners

- only supports OpenAI

Agents

Which Frameworks are available?

Microsoft

AG2 (formerly: AutoGen)

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

Microsoft

Magentic-One

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

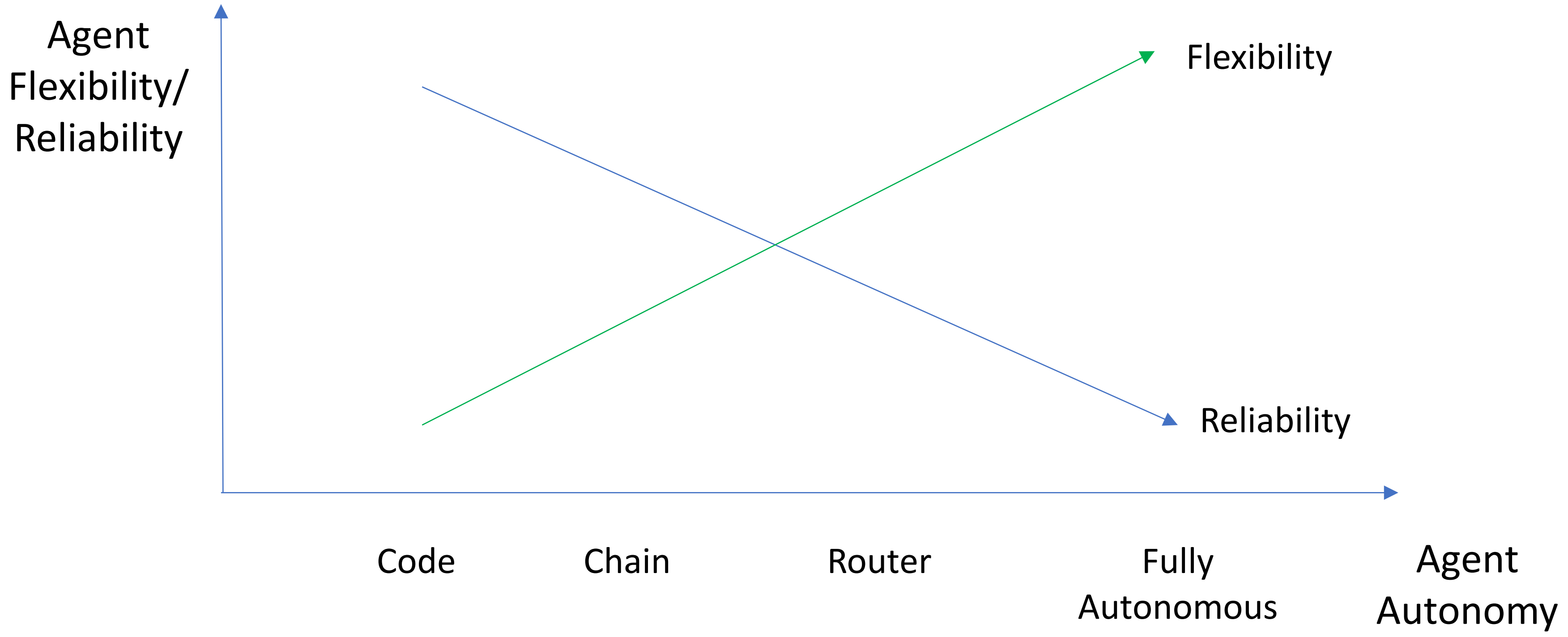
Microsoft

tinytroupe

- multiagent persona simulation for imagination enhancement and business insights
- only works with GPT-4o
- Link

