Design, Develop, and Deploy Multi-Agent Systems with CrewAI

Managing Systems of AI Agents

DeepLearning.AI

Managing Systems of AI Agents

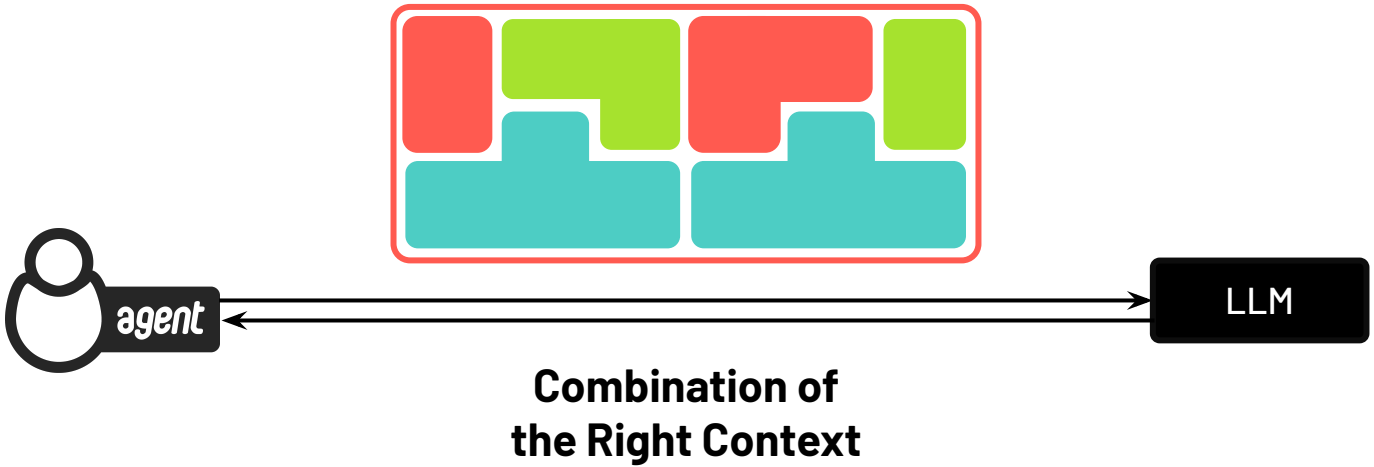
Collaboration

crewai

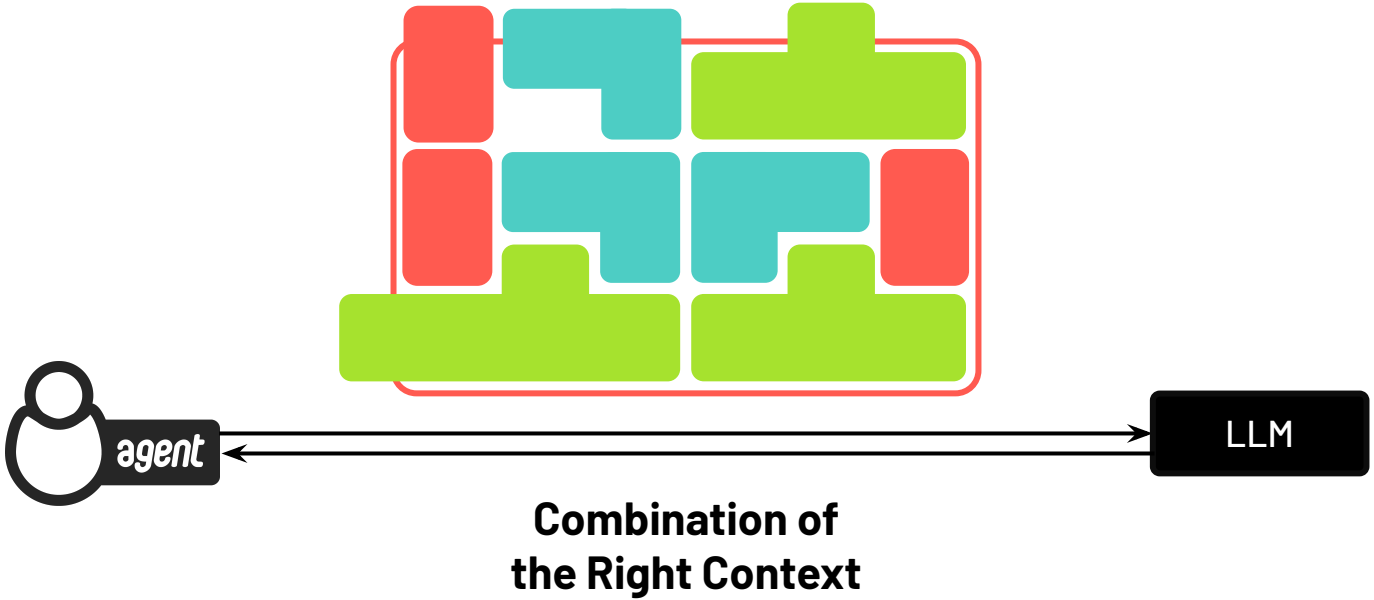
DeepLearning.AI

Single Agent System

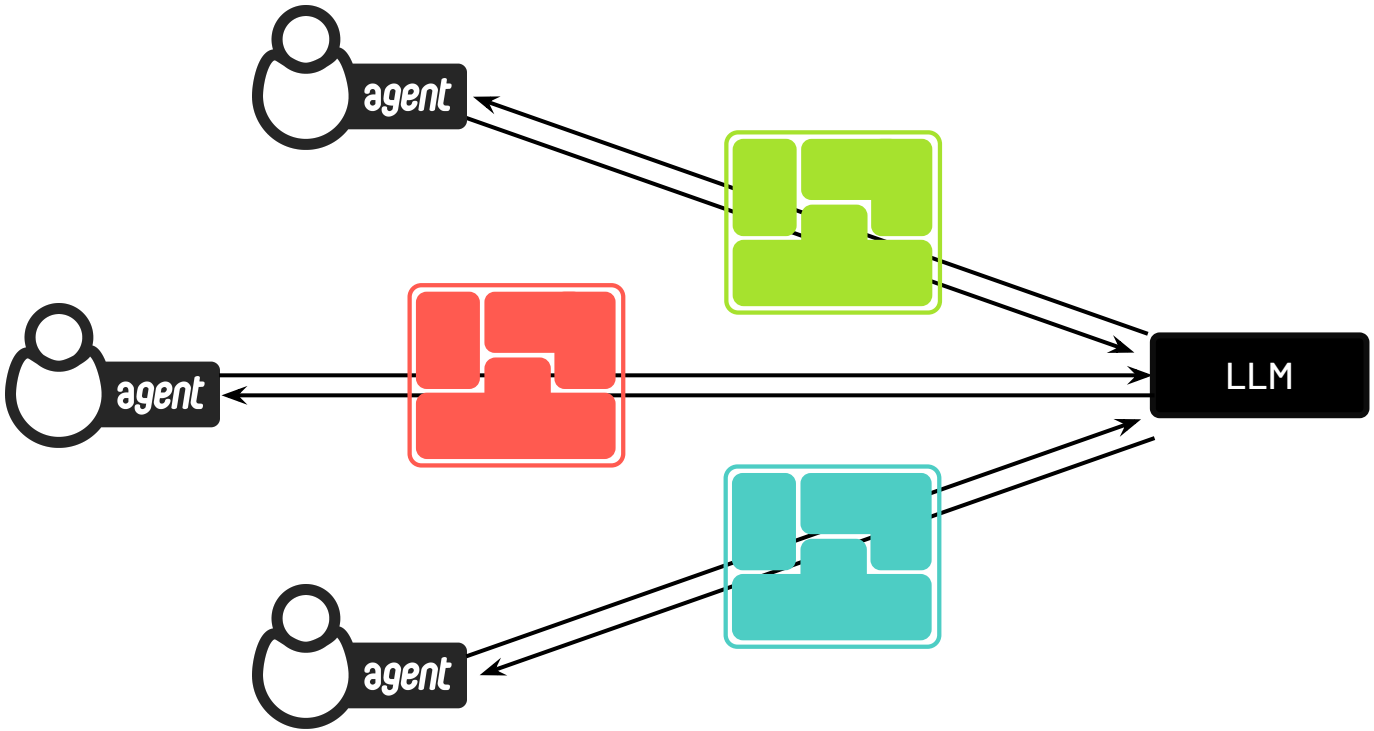
Single Agent System



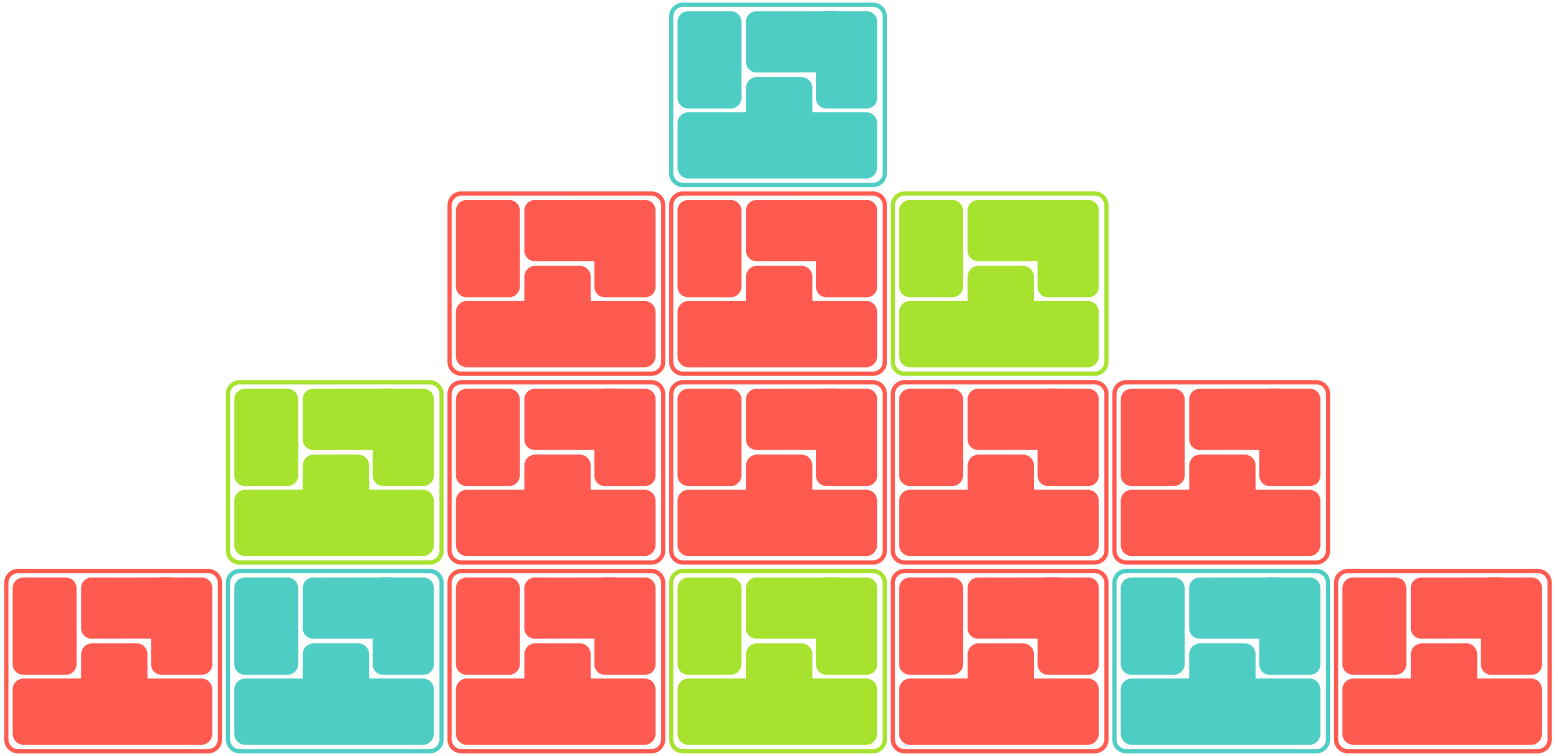
Single Agent System



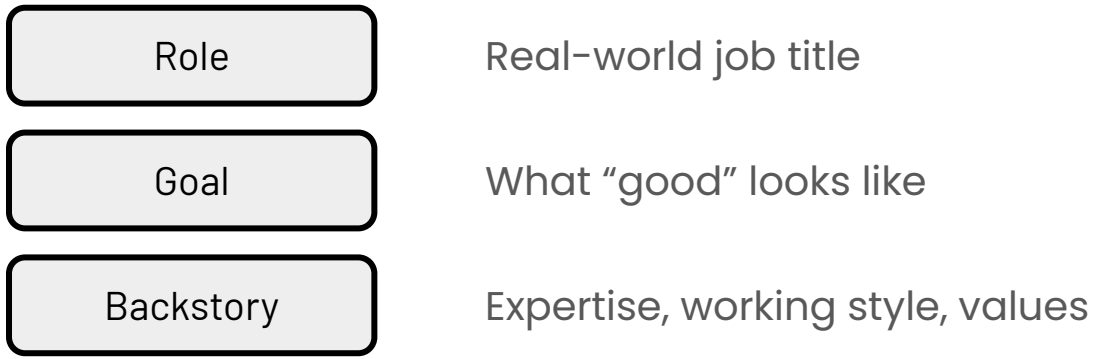
Multi-Agent System



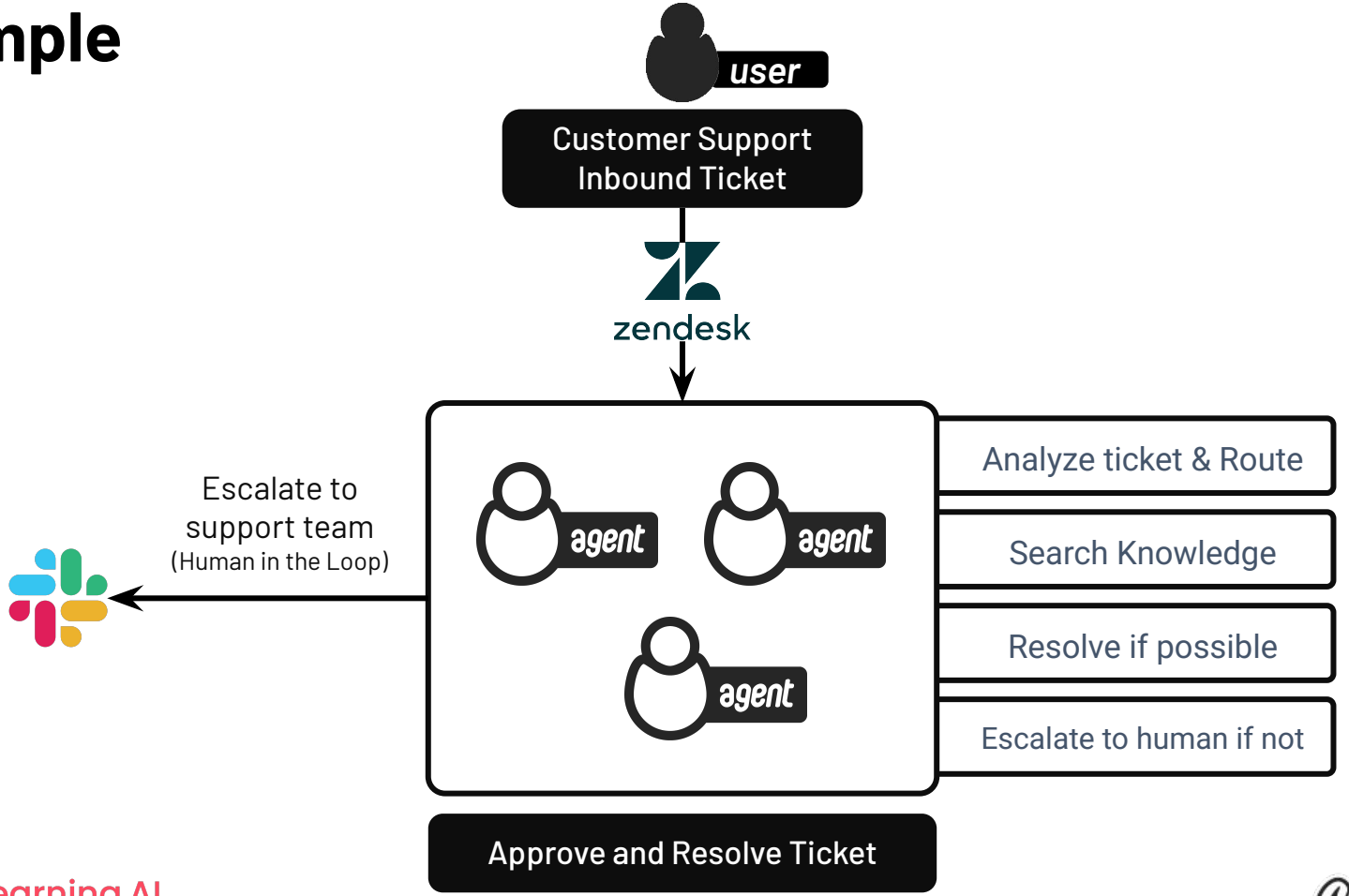
Breaking Down Complexity



Agent


Role
Real-world job title
Goal
What “good” looks like
Backstory
Expertise, working style, values

