AGENTIC AI

BUILD A MULTI-AGENT APPLICATION WITH CREWAI

ALESSANDRO ROMANO

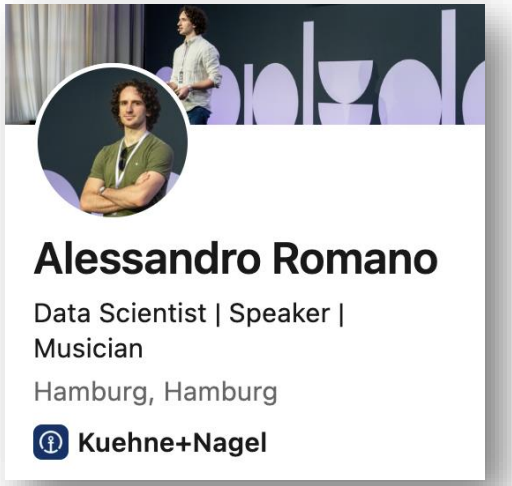
25 APRIL 2025

PYCON & PYDATA 2025

AGENDA

- Objectives and Outcomes (2 min)
- Intro to CrewAI (15 min)
- Assignment 1 – Your First Agent (15 min)
- Assignment 2 - Content Creation with Guardrails (20 min)
- Assignment 3 - Fraud Detection Workflow (20 min)
- Assignment 4 – AI-Powered Onboarding System (15 min)
- Wrap-Up (3 min)

ABOUT ME



Alessandro Romano

Data Scientist | Speaker | Musician

Hamburg, Hamburg

Kuehne+Nagel

Experience

Senior Data Scientist
Kuehne+Nagel · Full-time
Mar 2023 - Present · 2 yrs
Hamburg, Germany · Hybrid

Public Speaker/Data Science Advocate
Self-employed
Aug 2019 - Present · 5 yrs 7 mos
Hybrid

I am passionate about making Data and AI accessible and easy to understand for everyone. With a knack for distilling complex concepts into clear, actionable insights, I regularly share my use cases and lea ...see more

Data Science, Public Speaking and +3 skills

Alessandro Romano - Talks

Data Scientist
FREE NOW · Full-time
Jun 2021 - Jan 2023 · 1 yr 8 mos
Hamburg, Germany · Hybrid

Data Scientist
Cargonexx GmbH
Jun 2018 - May 2021 · 3 yrs
Hamburg Area, Germany

Education

Università di Pisa
Master's degree, Data Science
2015 - 2018
Grade: 110/110 cum Laude

Master's Thesis Topic: "Anomaly Detection for Time Series data - Deep Learning applied to the Auto-Motive field"

Università degli Studi di Bari
Bachelor's degree, Computer Science
2011 - 2015

Bachelor Thesis Topic: "Development of a multivariate algorithm for wind speed forecasting"

OBJECTIVES AND OUTCOMES

Objectives

- Understand CrewAI's architecture and agentic AI concepts
- Learn to define agents, tasks, crews, and tools
- Explore structured outputs, flow control, and async execution
- Apply concepts through hands-on projects

Expected Outcomes

- Ability to build and run agent workflows with CrewAI
- Experience designing multi-agent flows with guardrails and logic
- Practical understanding of hierarchical and conditional orchestration
- Ready-to-use code for real-world use cases (content, fraud, onboarding)

AGENTIC AI

- Agents take actions
- Use tools
- Implement reasoning through LLM
- Access an additional knowledge
- Interact with other agents to accomplish final goal
- Able to take decisions in a dynamic environment

IMPLEMENT REASONING

- How do we reason?
- Can we call it reasoning?
- Agents are not new!
- Reasoning is implemented through an LLM

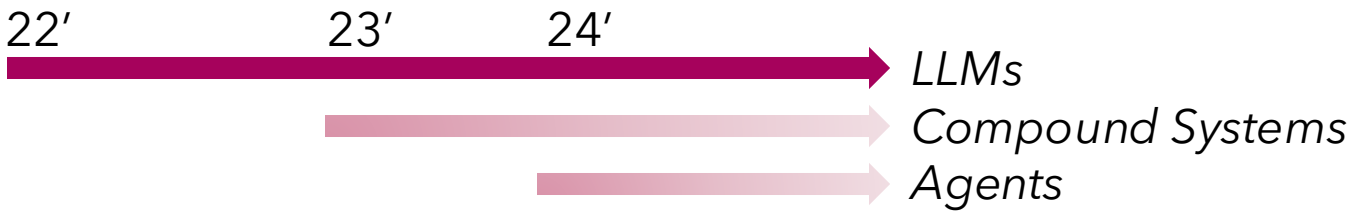
CAN WE CALL IT AN AGENT?