Agents

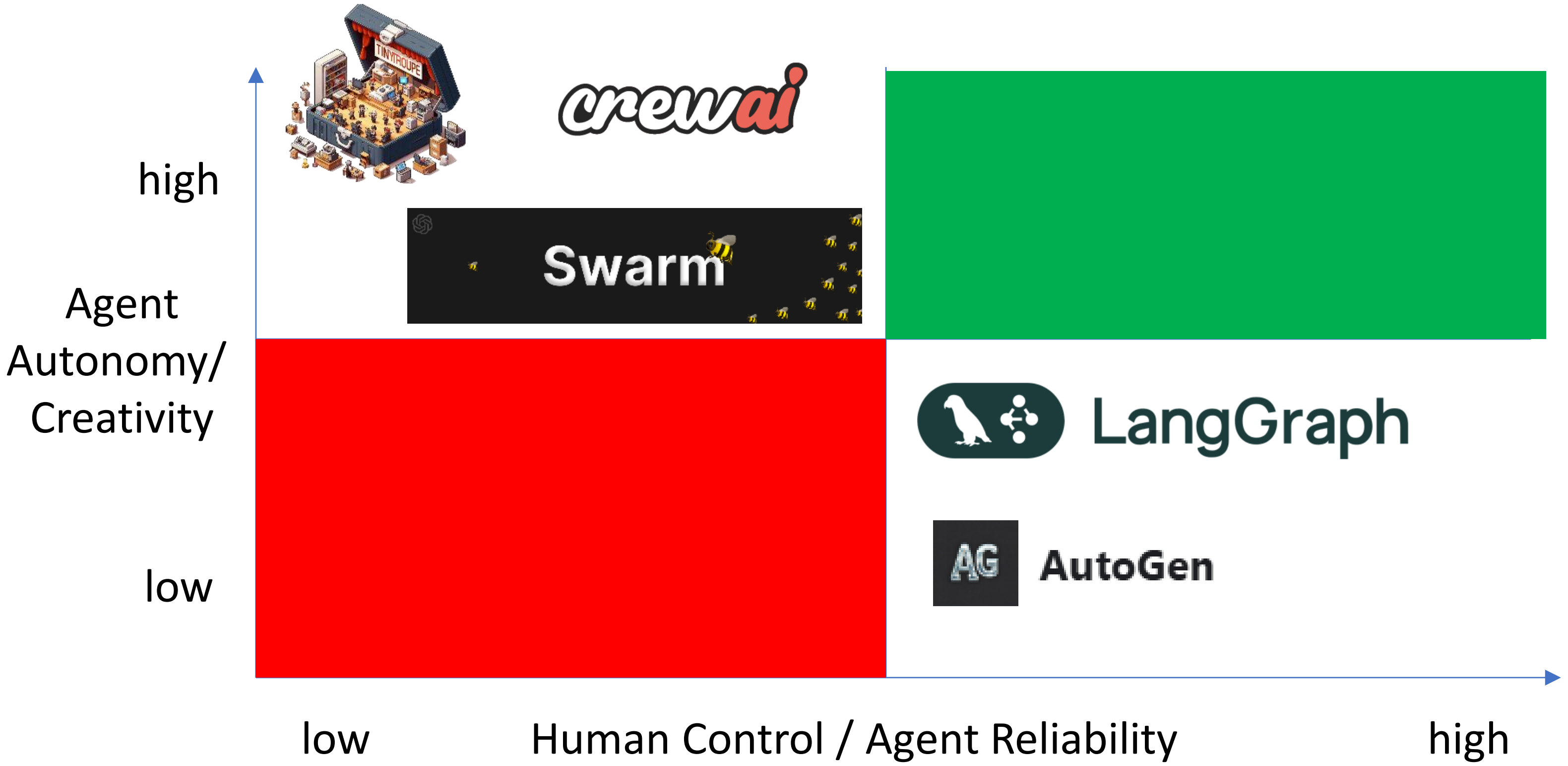
Flexibility / Reliability vs. Autonomy



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

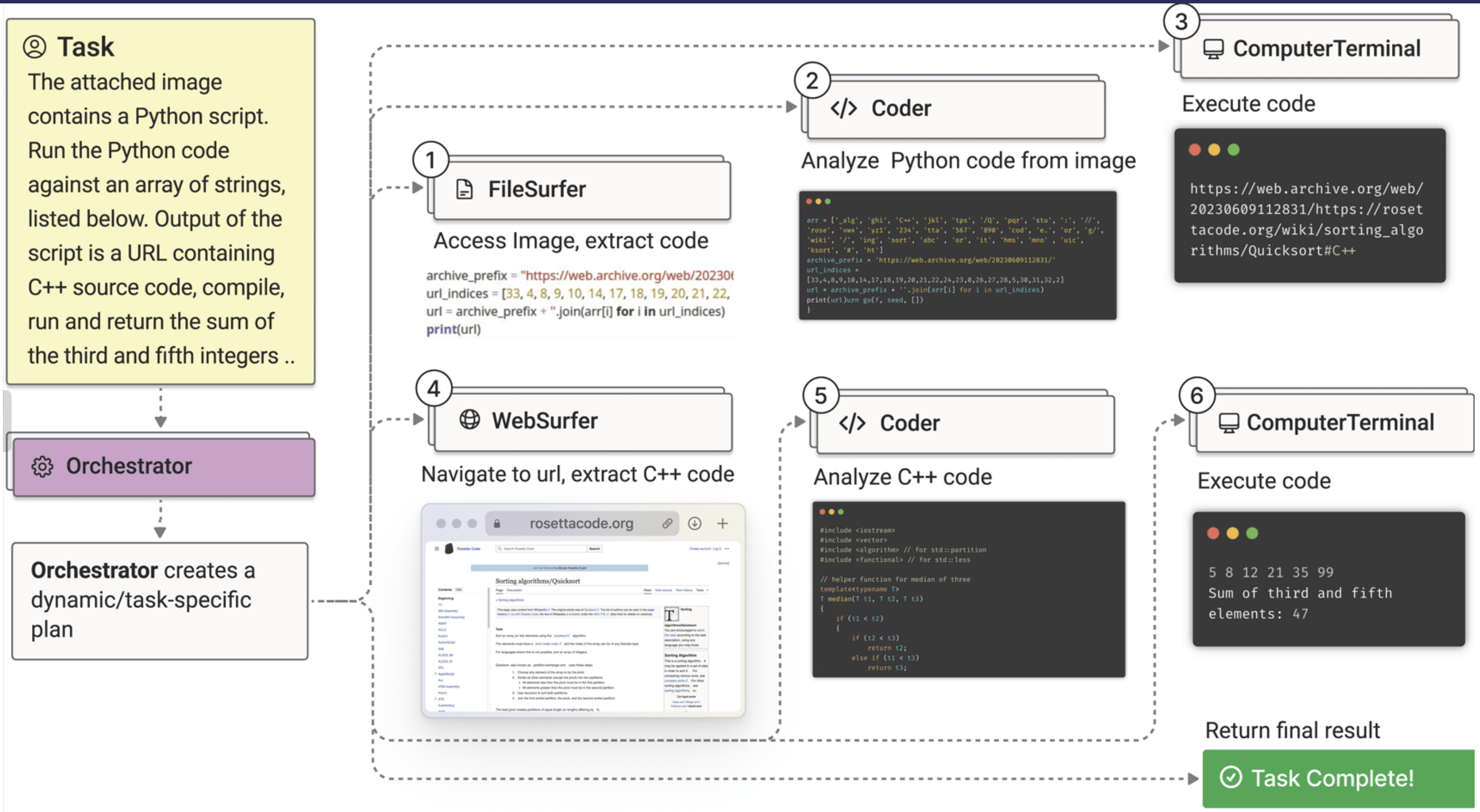
Agents

Flexibility vs.