Example


user
Customer Support Inbound Ticket
zendesk
Escalate to support team
(Human in the Loop)
agent
agent
agent
Analyze ticket & Route
Search Knowledge
Resolve if possible
Escalate to human if not
Approve and Resolve Ticket

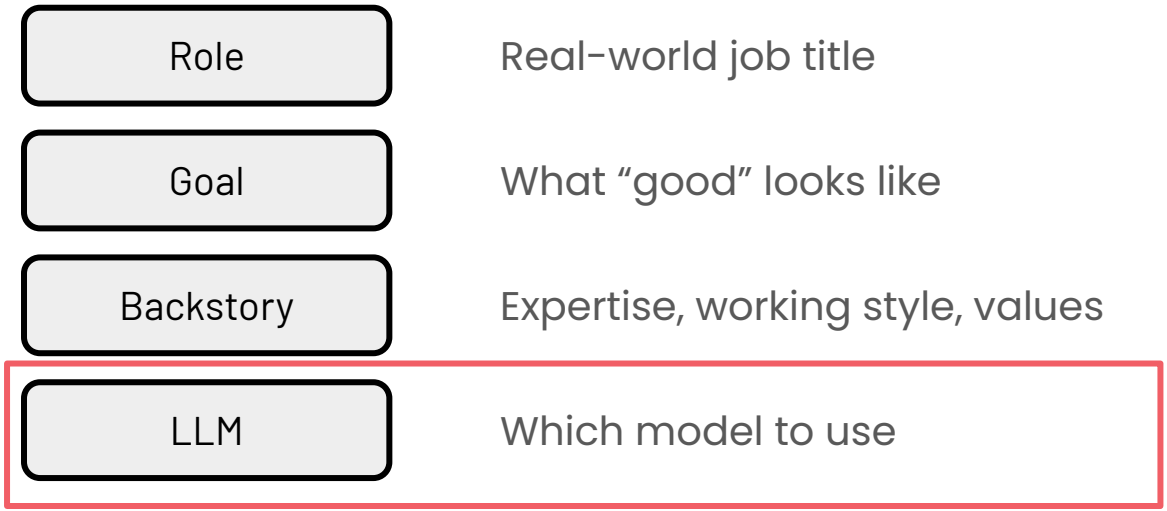
Specialists > Generalists

Focused agents deliver precise outputs

Example:
Technical Blog Writer Agent instead of Writer Agent
If the goal is to have a technical blog creation agent.

- Specialize in role, versatile in application
- Complementary skills across the crew

Agent


Role
Real-world job title
Goal
What “good” looks like
Backstory
Expertise, working style, values
LLM
Which model to use

Smaller models

- *Cheaper*
- *Faster*

Speed

Quality

Bigger Models

- *More capable reasoning*
- *Follow instructions better*

Smaller models

- *Cheaper*
- *Faster*

Bigger Models

- *More capable reasoning*
- *Follow instructions better*

Smaller models

- *Cheaper*
- *Faster*

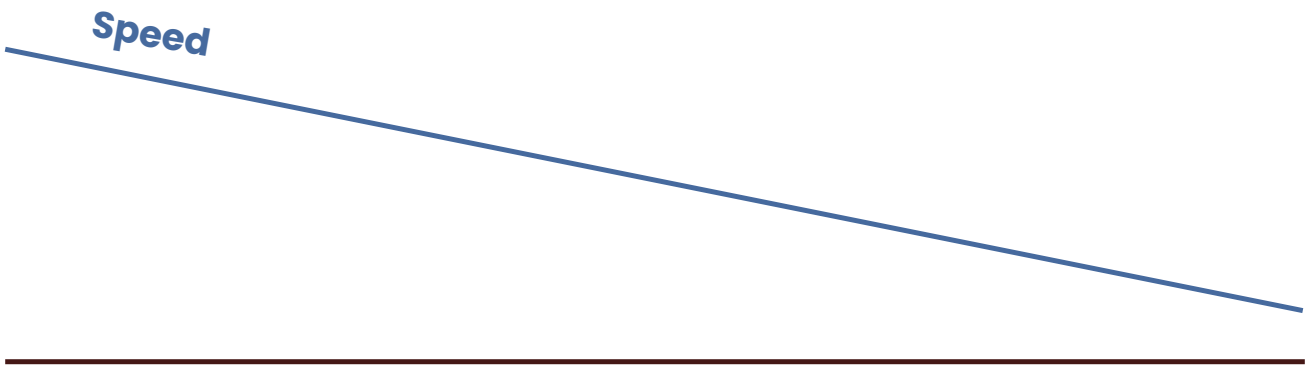
Speed

Quality

Consistency

Bigger Models

- *More capable reasoning*
- *Follow instructions better*



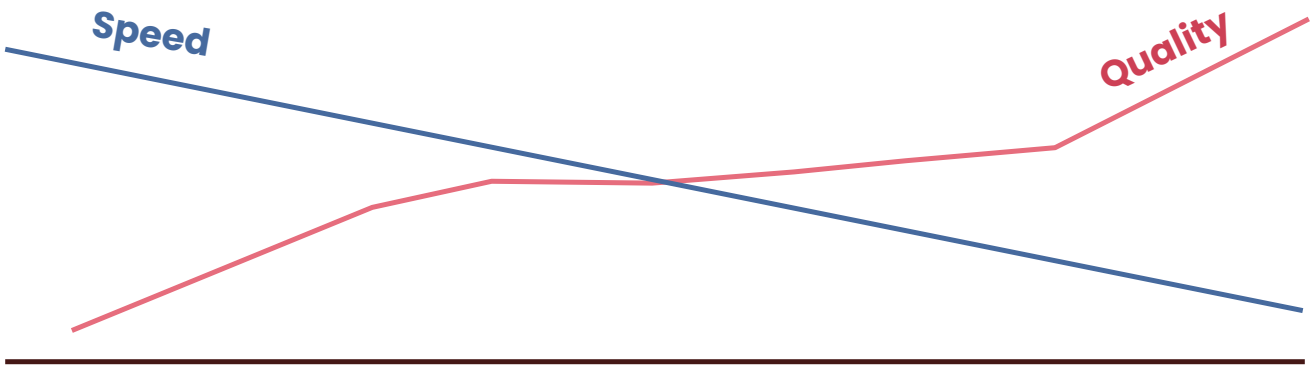
Smaller models
- *Cheaper*
- *Faster*

Speed

Quality

Bigger Models
- *More capable reasoning*
- *Follow instructions better*

Consistency



Managing Systems of AI Agents

Communication

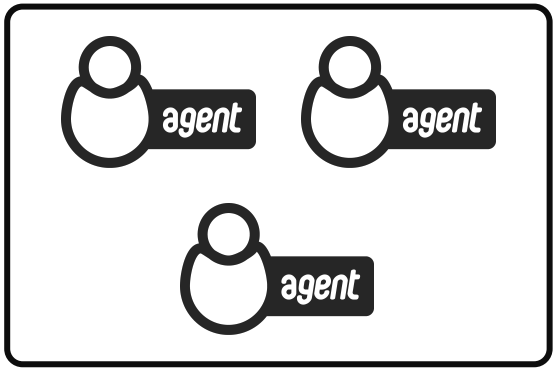
crewai

DeepLearning.AI

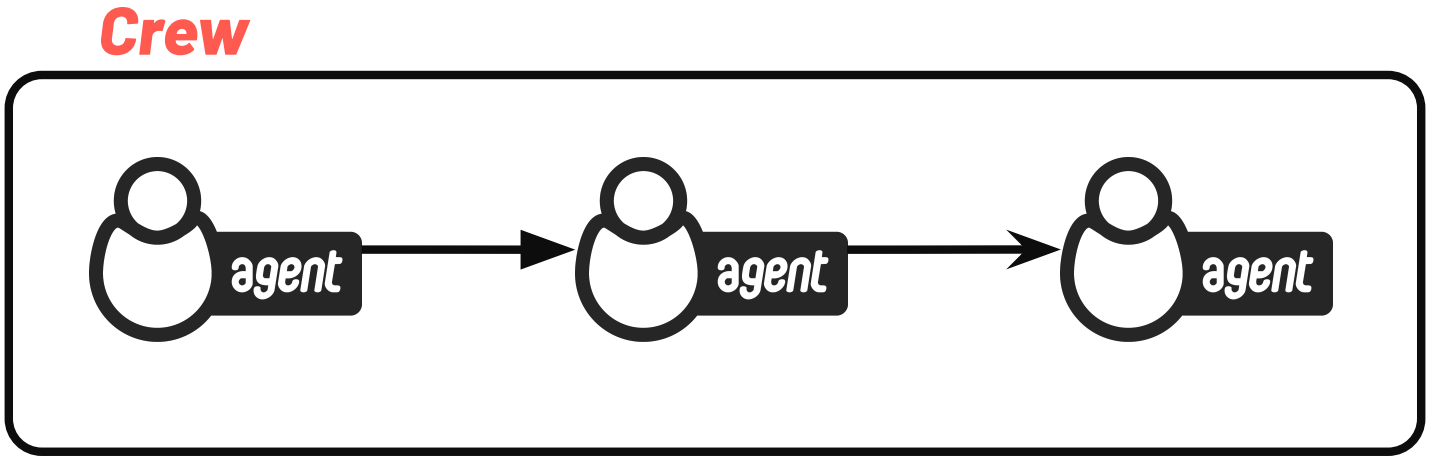
Agents vs Crews

Single Agent

Multi-Agent

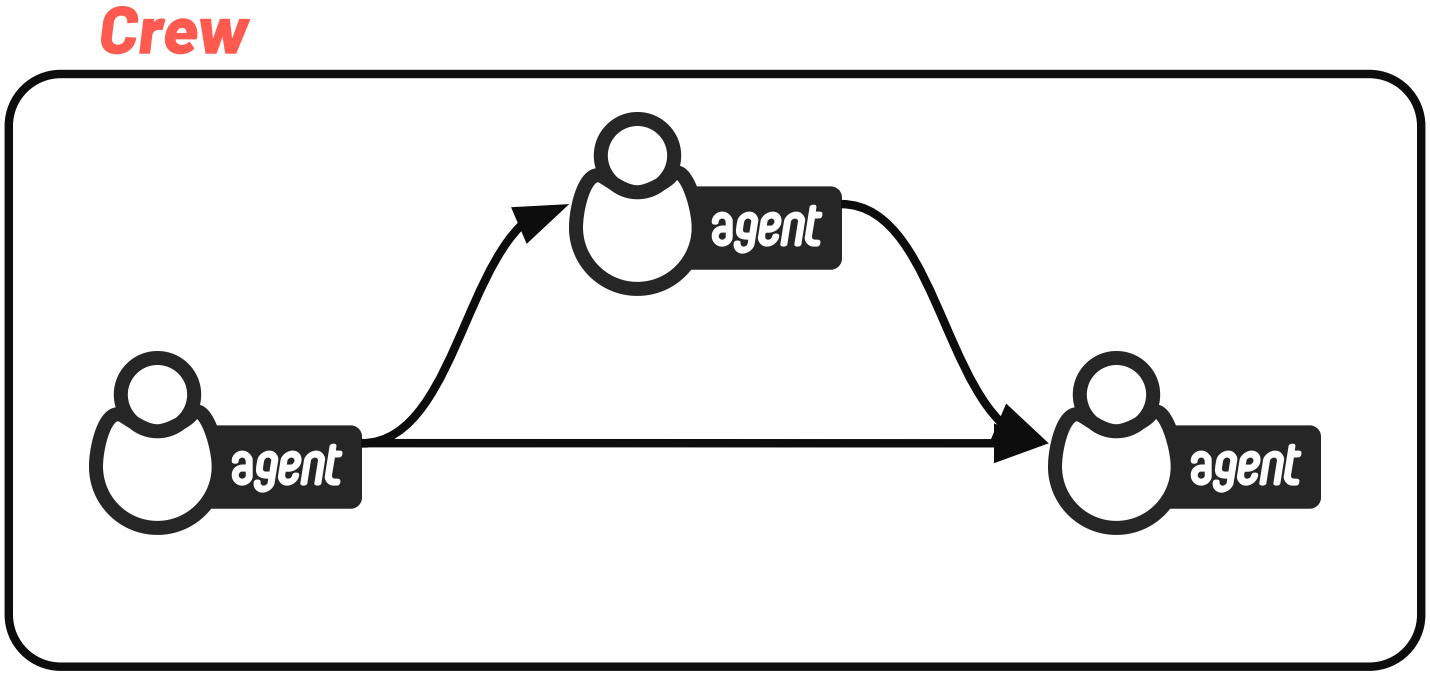


Sequential Process



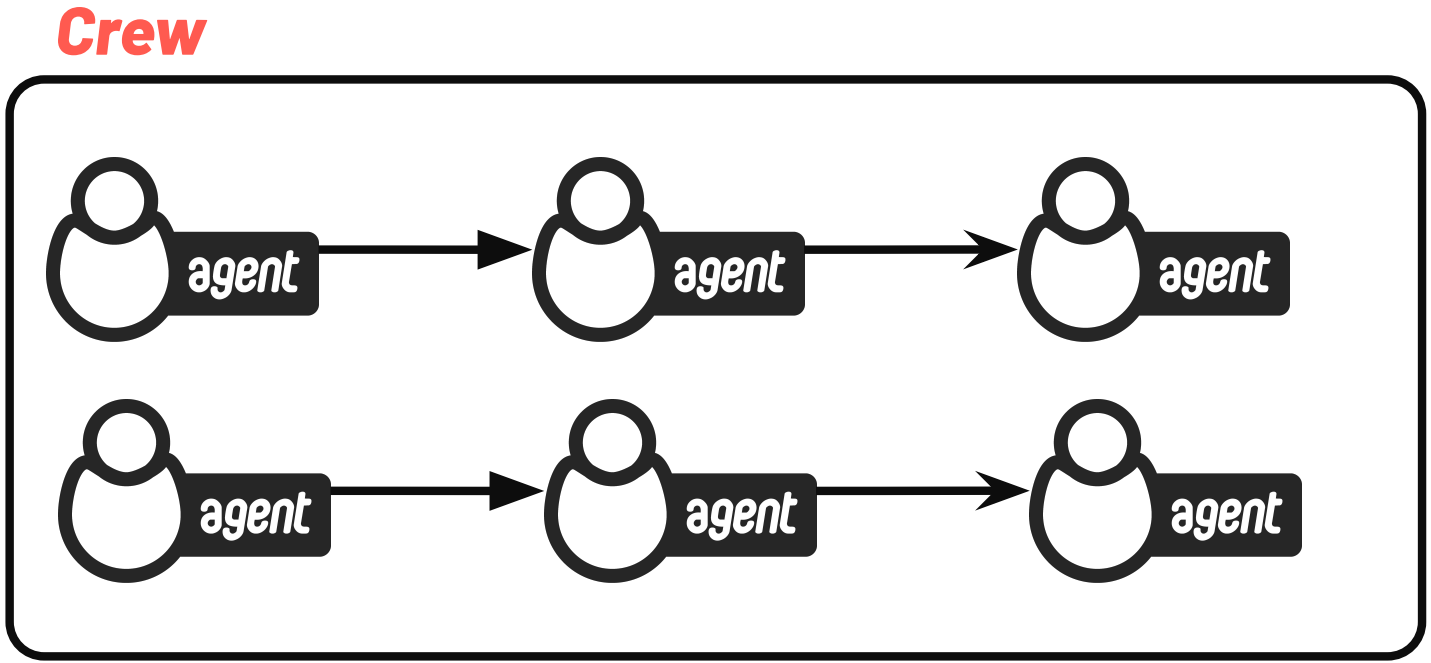
Output of each Agent's Task serves
as context for the next one.