A deterministic agent or service that implements a deterministic logic with no llm, could also be calssified as an agent, with the difference that is probably confined in a specific environment, where the laws in it don’t change.

Going from point A to point B is a trivial task, but if something unexpected happens in the middle, an Agent can read the environment and reason through an LLM in order to take the most appropriate next action.

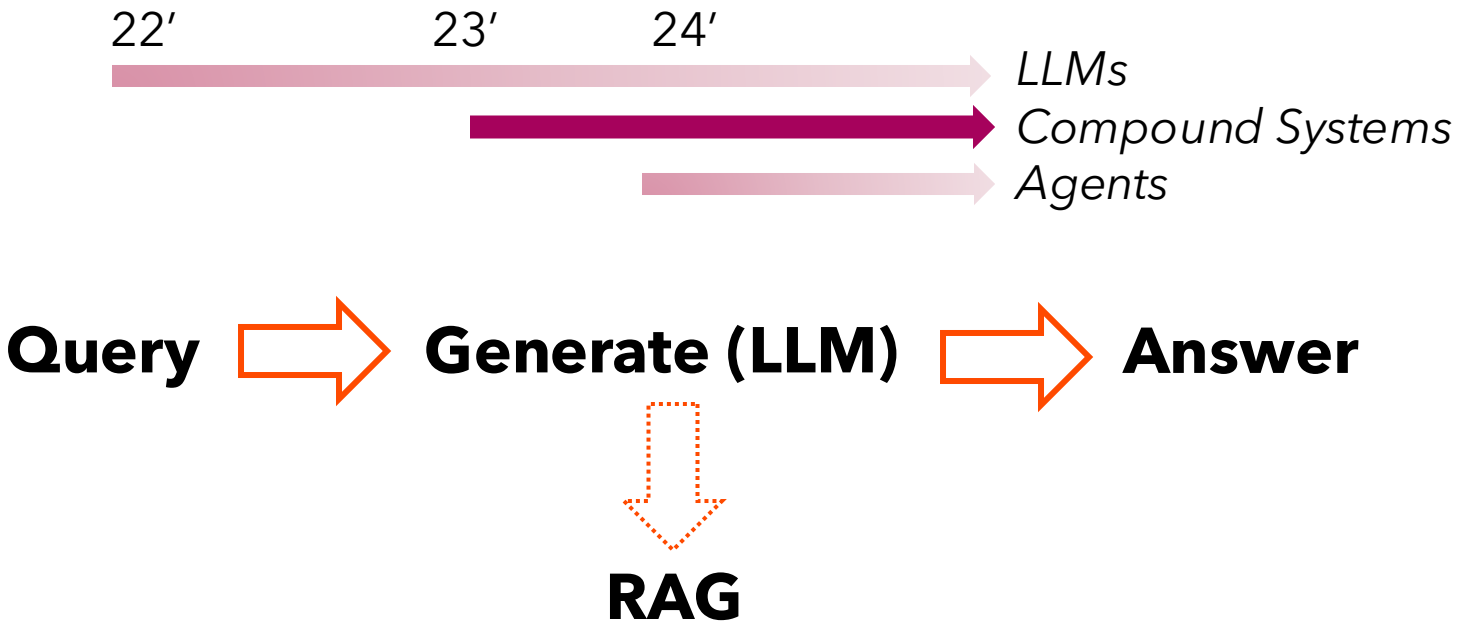
Therefore, Agents are not always needed.