Agents

Magentic One



Task

The attached image contains a Python script. Run the Python code against an array of strings, listed below. Output of the script is a URL containing C++ source code, compile, run and return the sum of the third and fifth integers ..

Orchestrator creates a dynamic/task-specific plan

```
archive_prefix = "https://web.archive.org/web/20230
url_indices = [33, 4, 8, 9, 10, 14, 17, 18, 19, 20, 21, 22,
url = archive_prefix + ''.join(arr[i] for i in url_indices)
print(url)
```

```
arr = ['_alg', 'ghi', 'C++', 'jkl', 'tps', '/Q', 'pqr', 'stu', ':', '//',
'rose', 'vwx', 'yz1', '234', 'tta', '567', '890', 'cod', 'e.', 'or', 'g/',
'wiki', '/', 'ing', 'sort', 'abc' , 'or', 'it', 'hms', 'mno' , 'uic',
'ksort', '#', 'ht']
archive_prefix = 'https://web.archive.org/web/20230609112831/'
url_indices =
[33,4,8,9,10,14,17,18,19,20,21,22,24,23,0,26,27,28,5,30,31,32,2]
url = archive_prefix + ''.join(arr[i] for i in url_indices)
print(url)urn go(f, seed, [])
}
```

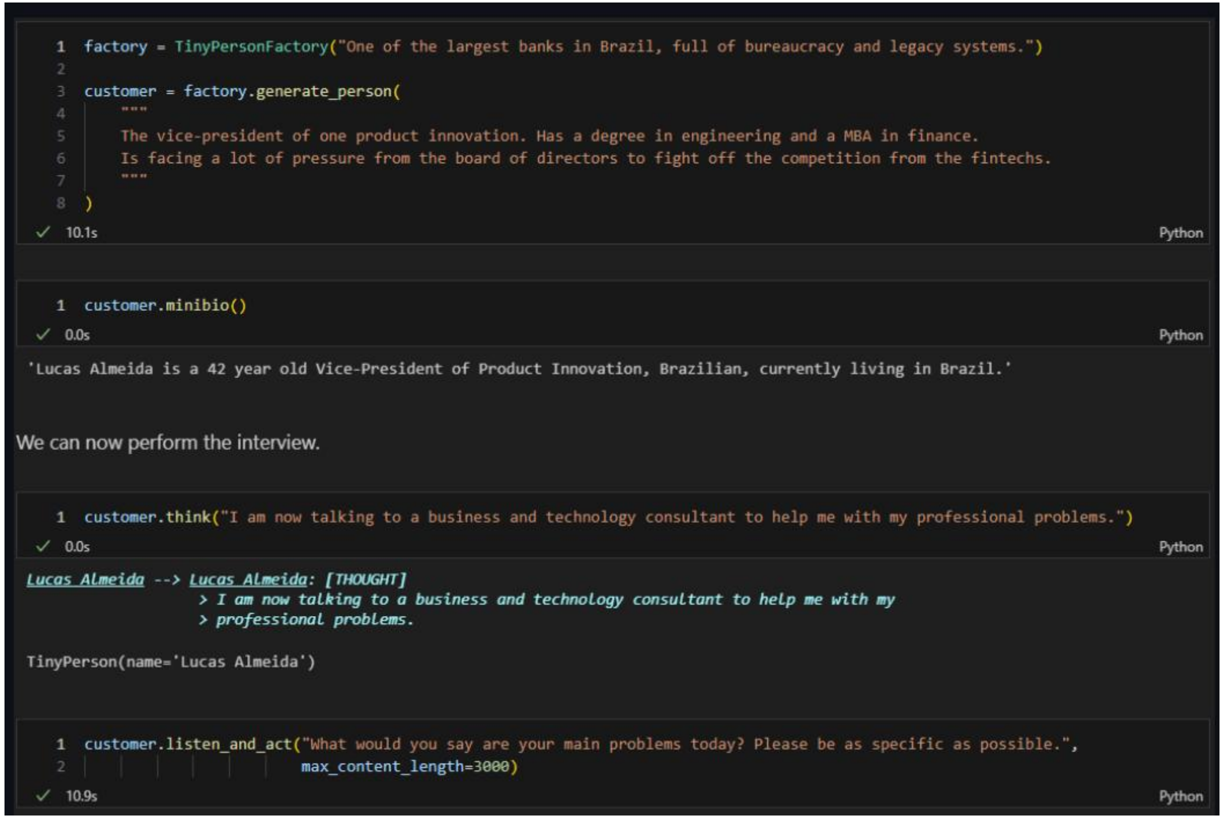
```
https://web.archive.org/web/
20230609112831/https://roset
tacode.org/wiki/sorting_algo
rithms/Quicksort#C++
```

```
#include <iostream>
#include <vector>
#include <algorithm> // for std::partition
#include <functional> // for std::less

// helper function for median of three
template<typename T>
T median(T t1, T t2, T t3)
{
    if (t1 < t2)
    {
        if (t2 < t3)
            return t2;
        else if (t1 < t3)
            return t3;
```

```
5 8 12 21 35 99
Sum of third and fifth
elements: 47
```

Agents

Tine Troupe



```
factory = TinyPersonFactory("One of the largest banks in Brazil, full of bureaucracy and legacy systems.")

customer = factory.generate_person(
    """
    The vice-president of one product innovation. Has a degree in engineering and a MBA in finance.
    Is facing a lot of pressure from the board of directors to fight off the competition from the fintechs.
    """
)
```
✓ 10.1s Python

```
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.