Sequential Process

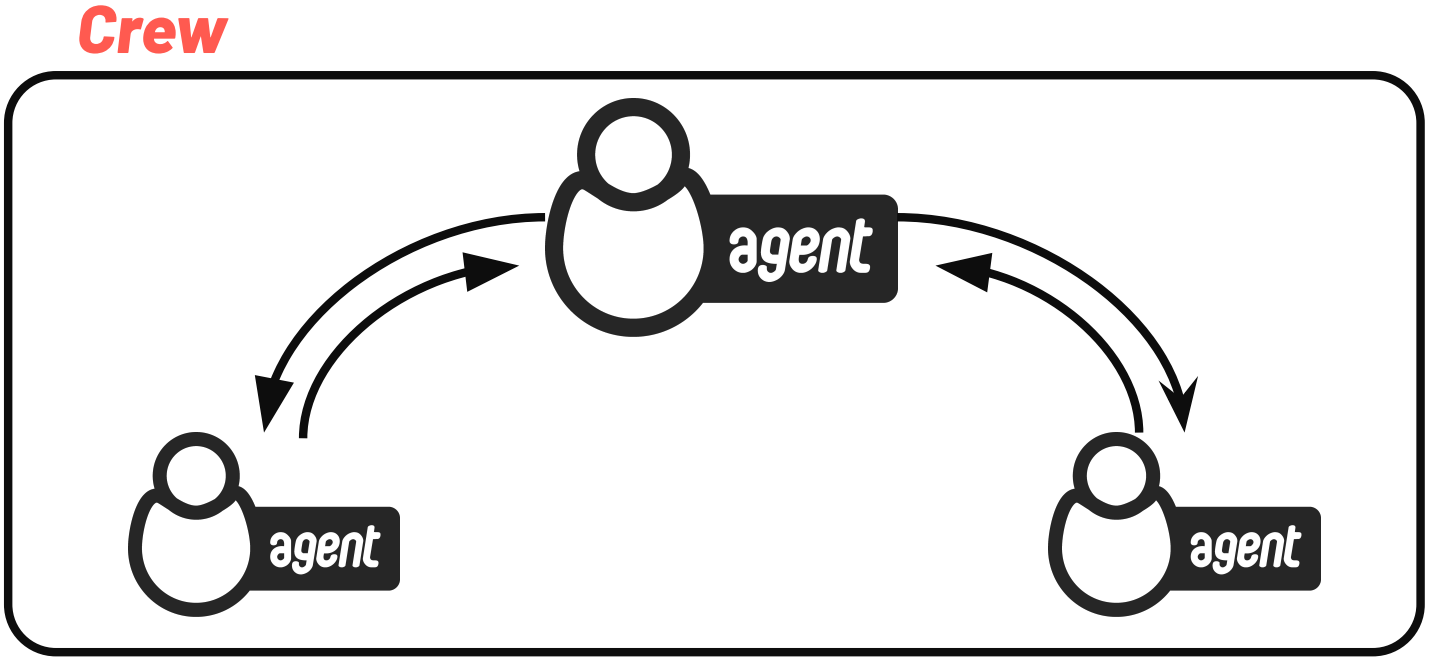


```
# Create a planning task
plan = Task(description="Develop a detailed plan to create a Ruby on Rails app...",
            expected_output="A detailed product specification document...",
            agent=planning_agent)

# Create a testing task
tests = Task(description="Create a set of unit tests for this app...",
             expected_output="A set of runnable unit tests...",
             agent=tester_agent,
             context=[plan])

# Create a app
app = Task(description="Develop a Ruby on Rails app using the specs, that passes the tests...",
           expected_output="A fully functional Ruby on Rails app...",
           agent=coding_agent,
           context=[plan, tests])
```

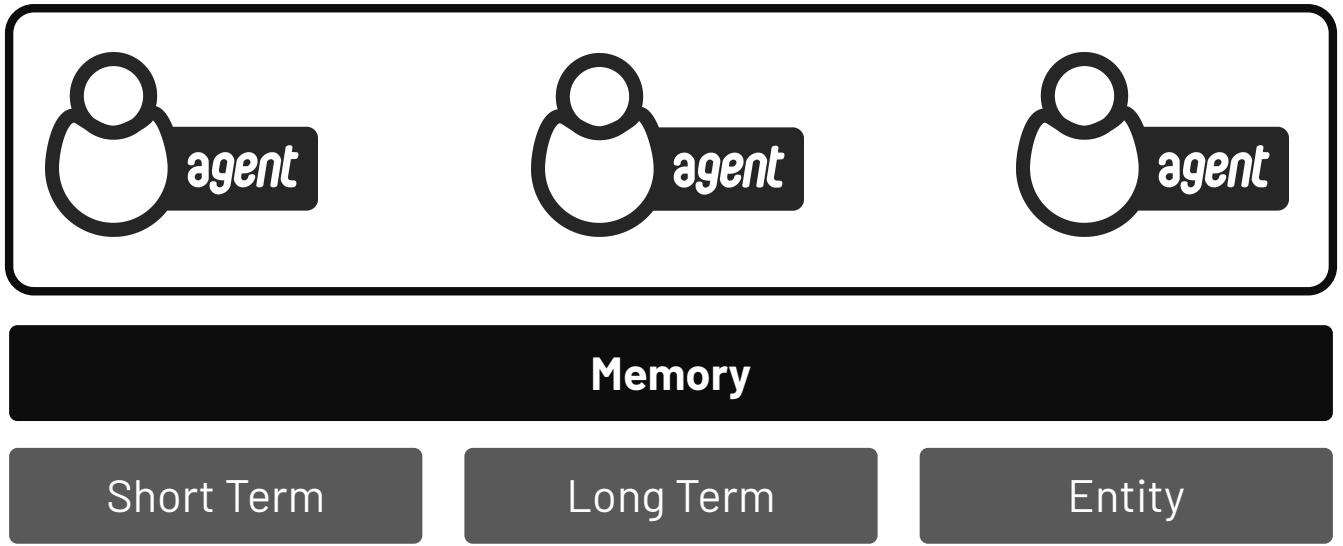
Parallel Process



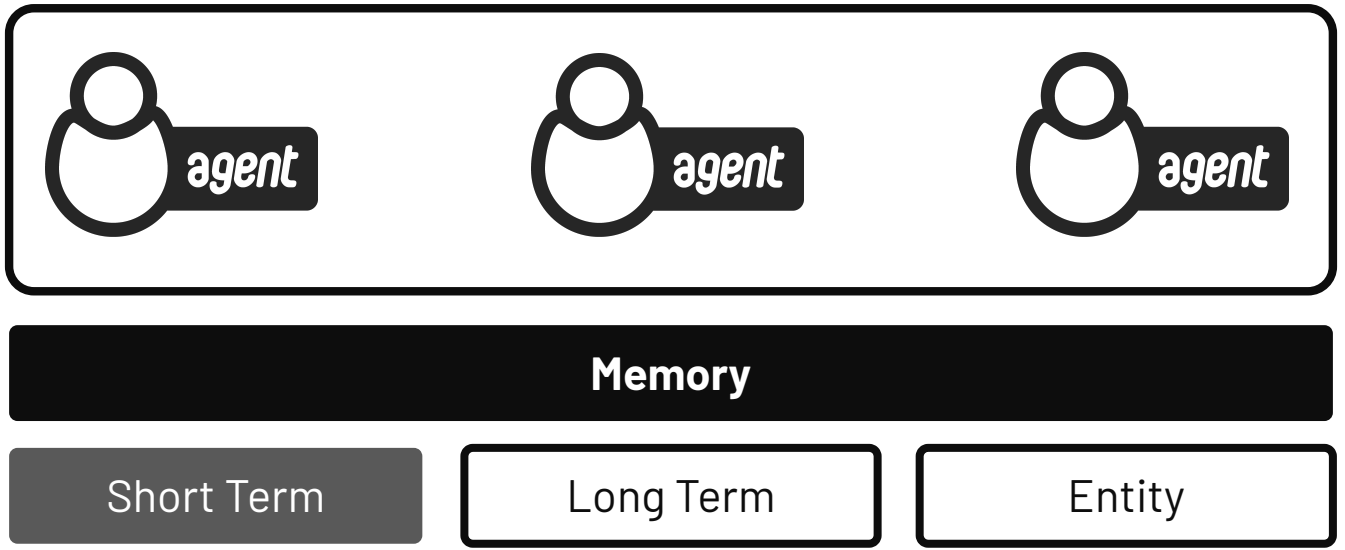
Hierarchical Process



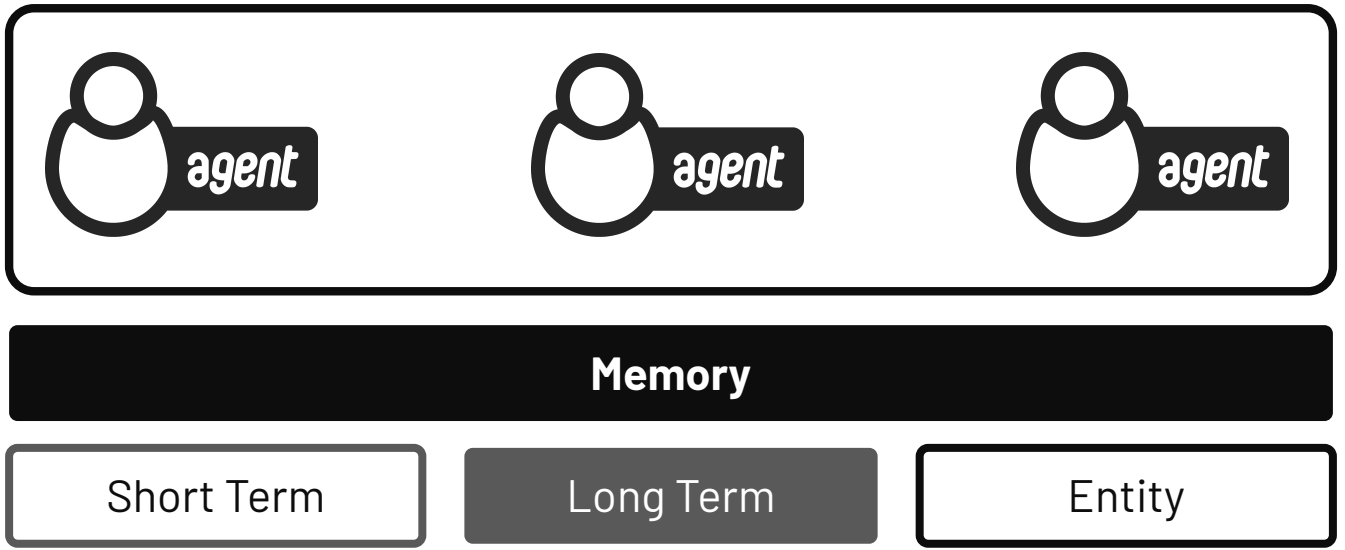
Using Memory for Communication between Agents



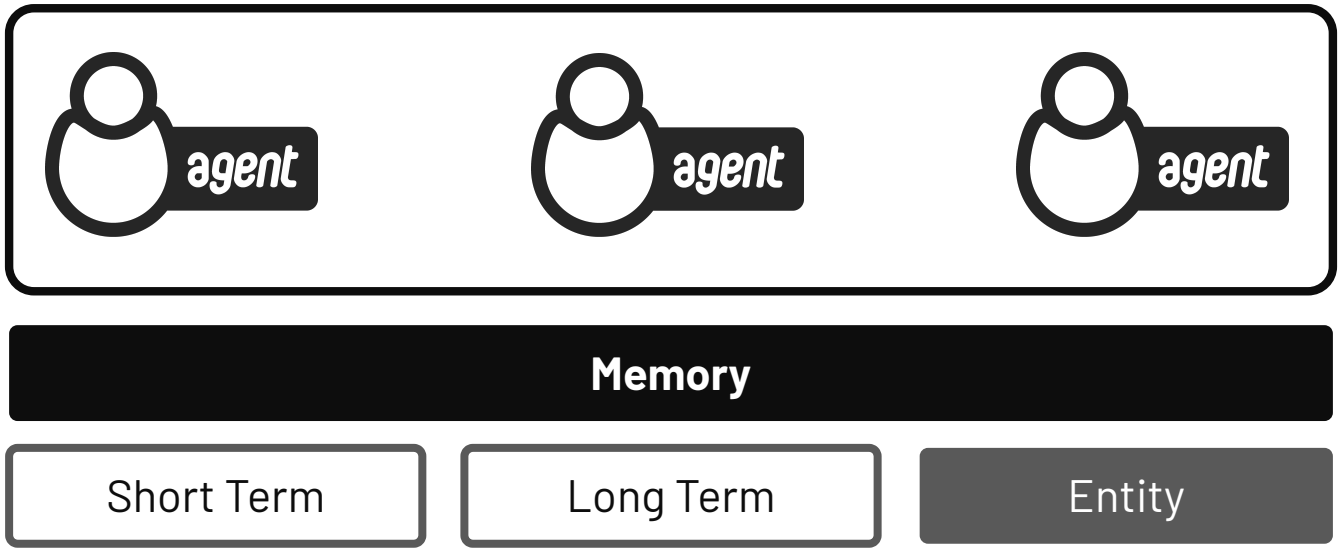
Using Memory for Communication between Agents



Using Memory for Communication between Agents



Using Memory for Communication between Agents



Common Coordination Issues

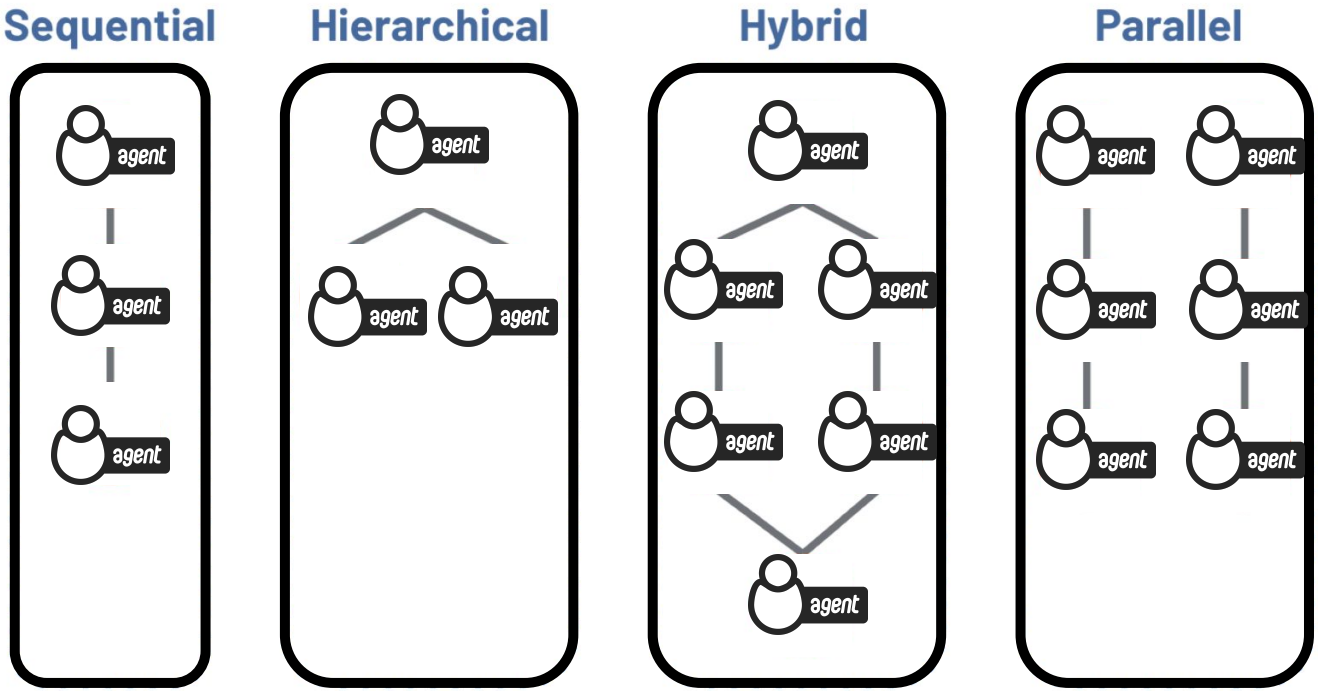
Redundant work:

Tighten the scope of tasks each agent performs to avoid overlap; start with parallel process then hierarchical when needed.