THE DAWN OF LLMS



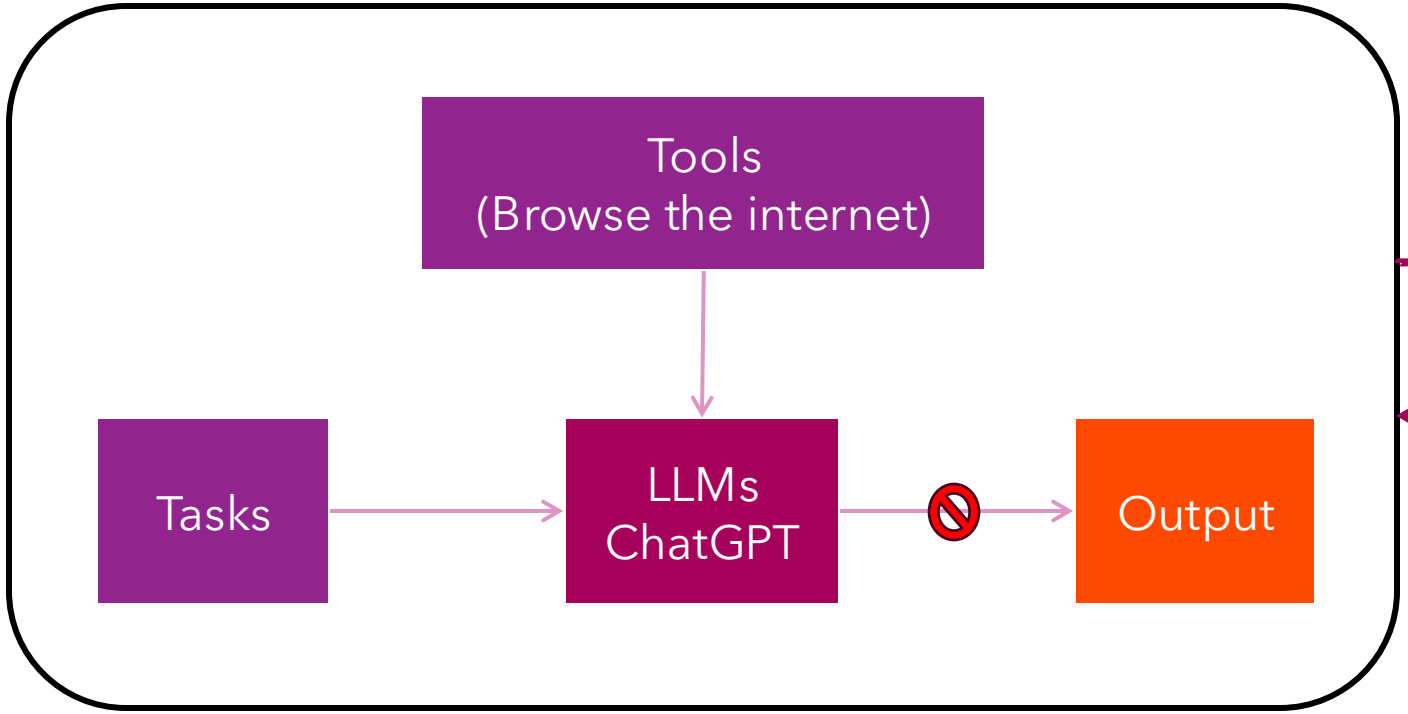
Query ⇨ **Generate (LLM)** ⇨ **Answer**

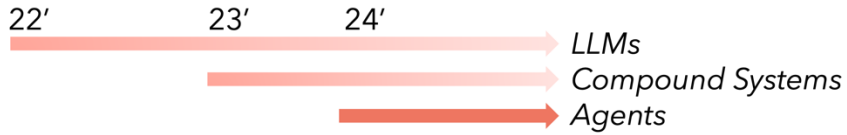
COMPOUND SYSTEMS



(Retrieval-Augmented Generation)

LLM AGENTS





- **Access Memory**
- **Actions (Browse Web)**
- **Incorporates "Reasoning"**

LLM AGENTS FRAMEWORKS

Some interesting mentions:

1. **LangChain**: Versatile for building applications with LLMs, including chains, agents, and memory.
2. **Hugging Face Transformers**: A leading library with extensive model options and community support.
3. **LlamaIndex**: Known for integrating LLMs with external data sources like databases and documents.
4. **Haystack**: Strong in question-answering and retrieval-based applications.
5. **DeepSpeed**: Optimizes large models for efficiency and scalability in production environments.