```
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')
```

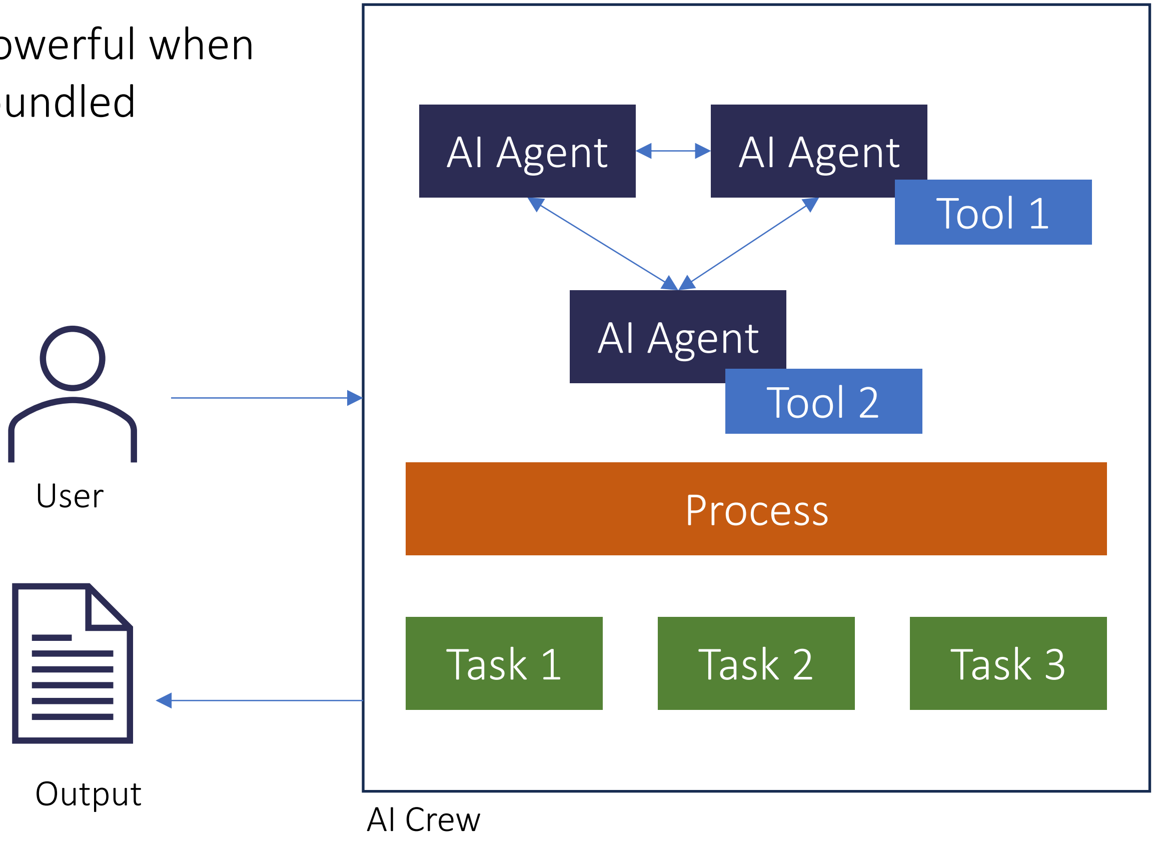
```
customer.listen_and_act("What would you say are your main problems today? Please be as specific as possible.",
                        max_content_length=3000)
```
✓ 10.9s Python

crewAI

crewAI

What is an AI Crew?

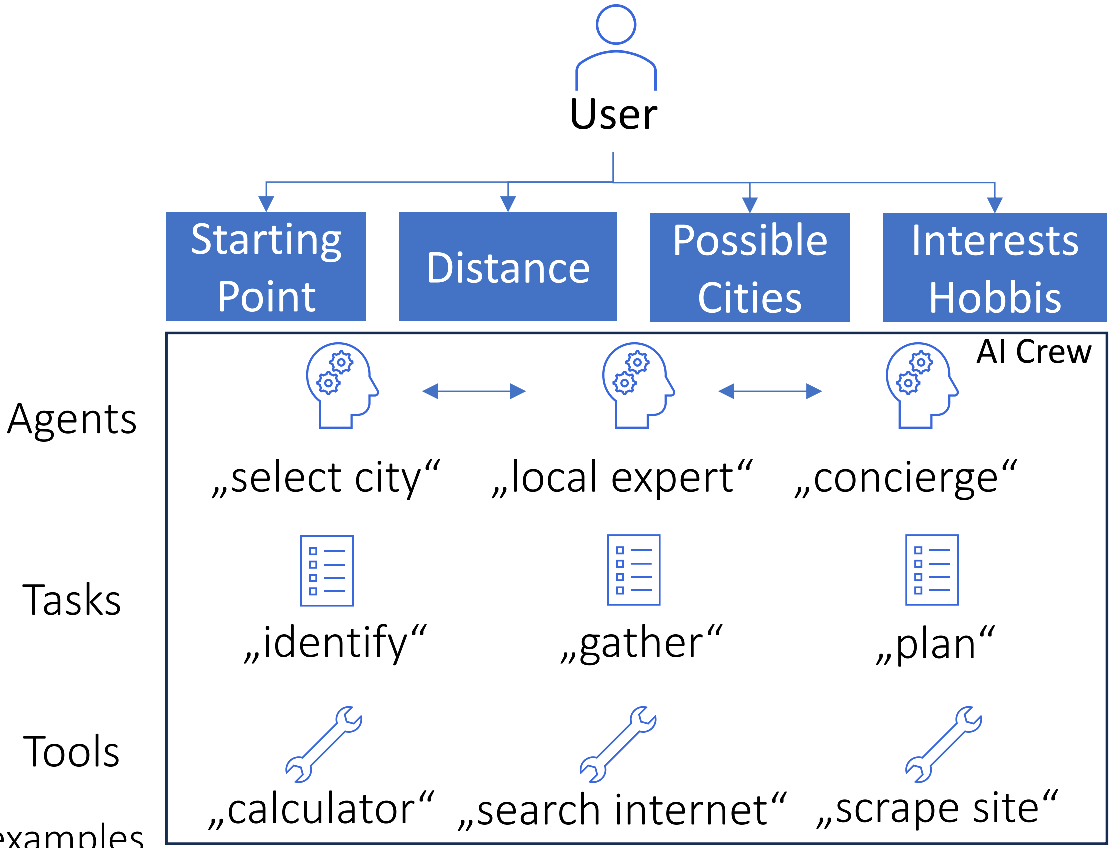
- concept extremely powerful when multiple agents are bundled
- system of experts



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

AI Crew

Agents

	„select city“	„local expert“	„concierge“
Role	City Selection Expert	Local Expert	Travel Concierge
Goal	select city based on weather, season, prices	provide insights about selected city	Create travel itineraries with budget and packing suggestions
Backstory	You are an expert in analyzing travel data to pick the best destination.	You are a knowledgeable local guide with extensive knowledge about the city.	specialist in travel planning and logistics

Tasks

„identify“ „gather“ „plan“

crewAI

Example – Plan your vacation: Tasks

AI Crew

Tasks

„identify“

„gather“

„plan“

Description

...

...

...

crewAI

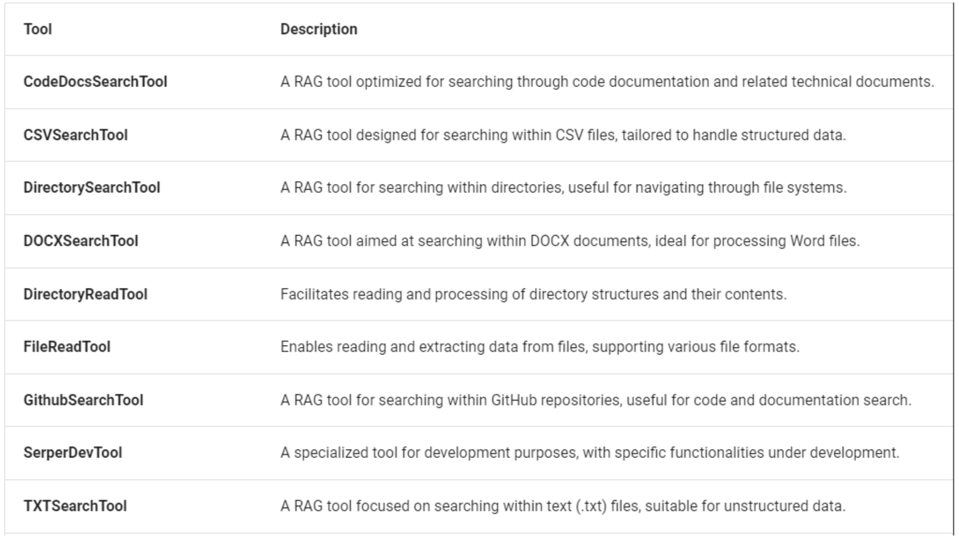
Tools

- 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.
SerperDevTool	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

Memory

- temporary storage of interactions
- enables agents to recall information to current context

Short-Term Memory

- preserves valuable insights and outcomes
- allows agents to build up knowledge over time

Long-Term Memory

- captures and organizes information on entities, e.g. people, places

Entity Memory

- keeps context of interactions
- increases relevance of agent responses

Contextual Memory

crewAI

Memory

- implementation is pretty simple
- by default
  - memory is disabled
  - uses OpenAI embeddings