Slow serial chains:

Mark independent tasks **`async_execution=True`** and gate with a later task's **`context`**

Process Types



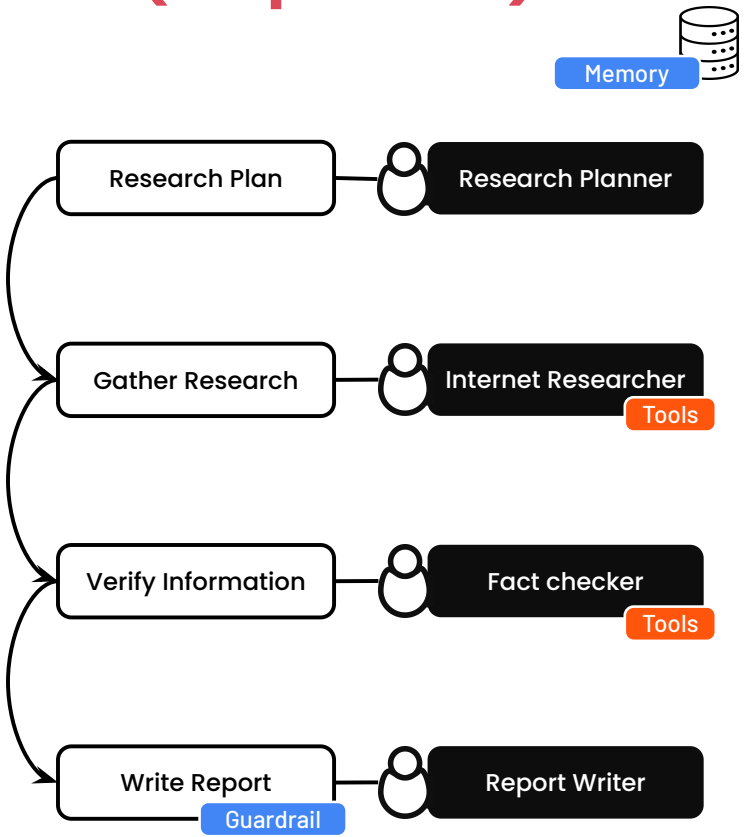
Managing Systems of AI Agents

Building Coordination Patterns

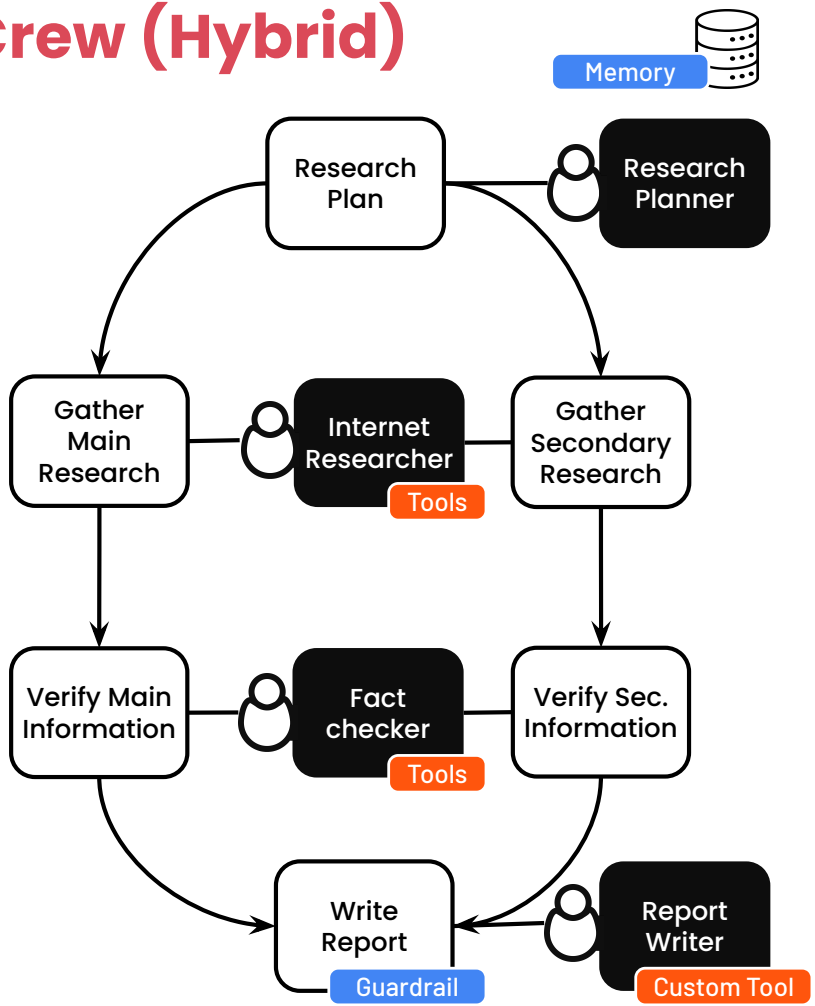
crewai

DeepLearning.AI

Deep Research Crew (Sequential)



Deep Research Crew (Hybrid)



Managing Systems of AI Agents

Using the A2A Protocol

crewai

DeepLearning.AI

Agent2Agent Protocol

Agent2Agent Protocol **(A2A)** is an open standard for AI agent communication across different frameworks and vendors

simplicity

Decouple public specifications and private implementations of agents

standardization

Promotes interoperability between agents without wrapping them as tools

security

Built on familiar constructs like HTTP / JSON-RPC with support for authentication

Key components of A2A

Agent Card	Document describing agent identity, capabilities, and connection details
Task	Stateful unit of work supporting long horizon jobs
Message	Single turn of conversation between agents and/or users
Artifact	Output from tasks consisting of multimodal data

A2A versus MCP

- A2A complements MCP
- Use MCP to connect agents to tools
- Use A2A to connect agents to agents
- Wrapping agents as tools could limit their capabilities for collaboration and communication