CREWAI

Cutting-edge framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling complex tasks:

- Provide **simple** tools for dealing with LLMs
- Implement "**orchestration**" strategies
- Core Concepts like **Agents, Tasks, and Crews**

AGENTS

An **agent** is an **autonomous unit** programmed to:

- Perform tasks
- Make decisions
- Communicate with other agents

Think of an agent as a member of a team, with specific skills and a particular job to do. Agents can have different roles like *Researcher*, *Writer*, or *Customer Support*, each contributing to the overall goal of the crew.

```python
from crewai import Agent

agent2 = Agent(
    role="agent role",
    goal="summarize the short bio for {input} and if needed do more research",
    backstory="agent backstory",
    verbose=True,
)
```

https://docs.crewai.com/core-concepts/Agents/

TASKS

In the crewAI framework, **tasks** are assignments given to agents, including details like descriptions, responsible agents, and required tools. Tasks can be collaborative, involving multiple agents working together, with the Crew's process managing coordination for better teamwork and efficiency.

```python
from crewai import Task

task = Task(
    description='Find and summarize the latest and most relevant news on AI',
    agent=sales_agent,
    expected_output='A bullet list summary of the top 5 most important AI news',
)
```

https://docs.crewai.com/core-concepts/Tasks/

CREWS

A **crew** in crewAI represents a collaborative group of agents working together to achieve a set of tasks. Each crew defines the strategy for task execution, agent collaboration, and the overall workflow.

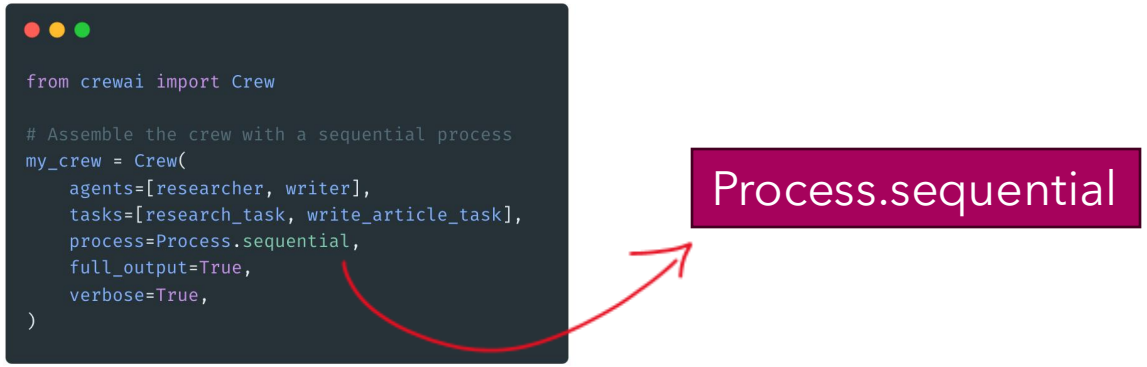
```
from crewai import Crew

# Assemble the crew with a sequential process
my_crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, write_article_task],
    process=Process.sequential,
    full_output=True,
    verbose=True,
)
```

https://docs.crewai.com/core-concepts/Crews/

PROCESSES

In CrewAI, **processes** orchestrate the execution of tasks by agents, akin to project management in human teams. These processes ensure tasks are distributed and executed efficiently, in alignment with a predefined strategy:

- **Sequential**: Executes tasks one after another in a specific order.
- **Hierarchical**: Organizes tasks using a chain of command, where a manager model or agent oversees and delegates tasks.
- **Consensual** (Planned): Intended for future development, this process will enable agents to collaborate and make decisions democratically on task execution.