```
from crewai import Crew, Agent,
Task, Process

my_crew = Crew(
    agents=[ … ],
    tasks=[ … ],
    process=Process.sequential,
    memory=True,
    verbose=True
)
```

crewAI

Memory - Benefits

Adaptive Learning

- crews adapt to new information and refine their approach to tasks

Enhanced Personalisation

- agents remember user preferences and historical interactions

Improved Performance

- more informed decisions
- use past learnings and contextual insights

crewAI

Asynchronous Operation

- s

crewAI

Callbacks

- task callback and step callback
- executed after task or step-completion
- can be used for
  - notifications
  - actions
- parameter passed inside Task

crewAI

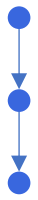
Collaboration

Agents can collaborate on a task to

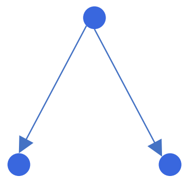
- share information
- assist on a task
- allocate and optimize resources

```python
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
)
```

Process.sequential



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

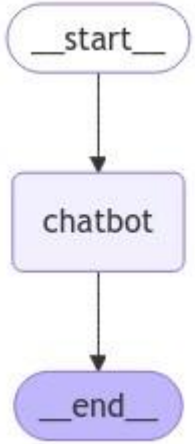
```
from langchain_groq import ChatGroq