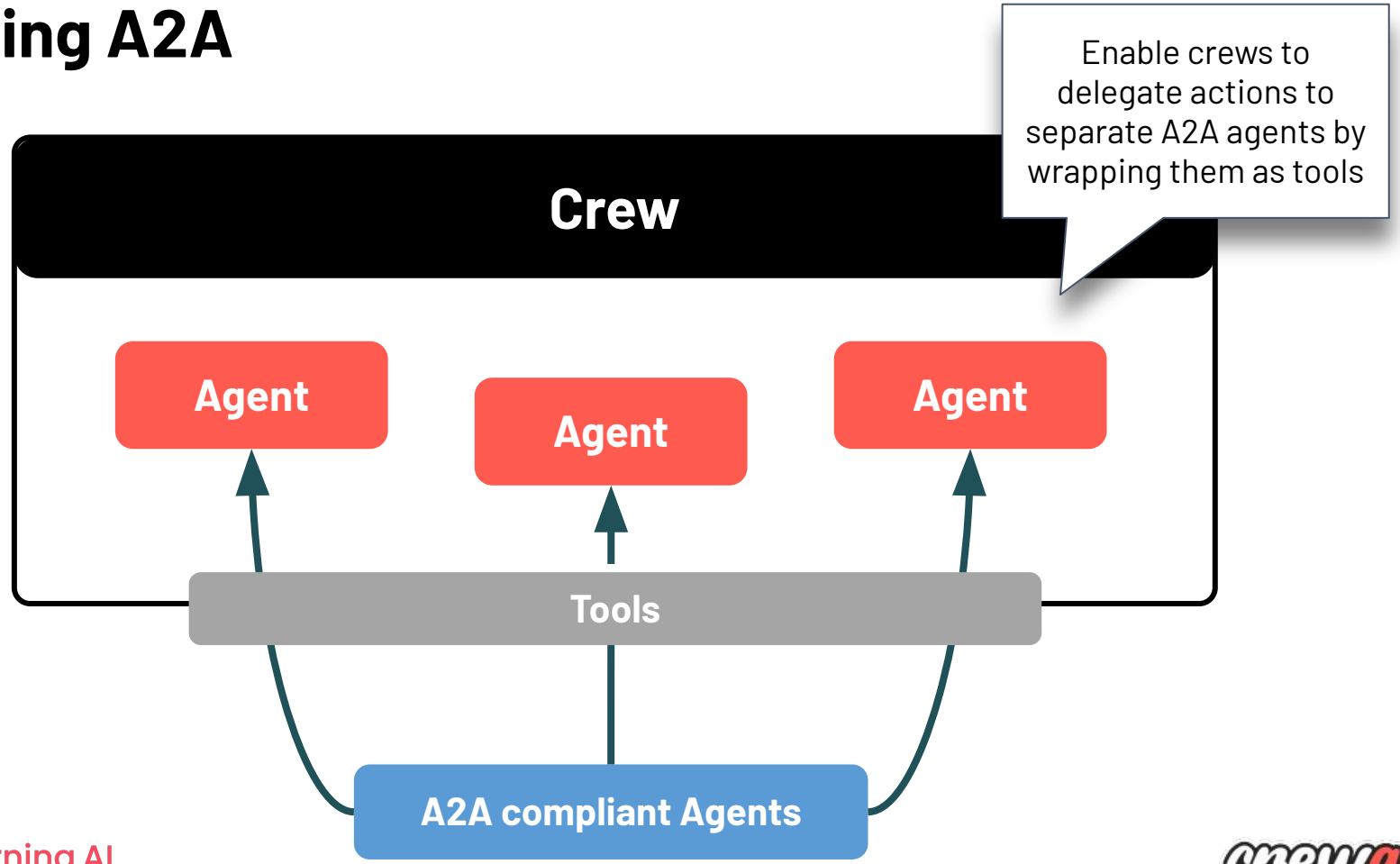
MCP

Stateless standard for connecting agent to tool, prompts or datasets

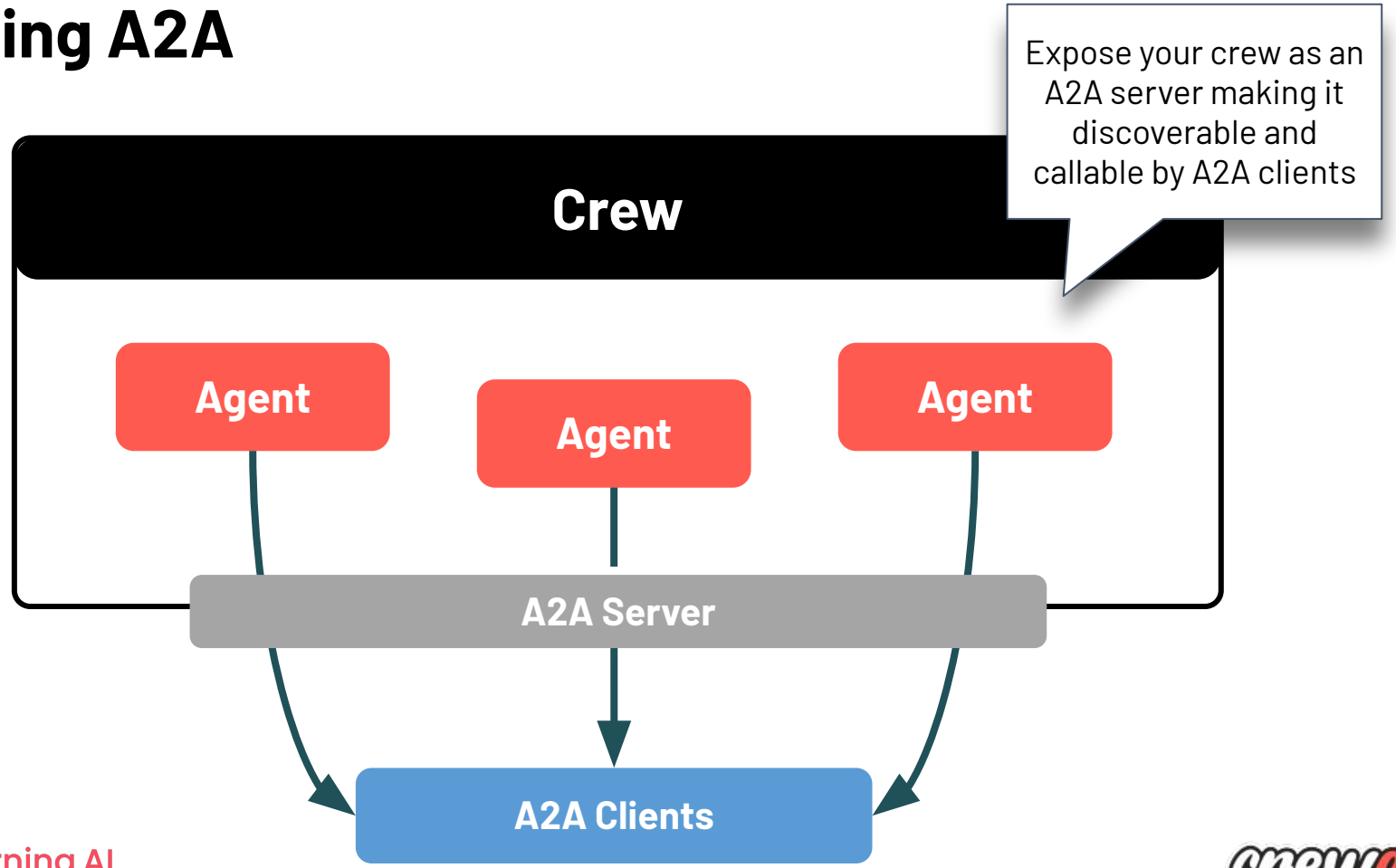
A2A

Stateful standard for interactions between different agents

Adopting A2A



Adopting A2A



Managing Systems of AI Agents

Orchestrating Agents with Flows

crewai

DeepLearning.AI

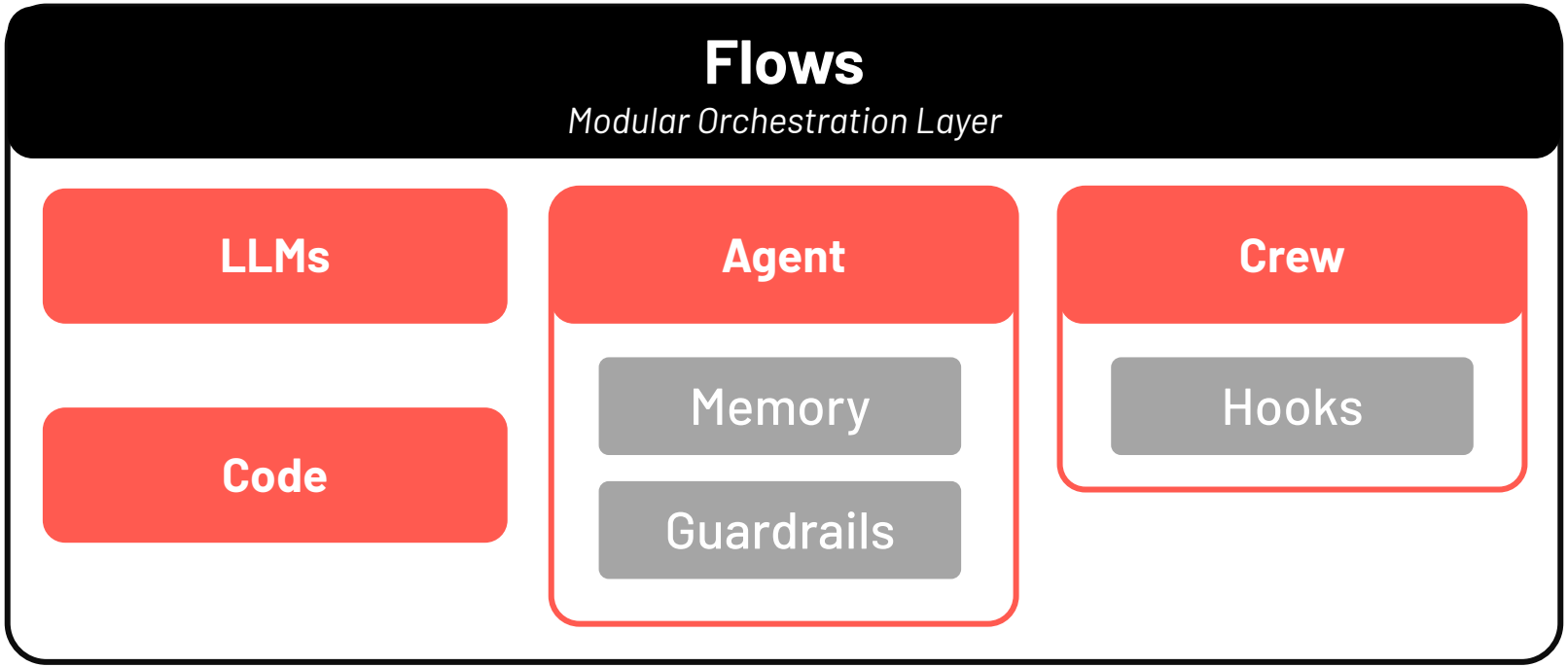
Mental Models for Agentic Systems

Agents
Real-time planning

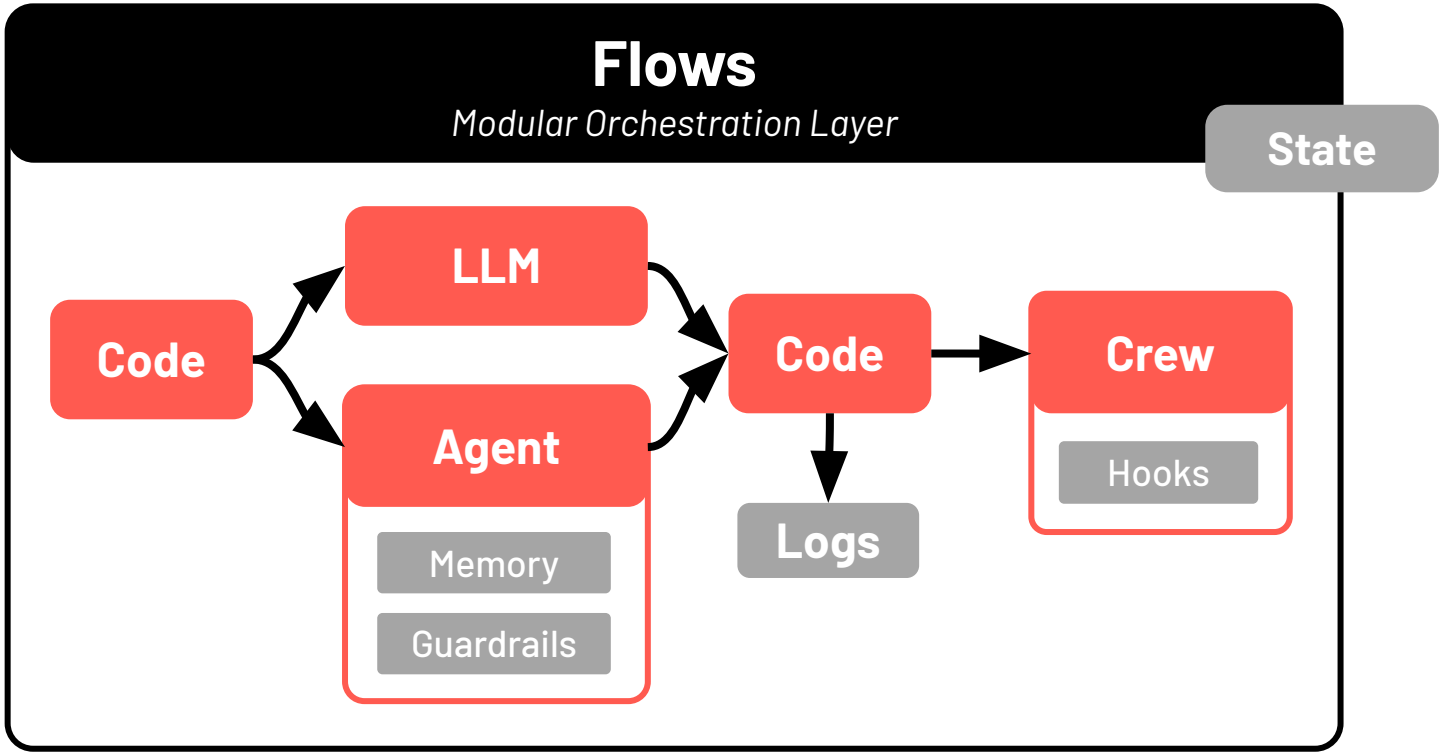
Graphs
Nodes and edges

Events
Trigger-based workflows

Flows

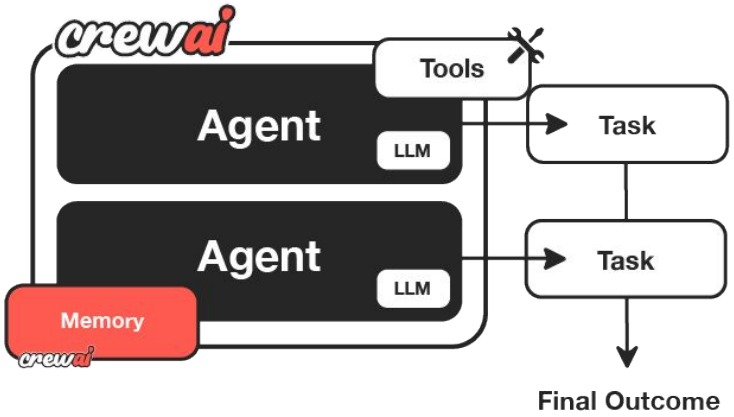

Flows
Modular Orchestration Layer
LLMs
Code
Agent
Memory
Guardrails
Crew
Hooks

Flows


Flows
Modular Orchestration Layer
State
LLM
Code
Agent
Memory
Guardrails
Code
Logs
Crew
Hooks

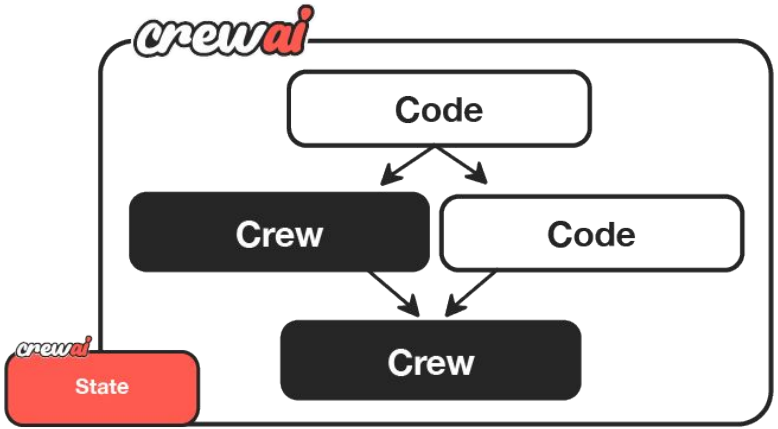
Crew

more agency



Flows

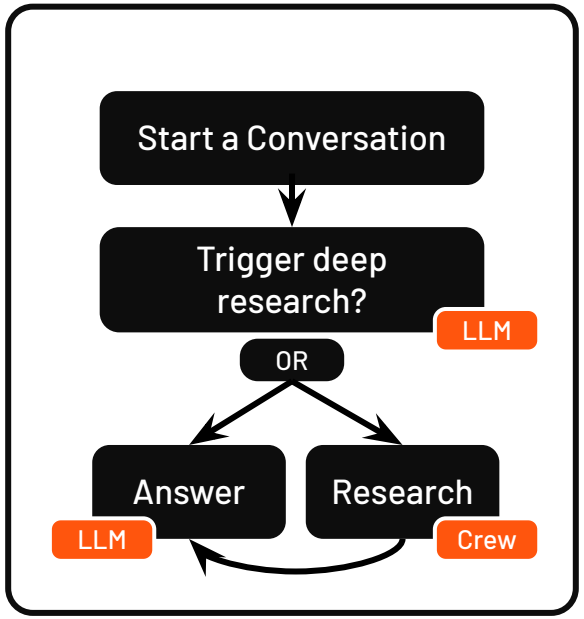
finer precision



Opt-In Agency Model

What is the minimum I can get away with for my use-case?

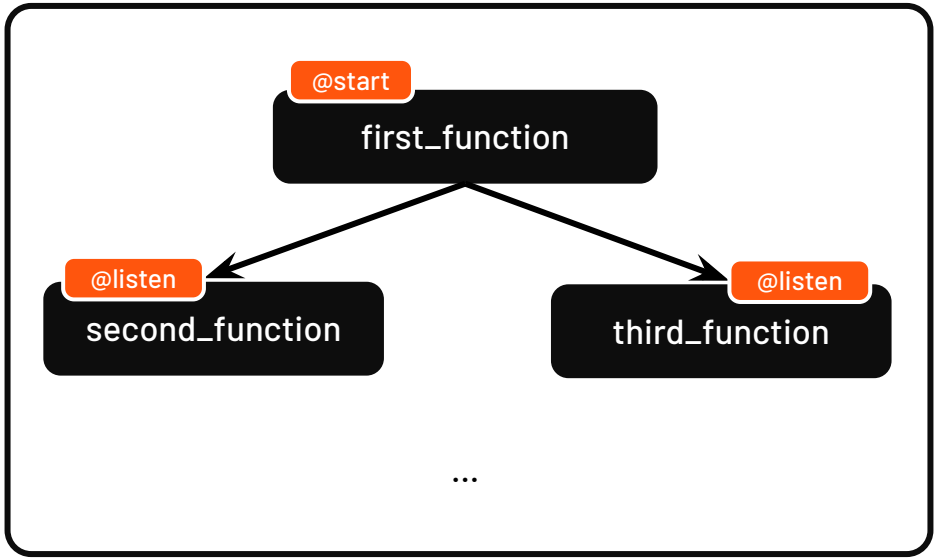
- Structured backbone
- Add agency according to the need
- Start with LLM and expand to full Crew

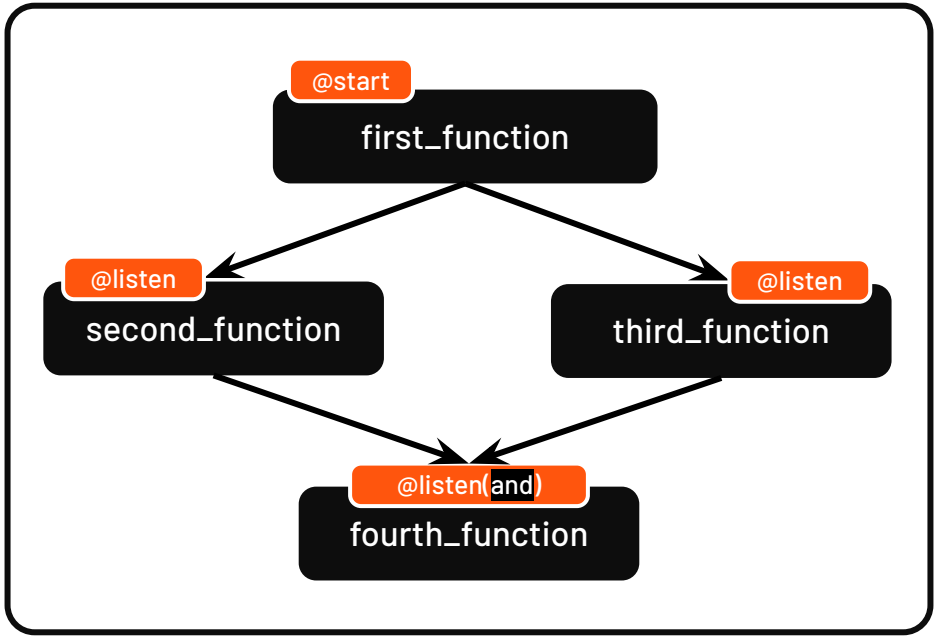

Start a Conversation
Trigger deep research?
LLM
OR
Answer
LLM
Research
Crew

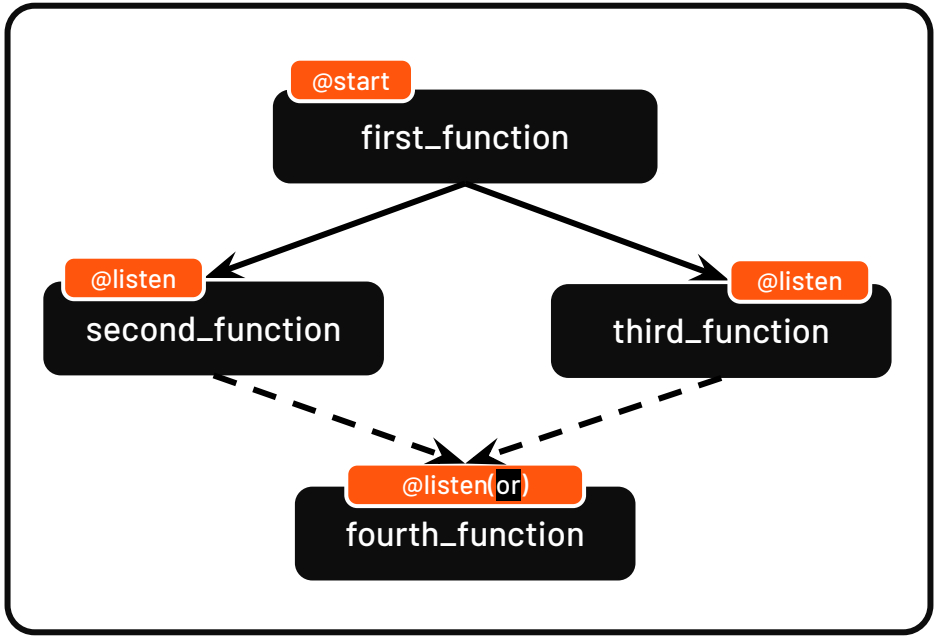
Building Blocks of Flows

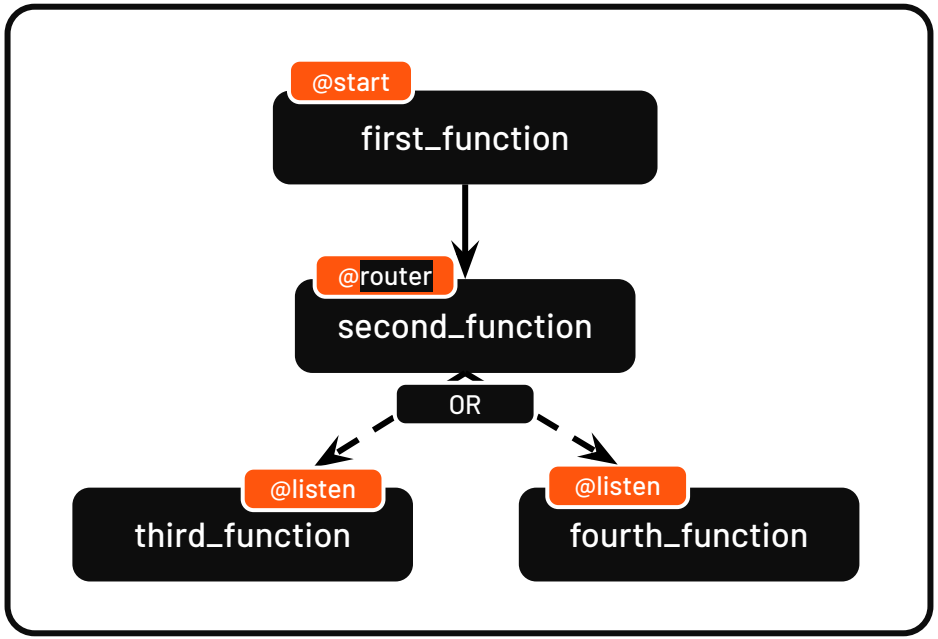
`@start`	`@listen`	`@router`
First function that is executed	Makes a method execute when the specified task generates an output	Defines conditional routing logic based on an input