TOOLS

CrewAI **tools** empower agents with capabilities like web searching, data analysis, collaboration, and task delegation, allowing users to build custom tools or leverage CrewAI and LangChain tools for seamless teamwork and complex interactions.

```python
from crewai import Agent, Task, Crew
from crewai_tools import SerperDevTool

# Set up the agent with a web search tool
researcher = Agent(
    role='Market Research Analyst',
    tools=[SerperDevTool()],
    verbose=True
)

# Define a single task
research = Task(description='Research AI industry trends.', agent=researcher)

# Assemble and run the crew
Crew(agents=[researcher], tasks=[research], planning=True).kickoff()
```

GUARDRAILS

With **Flow**, we can create conditional logics that help us to control the application's internal flow.

For instance, we can handle those cases where the **LLM hallucinates** and trigger the agent N number of times untill we get a "safe" output.

```python
from crewai.flow.flow import Flow, and_, listen, start

class AndExampleFlow(Flow):

    @start()
    def start_method(self):
        self.state["greeting"] = "Hello from the start method"

    @listen(start_method)
    def second_method(self):
        self.state["joke"] = "What do computers eat? Microchips."

    @listen(and_(start_method, second_method))
    def logger(self):
        print("---- Logger ----")
        print(self.state)

flow = AndExampleFlow()
flow.kickoff()
```

https://docs.crewai.com/concepts/flows

CONDITIONAL TASKS

Conditional Tasks in CrewAI let workflows adapt based on previous task results, enabling selective execution for more flexible and efficient processes.

```python
class EventOutput(BaseModel):
    events: List[str]

task1 = Task(
    description="Fetch data about events in San Francisco using Serper tool",
    expected_output="List of 10 things to do in SF this week",
    agent=data_fetcher_agent,
    output_pydantic=EventOutput,
)

conditional_task = ConditionalTask(
    description="""
        Check if data is missing. If we have less than 10 events,
        fetch more events using Serper tool so that
        we have a total of 10 events in SF this week..
        """,
    expected_output="List of 10 Things to do in SF this week",
    condition=is_data_missing,
    agent=data_processor_agent,
)
```

TASKS INPUT/OUTPUT

```
get_recipe = Task(
    description="Give me recipe given my food allergies {food_allergies}.",
    agent=chef,
)

...

crew.kickoff(inputs={'food_allergies': 'Peanuts'})
```

Input:

- Use *inputs* field in *kickoff()*
- Wrap your tasks into a function

The expected output is a safe yet powerfull way to make sure the next agent is able access the input data correctly.

```
task = Task(
  ...
  expected_output="image_url: URL of the generated image",
  ...
)
```

ACCESS TASKS OUTPUT

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

# Define an agent
weather_agent = Agent(
    role="Weather Analyst",
    goal="Fetch and summarize today's weather forecast"
)

# Example task
task = Task(
    description='Summarize today’s weather forecast for New York City',
    expected_output='A short weather summary including temperature and
conditions',
    agent=weather_agent
)

# Execute the crew
crew = Crew(
    agents=[weather_agent],
    tasks=[task],
    verbose=True
)

result = crew.kickoff()

# Accessing the task output
output = task.output

print(f"Task Description: {output.description}")
print(f"Task Summary: {output.summary}")
print(f"Raw Output: {output.raw}")
if output.json_dict:
    print(f"JSON Output: {output.json_dict}")
if output.pydantic:
    print(f"Pydantic Output: {output.pydantic}")
```

ASYNCHRONOUS CREW KICKOFF

```python
# Async function to kickoff multiple crews asynchronously and wait for all to finish
async def async_multiple_crews():
    result_1 = crew_1.kickoff_async(inputs={"ages": [25, 30, 35, 40, 45]})
    result_2 = crew_2.kickoff_async(inputs={"ages": [20, 22, 24, 28, 30]})

    # Wait for both crews to finish
    results = await asyncio.gather(result_1, result_2)

    for i, result in enumerate(results, 1):
        print(f"Crew {i} Result:", result)

# Run the async function
asyncio.run(async_multiple_crews())
```

HANDS-ON GUIDELINES

Explore the slides and the website

Feel free to ask me anything

Share your ideas with fellow attendees

Lend a hand to your neighbors

Get creative – and most importantly, have fun!

https://pigna90.github.io/crewai-workshop-pyconde-2025/

THANK YOU