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
)
```

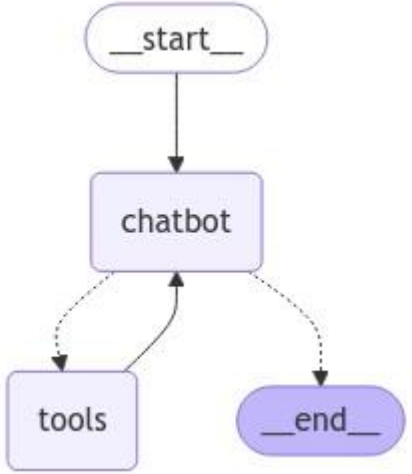
LangGraph

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)

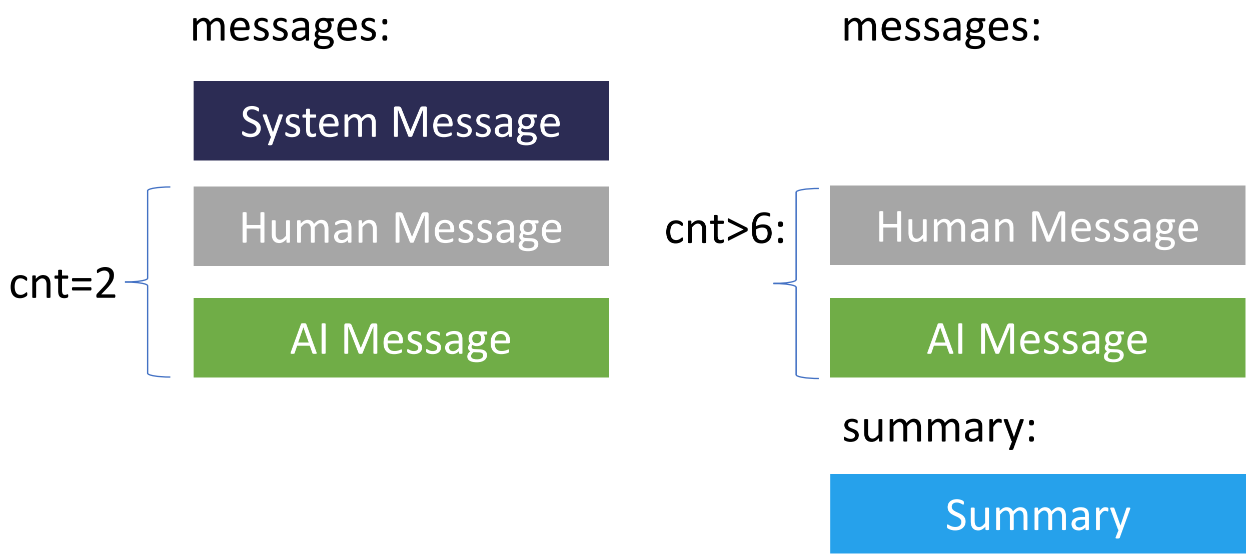


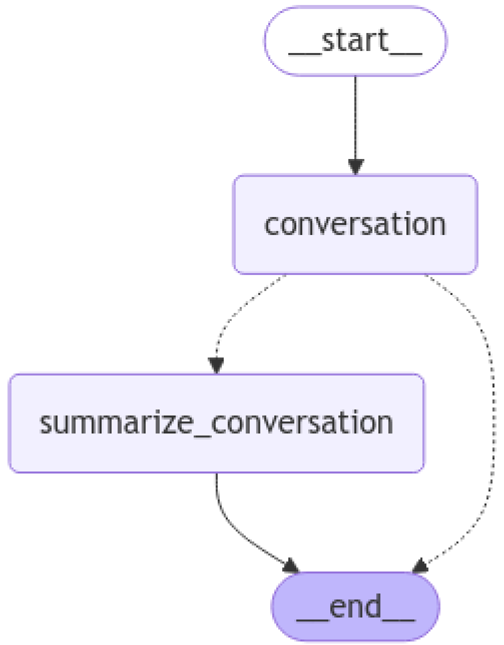


Typical LangGraph graphs

LangGraph

Chat with Summarization





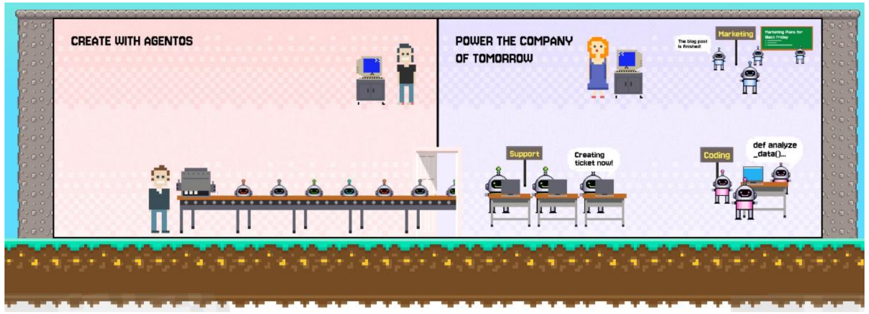
AG2 (formerly autogen)

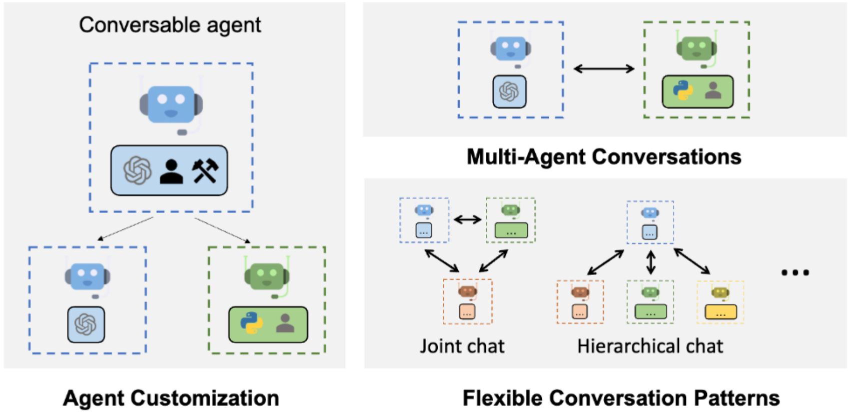
AG2

Introduction

- Open-Source framework for building AI agents

- Installation: pip install ag2
- Docker: optional

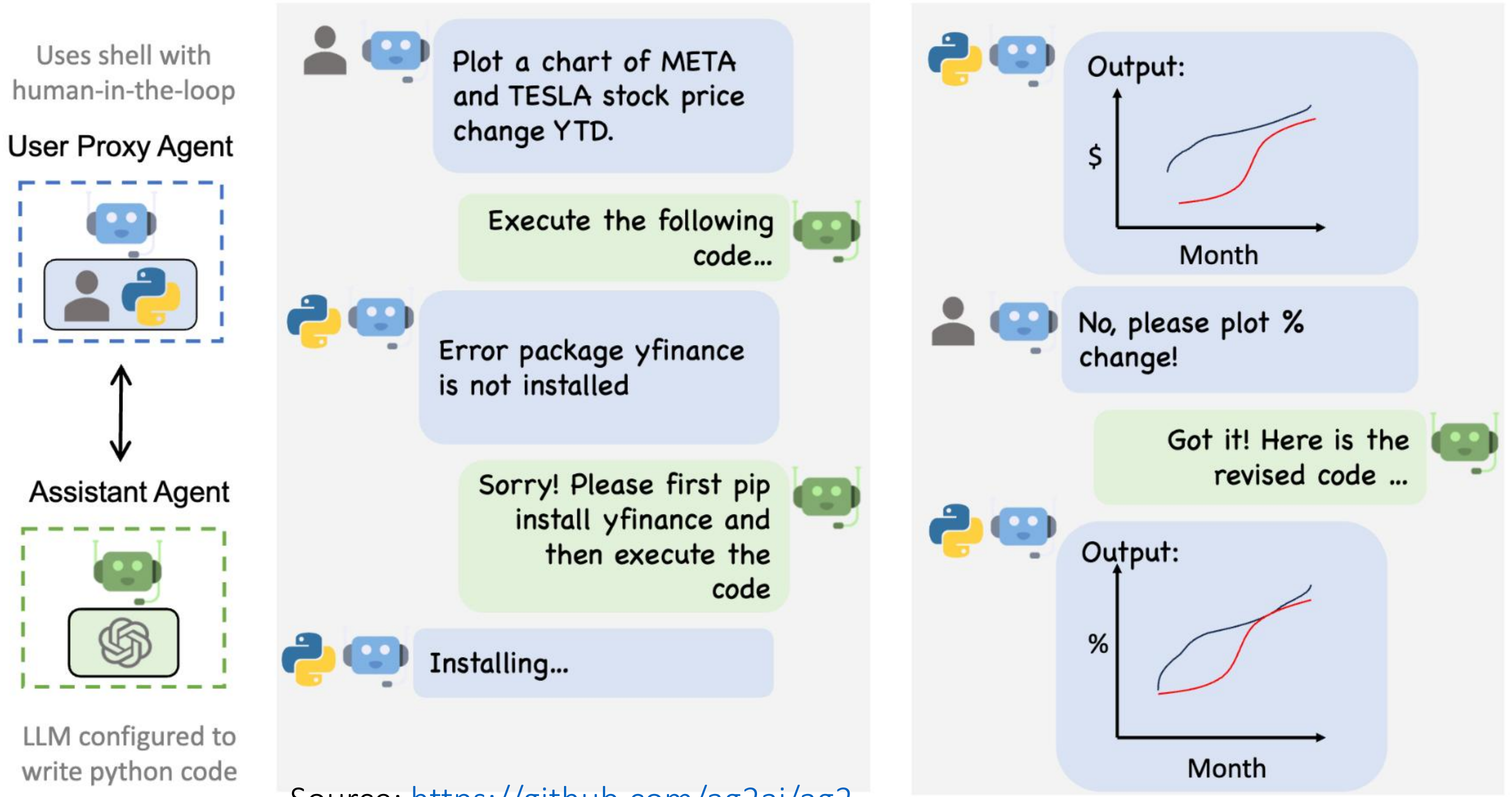




Source: https://github.com/ag2ai/ag2

AG2

Introduction



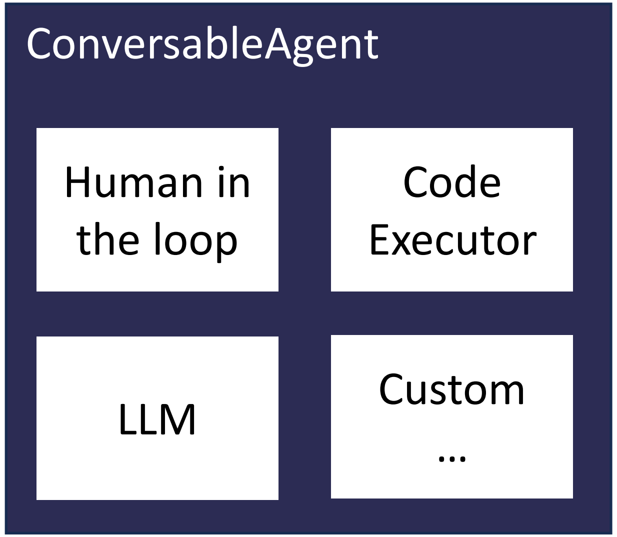