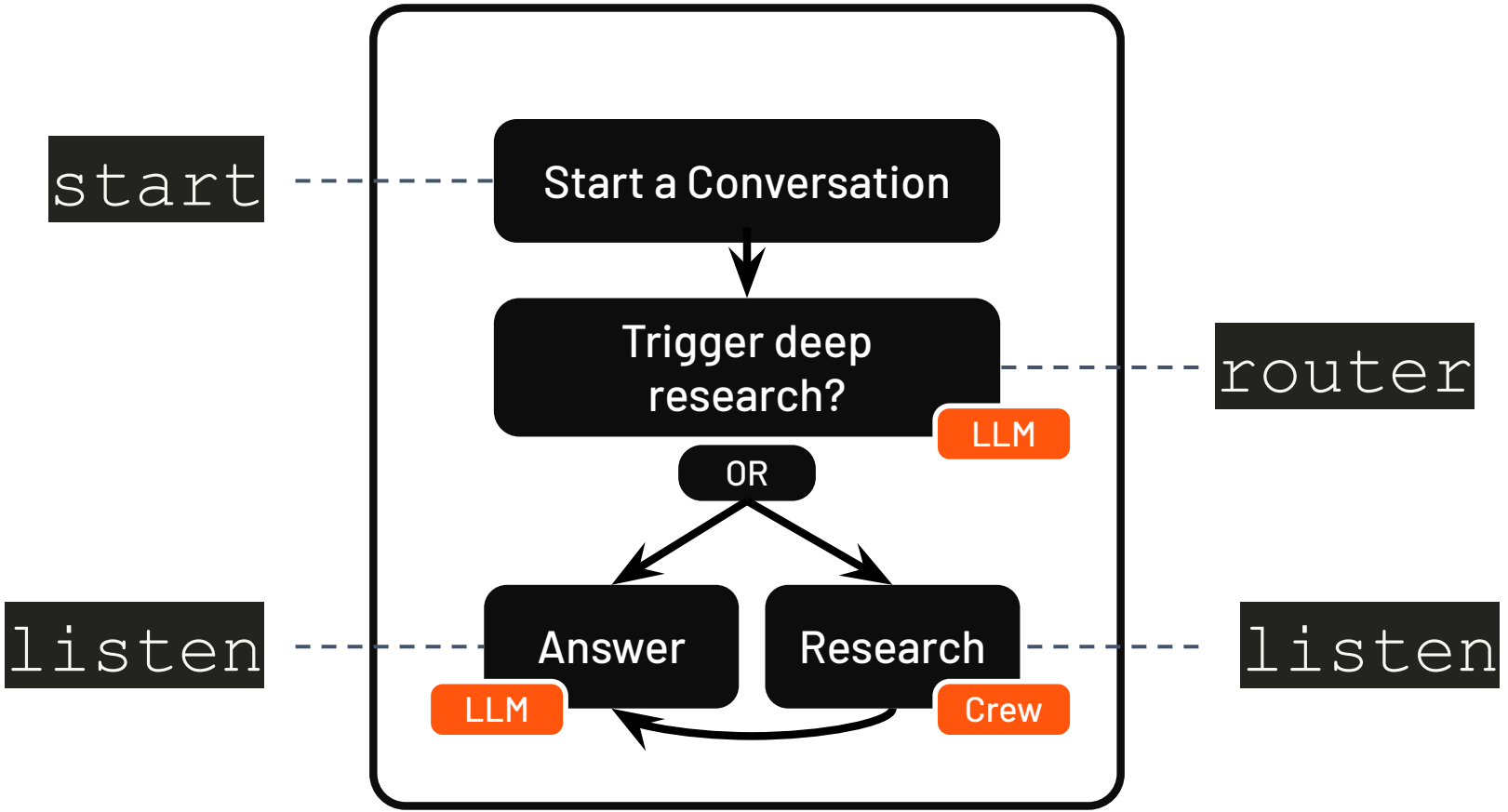
```
from crewai.flow.flow import listen, start

@start()
def start_conversation(self):
    print("Starting conversation")

@listen(start_conversation)
def trigger_research(self):
    print("Checking for research")
```


@start
first_function
@listen
second_function
@listen
third_function
...

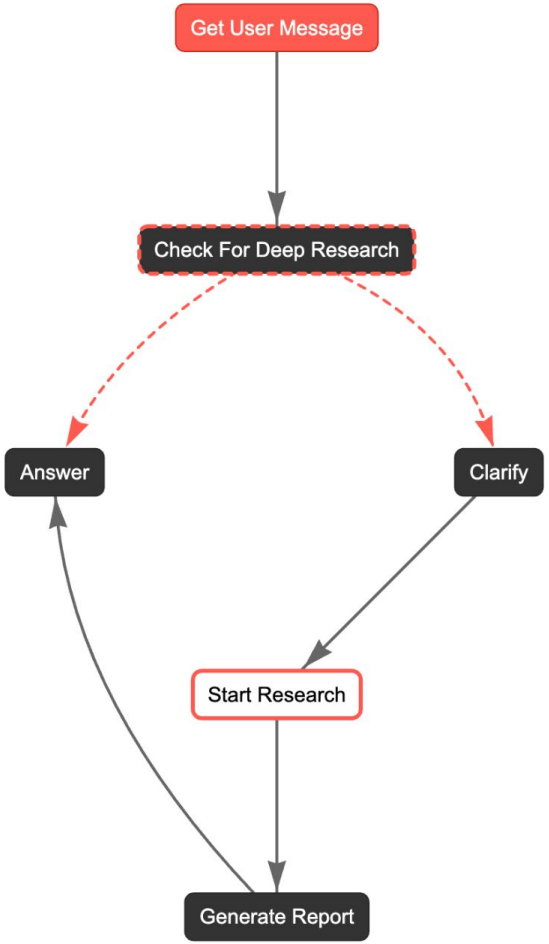

@start
first_function
@listen
second_function
@listen
third_function
@listen(and)
fourth_function


@start
first_function
@listen
second_function
@listen
third_function
@listen(or)
fourth_function


@start
first_function
@router
second_function
OR
@listen
third_function
@listen
fourth_function


start
Start a Conversation
Trigger deep research?
LLM
router
OR
listen
Answer
LLM
Research
Crew
listen

```
C:>
crewai flow plot
```



```
C:>
crewai create flow deep_research
```

```
+--- pyproject.toml
+--- tests
+--- README.md
+--- .gitignore
+--- .env
+--- src
    +--- deep_research
        +--- crews
        |   +--- research_crew
        |   |   +--- research_crew.py
        |   |   +--- config
        |   |   |   +--- agents.yaml
        |   |   |   +--- tasks.yaml
        |   |   +--- __init__.py
        +--- tools
        |   +--- __init__.py
        |   +--- custom_tool.py
        +--- __init__.py
        +--- main.py
```

Adding State to Flows

State provides **shared context** across each step of your flow

During Execution

- Each step of the flow consists of a function with access to state
- All functions can read / write state throughout execution
- Accumulated data in state can inform routing of the flow

After Execution

- Optionally you can persist state. Persistence store state for later use.
- Note persistent state of flows is different from the memory of crews in the flow
- Using persistence is especially important with conversational agents

Making State Persistent

Persistent state allows for flows that can be **paused**, **resumed**, and even **recover** after failures

Class Level Persistence

- When applied at the class level, the state updates and saves after every function execution in the flow

Method Level Persistence

- When applied at the method level, the state updates and saves after execution of functions with `@persist()` decorator

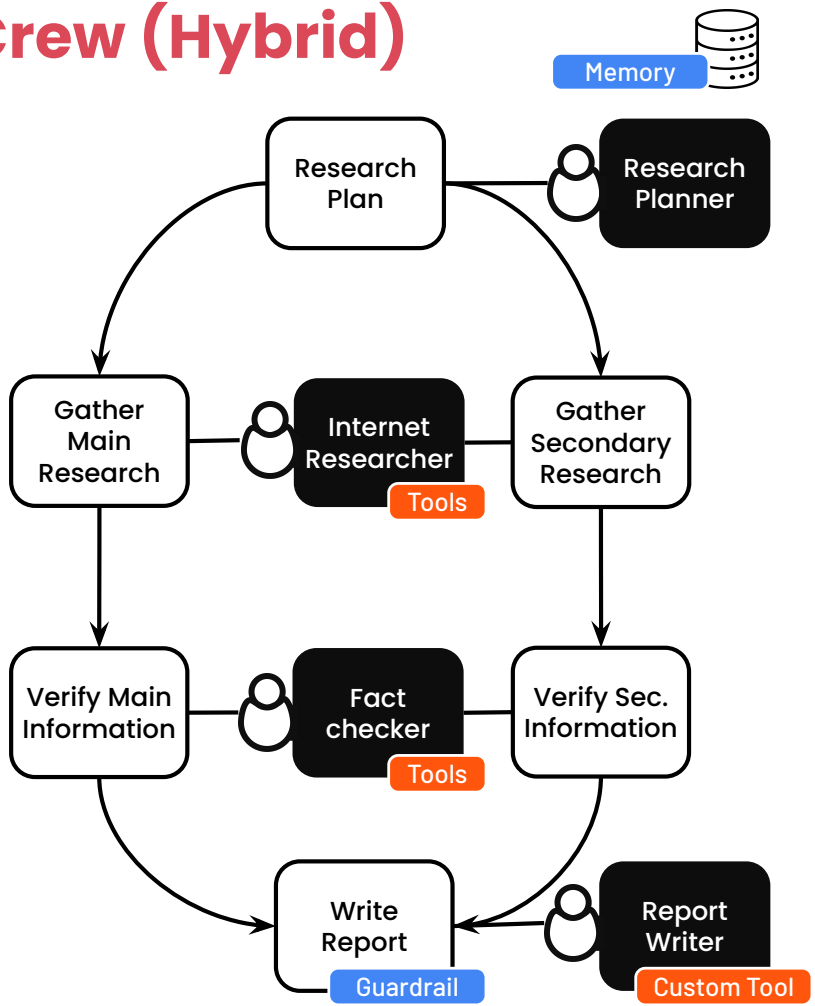
Managing Systems of AI Agents

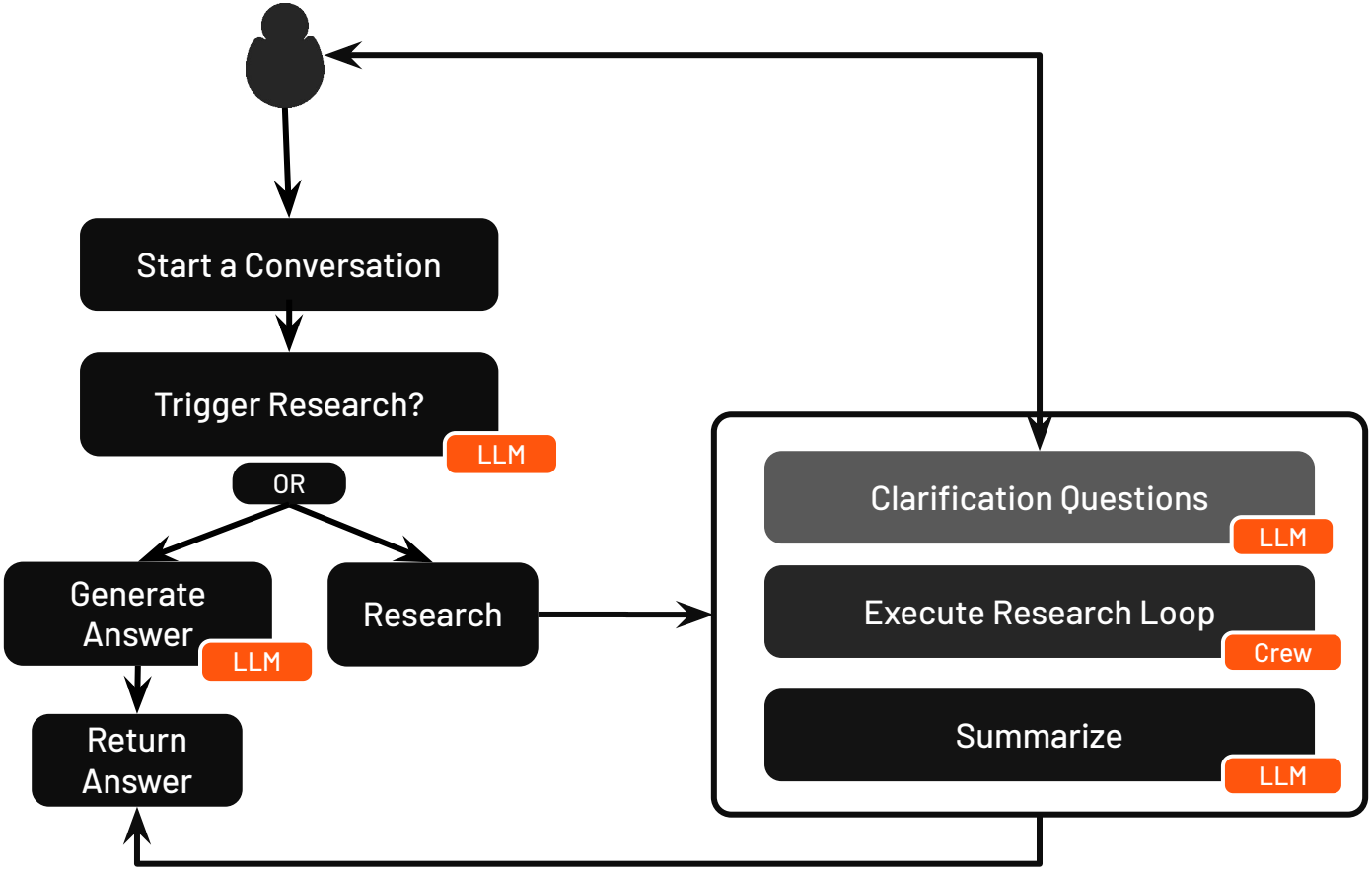
Building a Deep Research Flow

crewai

DeepLearning.AI

Deep Research Crew (Hybrid)




Start a Conversation
Trigger Research?
LLM
OR
Generate Answer
LLM
Research
Return Answer
Clarification Questions
LLM
Execute Research Loop
Crew
Summarize
LLM

Managing Systems of AI Agents

Tactics for Building Reliable Systems

crewai

DeepLearning.AI

You can build an agent

But can it run reliable every time?

More importantly: Can you trust it?