Uses shell with human-in-the-loop
User Proxy Agent
Assistant Agent
LLM configured to write python code
Plot a chart of META and TESLA stock price change YTD.
Execute the following code...
Error package yfinance is not installed
Sorry! Please first pip install yfinance and then execute the code
Installing...
Output:
$
Month
No, please plot % change!
Got it! Here is the revised code ...
Output:
%
Month

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

Agent Types

ConversableAgent

- purpose: interactive, conversational tasks
- features: maintains context across turns in conversation, handles interactions with other agents

AssistentAgent

- purpose: virtual assistant, used for tasks requiring retrieval, summarization, or user support
- features: connection to external knowledge possible

ToolAgent

- purpose: optimized for information extraction
- features: uses vector DB, suitable for RAG

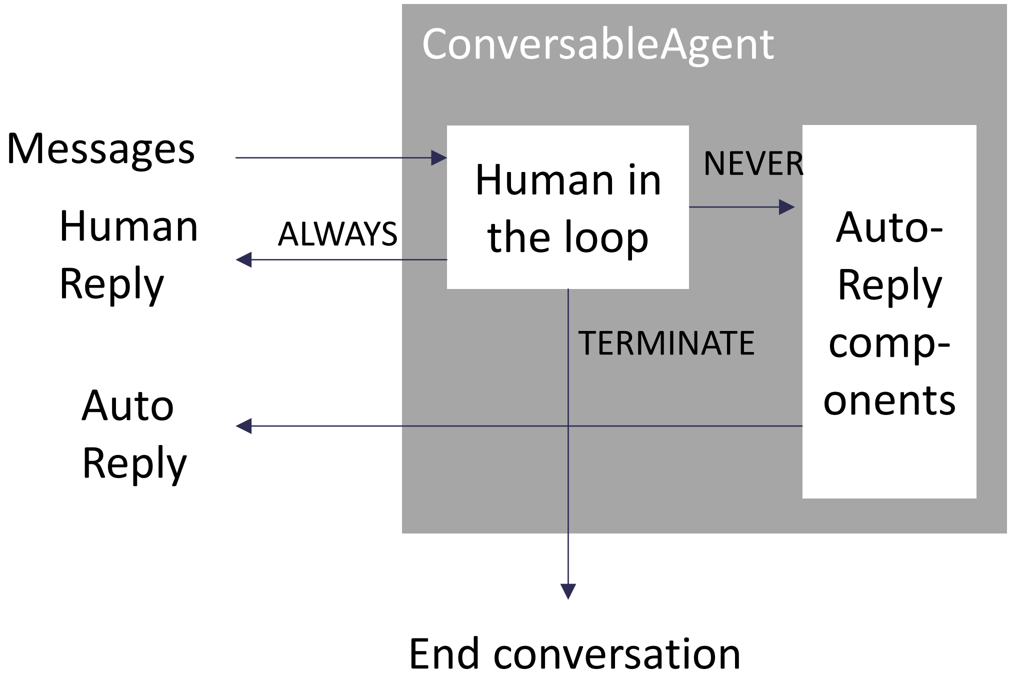
TaskAgent

- purpose: handles code or script execution
- features: capable of running, validation, or testing Python scripts

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



Source: own graph, adapted from https://ag2ai.github.io/ag2/docs/tutorial/human-in-the-loop

AG2

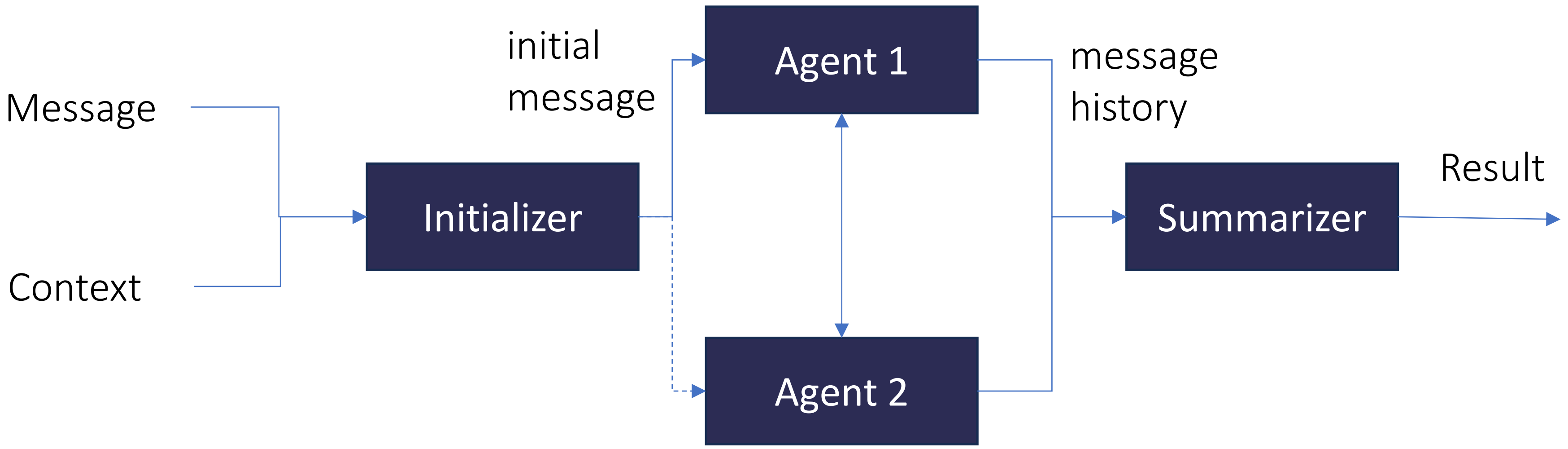
Tools

- 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


Message
Context
Initializer
initial message
Agent 1
Agent 2
message history
Summarizer
Result