How to Build Agents you Trust

Including Deterministic Controls to Agentic system

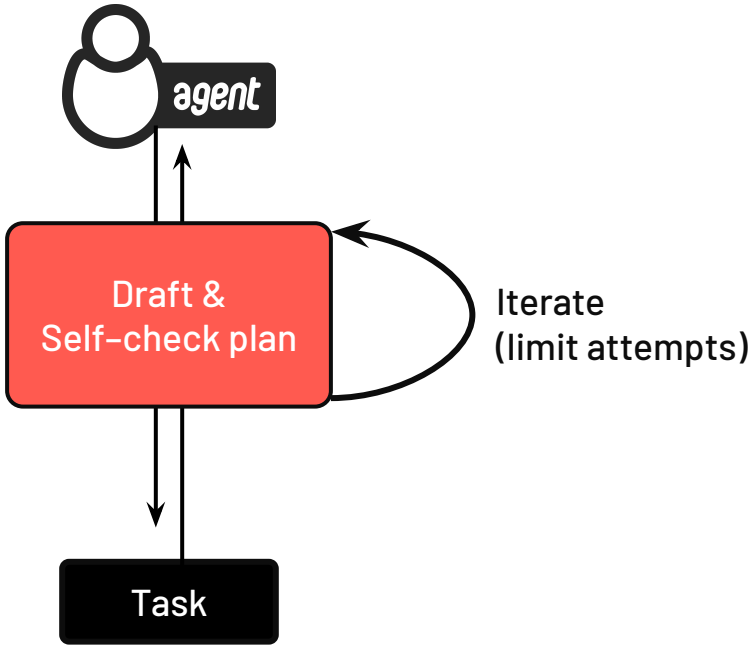
- Flows
- Guardrails
- Reasoning agents
- Human-in-the-loop oversight
- Testing
- Training
- Structured output
- Safe code execution

Adding reasoning

Requiring your agents to pre-plan

Turn on Reasoning at the Agent level to force a Pre-Plan before execution

Reasoning

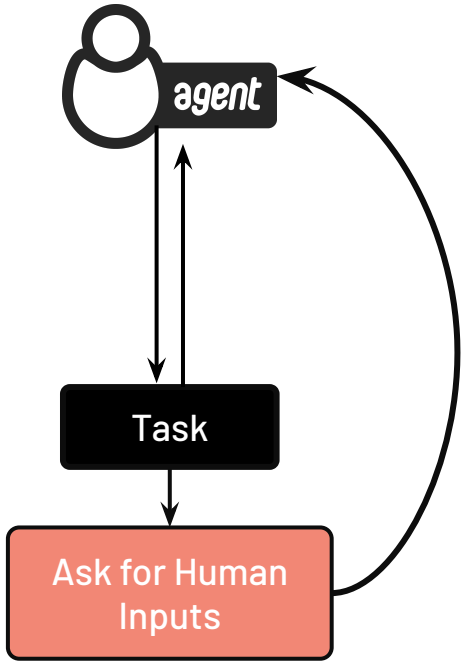


```python
from crewai import Agent

# Create an analyst agent with reasoning enabled
analyst = Agent(
    role="Data Analyst",
    goal="Find actionable insights from the data",
    backstory="Experienced at EDA and communication",
    reasoning=True,
    max_reasoning_attempts=3,
    verbose=True
)
```

Adding Human in the Loop

Strong code check for tasks outputs

Human in the Loop



Provide feedback to the agent

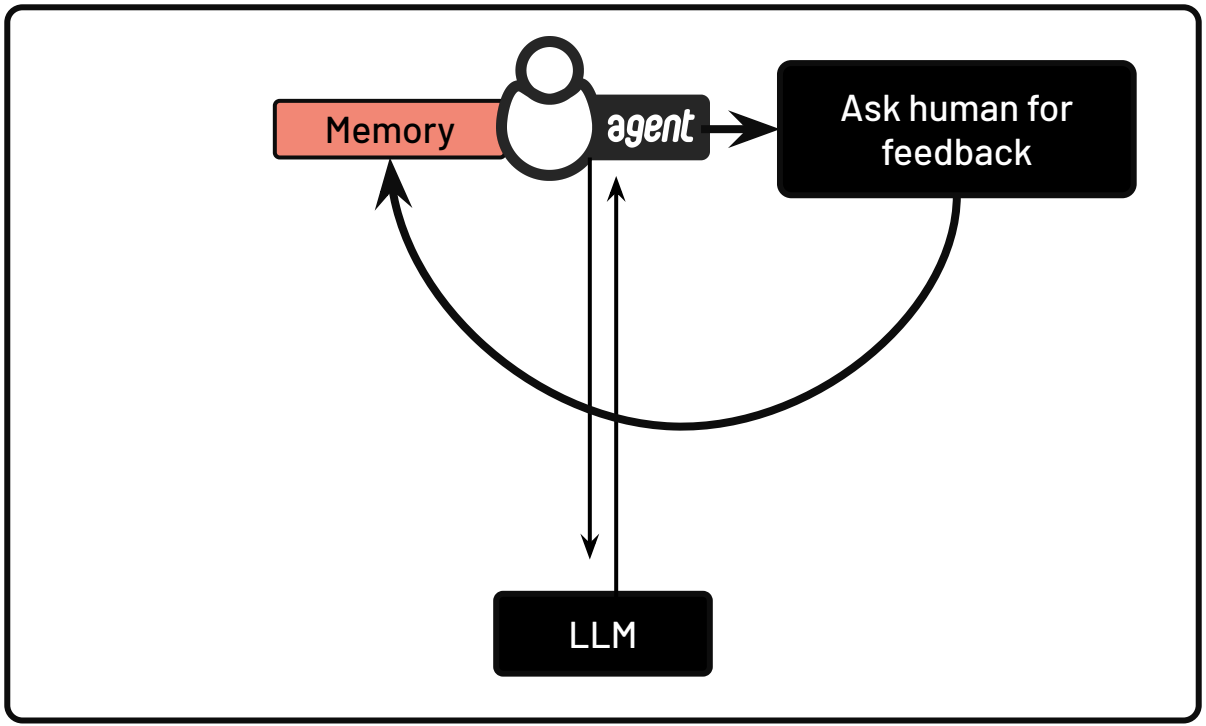
Human in the Loop

```
C:>
 =====
 ## HUMAN FEEDBACK: Provide feedback on the Final Result and Agent's actions.
 Please follow these guidelines:
  - If you are happy with the result, simply hit Enter without typing anything.
  - Otherwise, provide specific improvement requests.
  - You can provide multiple rounds of feedback until satisfied.
 =====
```

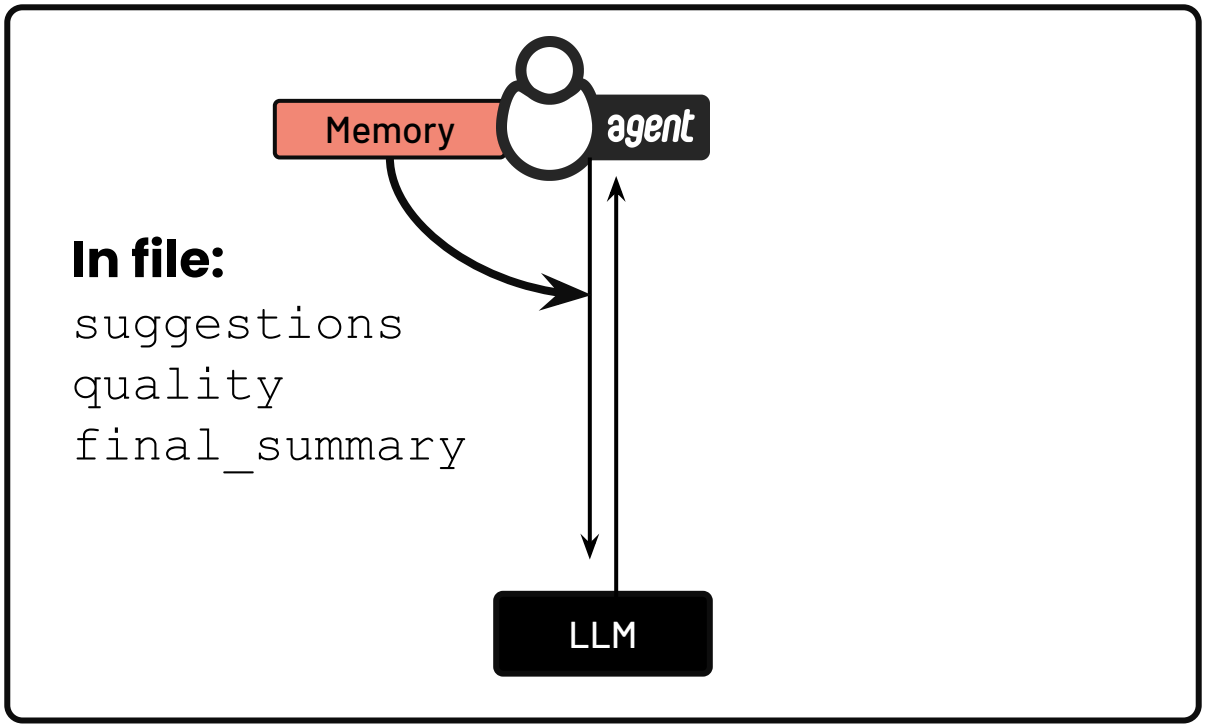
Training your crew

Building iterative feedback into memory

Training



Training


Memory
agent
In file:
suggestions
quality
final_summary
LLM

Training

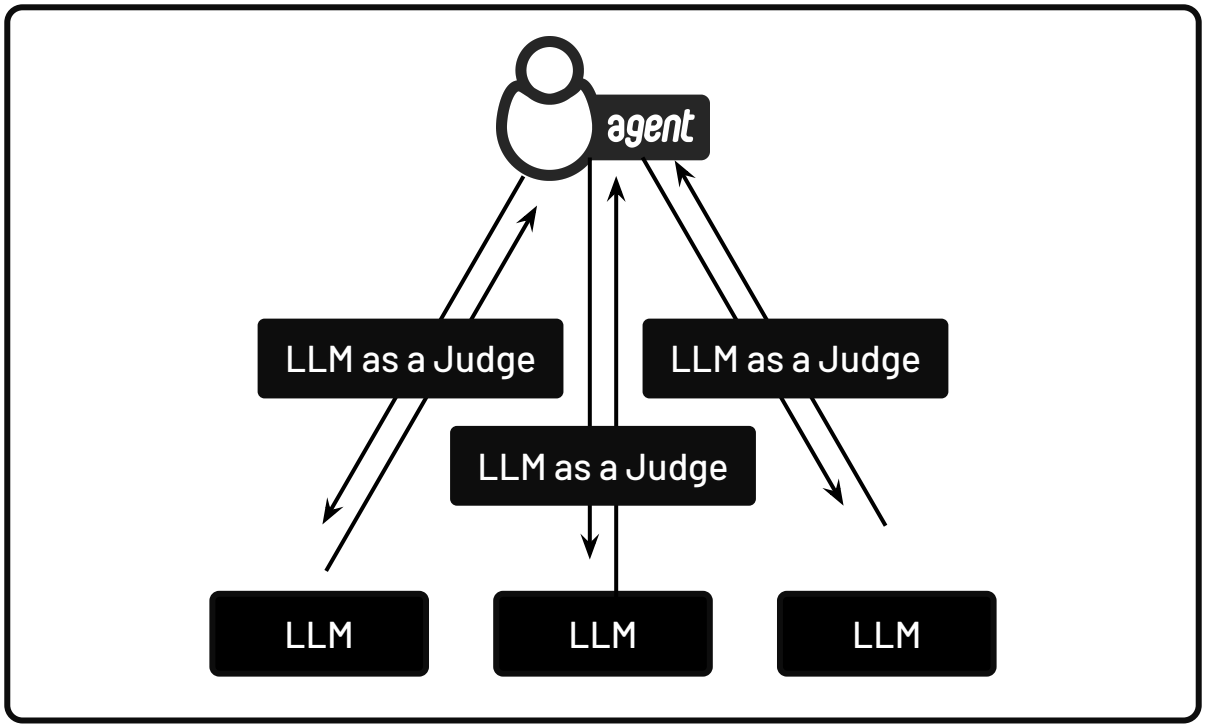
```
C:>
crewai train
```

```
DeepResearchFlow().crew().train(
    n_iterations=n_iterations,
    inputs=inputs,
    filename=filename
)
```

Running tests for your crew

Comparing performance of different LLMs

Testing

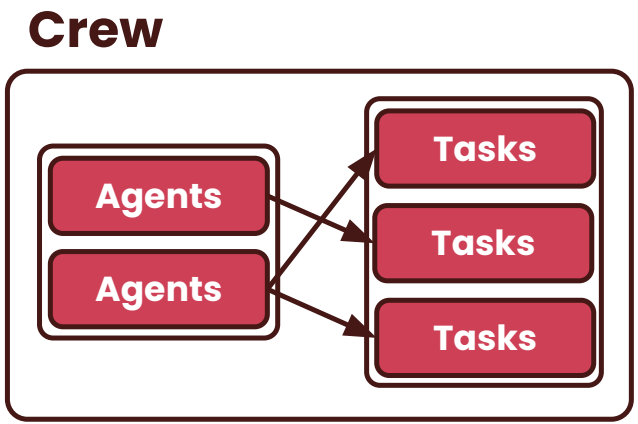


Testing

```
C:>
crewai test
```

```
DeepResearchFlow().crew().test(
    n_iterations=n_iterations,
    model=model_id,
)
```

Testing



Judge LLM

Tasks Scores
(1-10 Higher is better)

Tasks/Crew	Run 1	Run 2	Run 3	Avg. Total
Task 1	7.0	6.5	8.0	7.2
Task 2	5.5	7.0	6.0	6.2
Crew	7.0	5.0	6.5	6.3
Execution Time (s)	45	40	59	48

Safe code execution

Running code written by agents

Code Interpreter Tool

```
from crewai import Agent

programmer_agent = Agent(
    role="Python Programmer",
    goal="Write and execute Python code to solve problems",
    backstory="An expert Python programmer.",
    allow_code_execution=True,
    code_execution_mode="safe",
)
```

Reliable agents aren’t just accurate

they’re predictable, measurable, and recoverable

Managing Systems of AI Agents

Monitoring and Observability

crewai

DeepLearning.AI

How to Build Agents you Trust

Debuggability

Quality Monitoring

Data Governance

Protecting PII

Prompt Injection Safeguards

Secure Code Generation

Observability	Security	Compliance
Debuggability Quality Monitoring	Data Governance Prompt Injection Safeguards Secure Code Generation	Protecting PII

“You can’t fix what you can’t see.”

Tracing

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

# Define your first agent
researcher = Agent(
   role="AI LLMs Senior Data Researcher",
   goal="Uncover cutting-edge developments in AI and data science",
   backstory="""You work at a leading tech think tank.
   Your expertise lies in identifying emerging trends.
   You have a knack for dissecting complex data and presenting actionable insights.""",
   verbose=True,
   tools=[SerperDevTool()],
)
```

Tracing

```
# Define your second agent
writer = Agent(
    role="AI LLMs Reporting Analyst",
    goal="Craft compelling content on tech advancements",
    backstory="""You are a renowned Technical Writer, known for your insightful and engaging
articles.
    You transform complex concepts into compelling narratives.""",
    verbose=True,
)
```

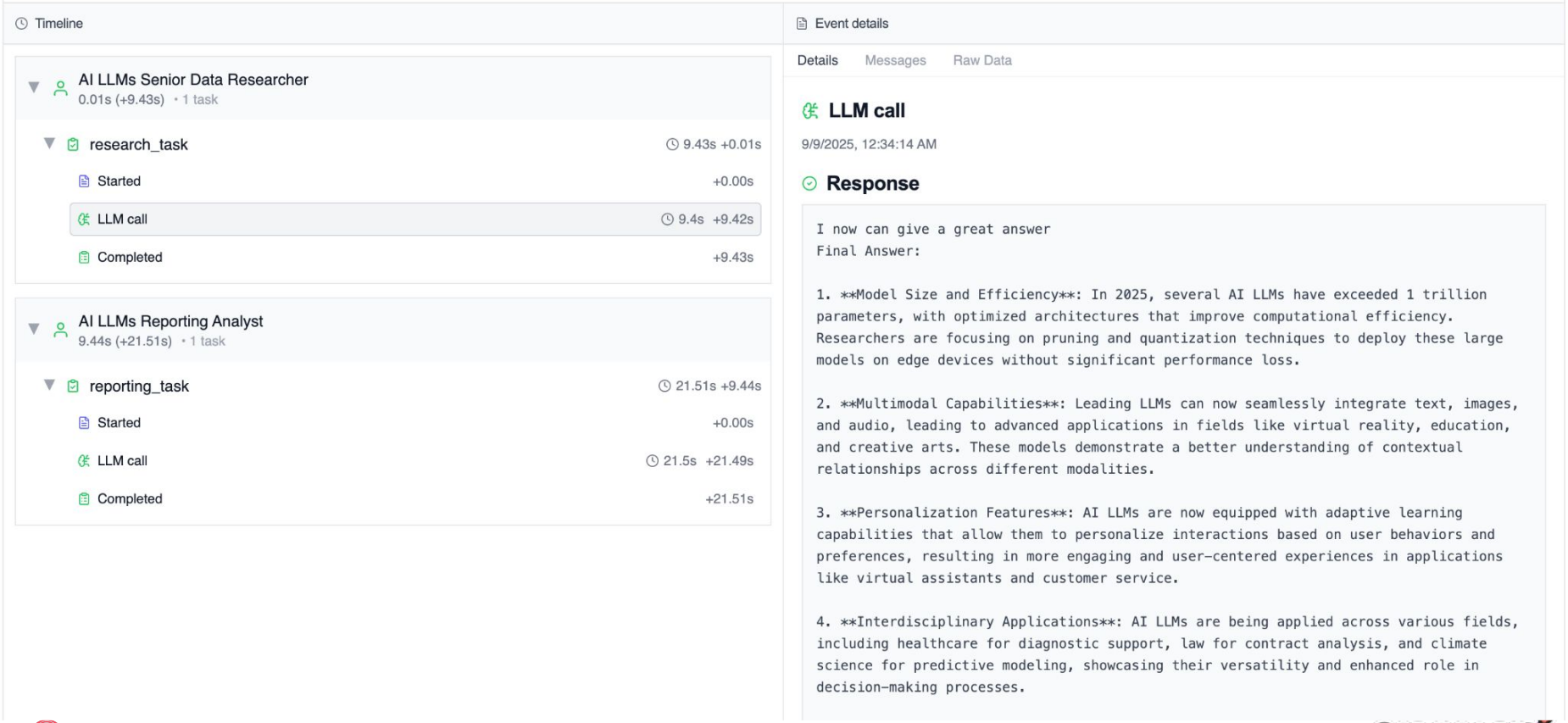
Tracing

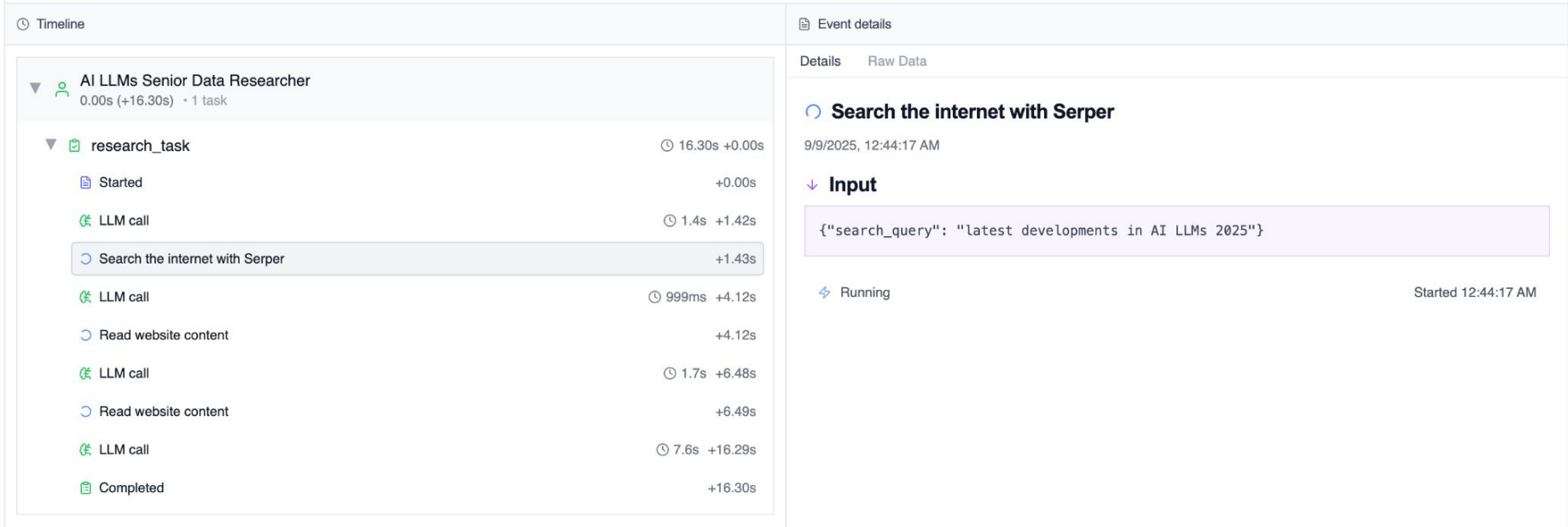
```
# Create tasks for your agents
research_task = Task(
    description="""Conduct a comprehensive analysis of the latest advancements in LLMs.
    Identify key trends, breakthrough technologies, and potential industry impacts.""",
    expected_output="Full analysis report in bullet points",
    agent=researcher,
)
reporting_task = Task(
    description="""Using the insights provided, develop an engaging white paper
    that highlights the most significant AI advancements.
    Your post should be informative yet accessible, catering to a tech-savvy audience.""",
    expected_output="white paper with at least 4 paragraphs",
    agent=writer,)
```

Tracing

```
# Enable tracing in your crew
crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, writing_task],
    process=Process.sequential,
    tracing=True,  # Enable built-in tracing
    verbose=True
)

# Execute your crew
result = crew.kickoff()
```


AI LLMs Senior Data Researcher
0.01s (+9.43s) • 1 task
research_task
9.43s +0.01s
Started
+0.00s
LLM call
9.4s +9.42s
Completed
+9.43s
AI LLMs Reporting Analyst
9.44s (+21.51s) • 1 task
reporting_task
21.51s +9.44s
Started
+0.00s
LLM call
21.5s +21.49s
Completed
+21.51s


Timeline
AI LLMs Senior Data Researcher
0.01s (+9.43s) • 1 task
research_task
9.43s +0.01s
Started
+0.00s
LLM call
9.4s +9.42s
Completed
+9.43s
AI LLMs Reporting Analyst
9.44s (+21.51s) • 1 task
reporting_task
21.51s +9.44s
Started
+0.00s
LLM call
21.5s +21.49s
Completed
+21.51s
Event details
Details
Messages
Raw Data
LLM call
9/9/2025, 12:34:14 AM
Response
I now can give a great answer
Final Answer:
1. **Model Size and Efficiency**: In 2025, several AI LLMs have exceeded 1 trillion parameters, with optimized architectures that improve computational efficiency. Researchers are focusing on pruning and quantization techniques to deploy these large models on edge devices without significant performance loss.
2. **Multimodal Capabilities**: Leading LLMs can now seamlessly integrate text, images, and audio, leading to advanced applications in fields like virtual reality, education, and creative arts. These models demonstrate a better understanding of contextual relationships across different modalities.
3. **Personalization Features**: AI LLMs are now equipped with adaptive learning capabilities that allow them to personalize interactions based on user behaviors and preferences, resulting in more engaging and user-centered experiences in applications like virtual assistants and customer service.
4. **Interdisciplinary Applications**: AI LLMs are being applied across various fields, including healthcare for diagnostic support, law for contract analysis, and climate science for predictive modeling, showcasing their versatility and enhanced role in decision-making processes.


Timeline
AI LLMs Senior Data Researcher
0.00s (+16.30s) • 1 task
research_task
16.30s +0.00s
Started
+0.00s
LLM call
1.4s +1.42s
Search the internet with Serper
+1.43s
LLM call
999ms +4.12s
Read website content
+4.12s
LLM call
1.7s +6.48s
Read website content
+6.49s
LLM call
7.6s +16.29s
Completed
+16.30s
Event details
Details
Raw Data
Search the internet with Serper
9/9/2025, 12:44:17 AM
Input
{"search_query": "latest developments in AI LLMs 2025"}
Running
Started 12:44:17 AM

Production Agents Operate at Machine Speed

So observability becomes reactive and proactive alerting becomes larger priority

Crew Quality Sampling

Sampling 10% of executions for quality checks.

Sampling Percentage

Select the portion of executions to be evaluated.

10%

Enable Quality Sampling

Toggle this feature to enable or disable quality sampling for this deployment.

Enabled

▼ Advanced thresholds

Performance Threshold (0-10)

7

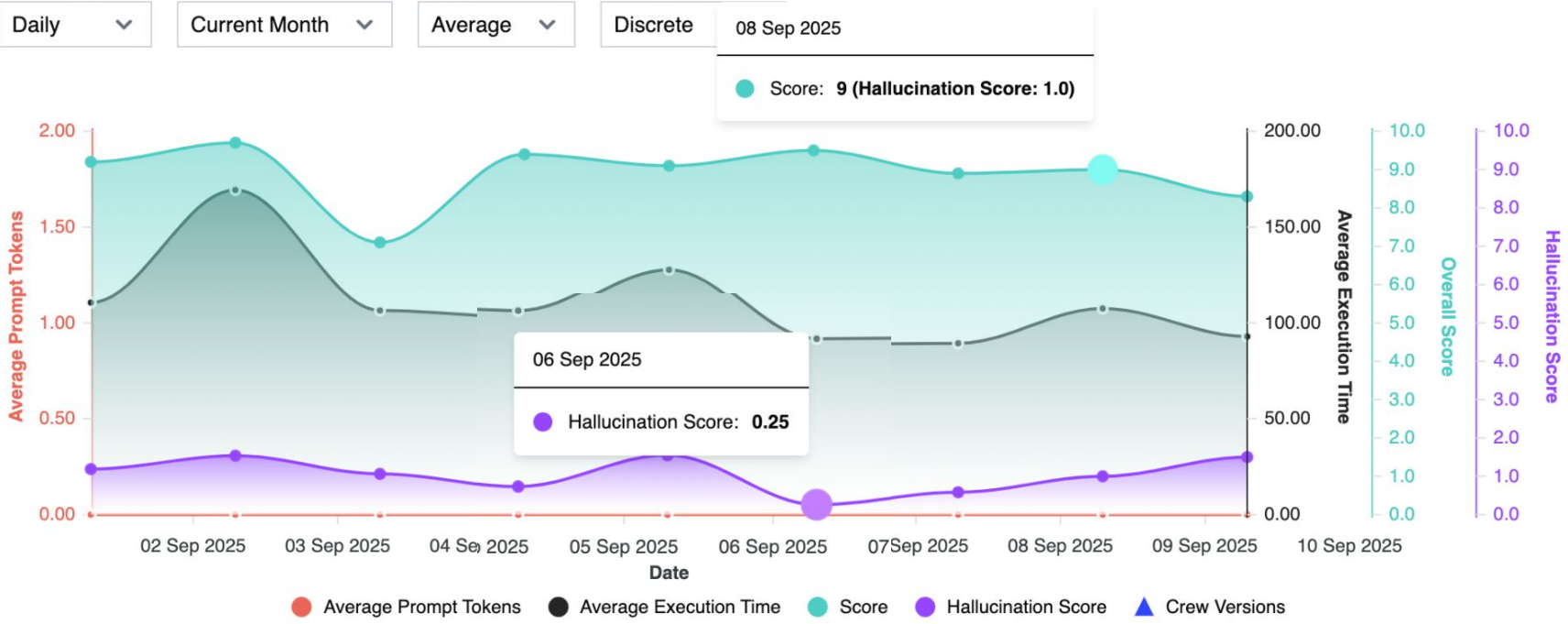
Max Execution Time (seconds)

3600

Tokens, Execution Time and Test Results

Aggregating prompt tokens and execution time. Displaying averaged daily test results.

Time Range: Daily | Period: Current Month | Aggregation: Average | Data type: Discrete



DeepLearning.AI

crewai

Managing Systems of AI Agents

CI/CD for Agents

crewai

DeepLearning.AI

CI & CD for Agentic Systems

- How can you support new models?
- How do these systems evolve over time?
- How do you ensure they are still working?

CI & CD for Agentic Systems

Map the Process

Implement First Version

Establish Validation Set

Implement Guardrails

Continuous Monitoring

Prompts

(Roles, Goals, Backstories, Descriptions and Expected Outputs)

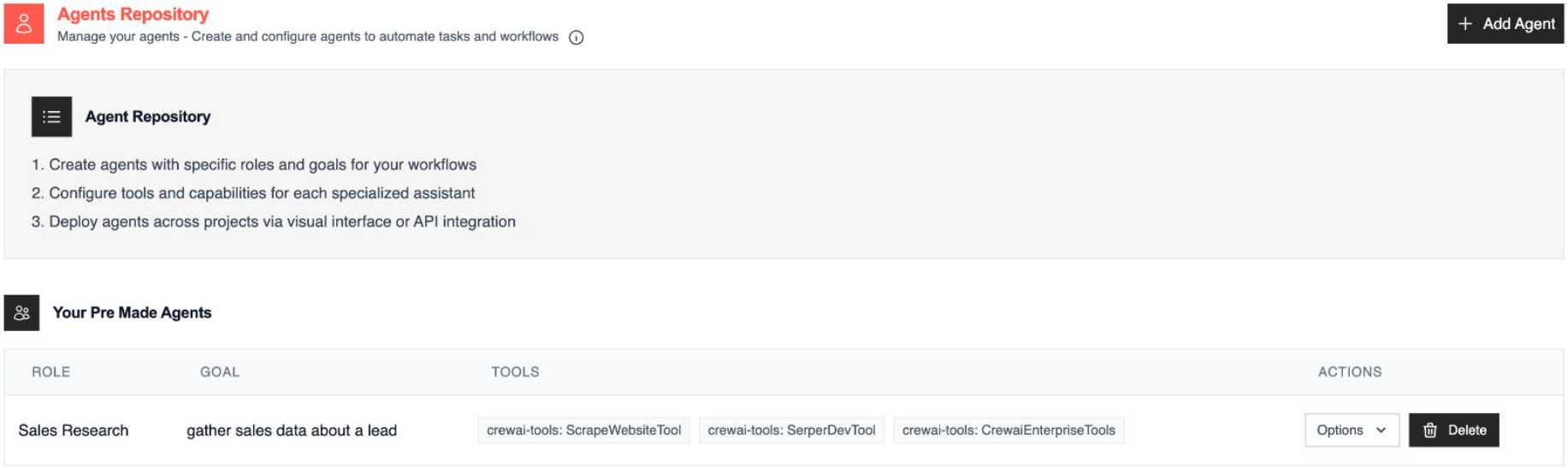
Agent Logic

(Custom tools, Hooks, Guardrails and such)

```
C:>
crewai create crew research_crew

C:>
cd research_crew
```

```
research_crew/
├── .gitignore
├── pyproject.toml
├── README.md
├── .env
└── src/
    └── research_crew/
        ├── __init__.py
        ├── main.py
        ├── crew.py
        ├── tools/
        │   ├── custom_tool.py
        │   └── __init__.py
        └── config/
            ├── agents.yaml
            └── tasks.yaml
```

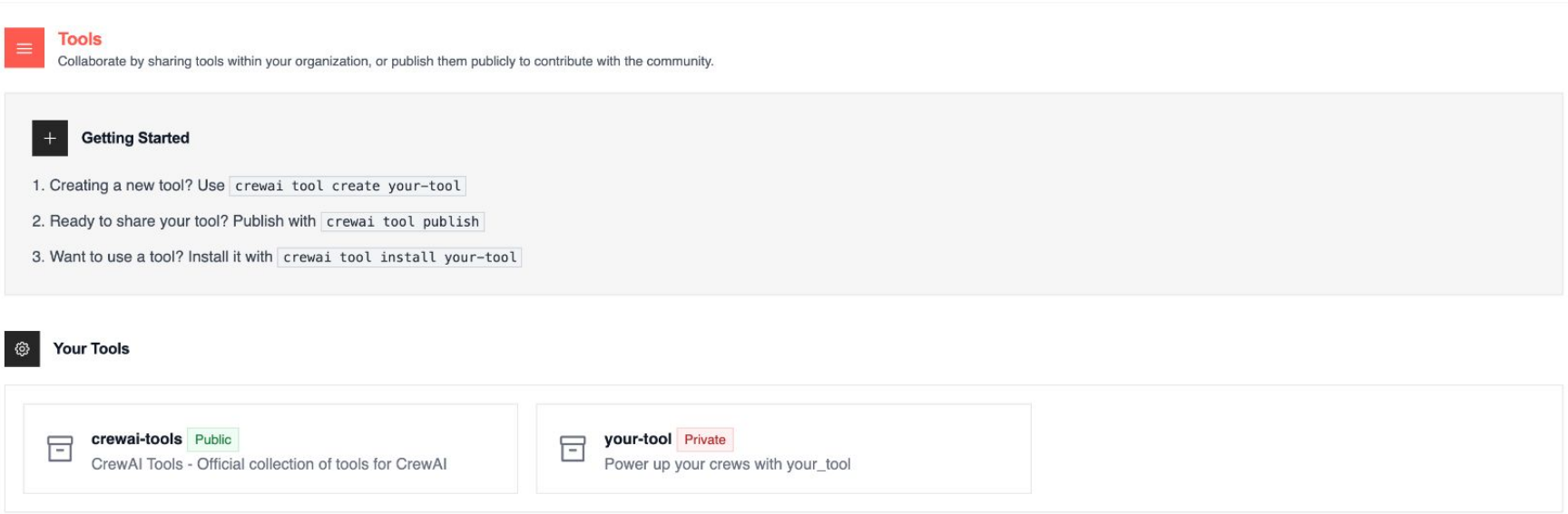
Agent Repository



Agent Repository

```
from crewai import Agent

agent = Agent(from_repository="sales-research")
```

Tool Repository



Don’t Overlook the Planning

1. Not spending time on planning use cases
2. Not clear definition of success
3. Not breaking the process into smaller chunks
4. Not measuring / evaluating