

Feb 16, 2025

Agentic Systems 101: Fundamentals, Building Blocks, and How to Build Them (Part B)

AI Agents Crash Course—Part 2 (with implementation).



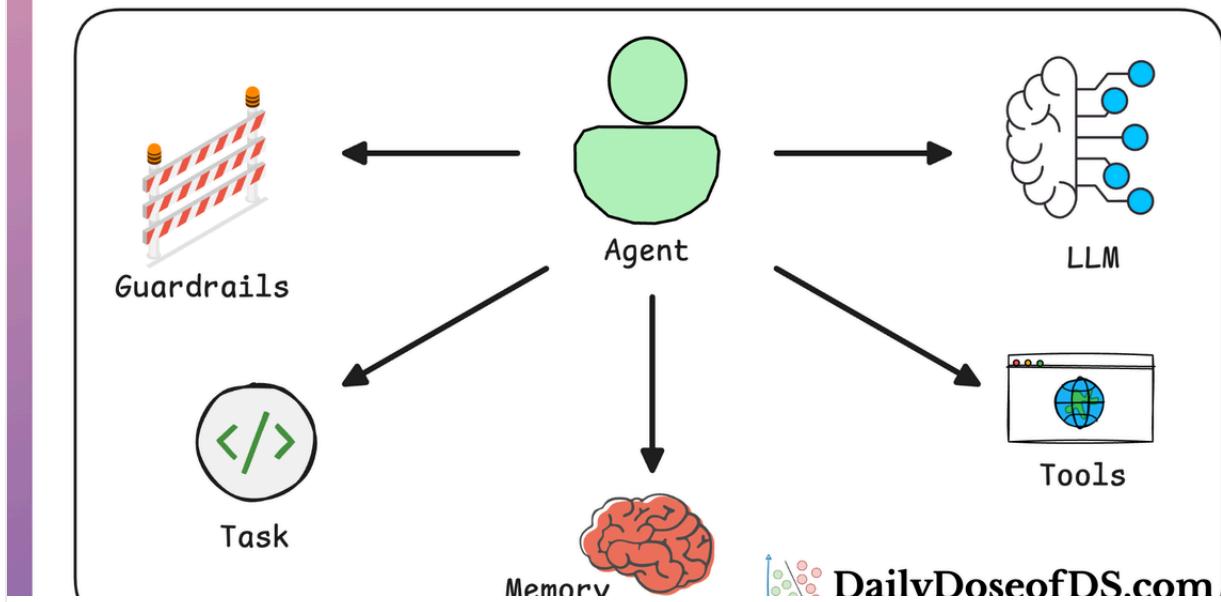
Avi Chawla, Akshay Pachaar

Introduction

In Part 1, we explored the fundamentals of Agentic systems:

- We saw how agents specialize in roles, collaborate dynamically, and adapt to real-time tasks, making them highly scalable and effective.
- We built single-agent and multi-agent systems to execute research tasks.
- We integrated tools into our Agents for better utility.
- We defined task dependencies and execution workflows to ensure logical processing.
- We experimented with YAML to decouple configuration from execution for better maintainability.

AI Agents Crash Course



AI Agents Crash Course—Part 1 (With Implementation)

A deep dive into the fundamentals of Agentic systems, building blocks, and how to build them.



Daily Dose of Data Science • Avi Chawla

Now that we've covered the fundamentals, it's time to deepen our understanding by implementing agentic systems as class modules, adding custom tools, handling structured outputs, comparing different execution strategies (sequential vs. hierarchical), and more.

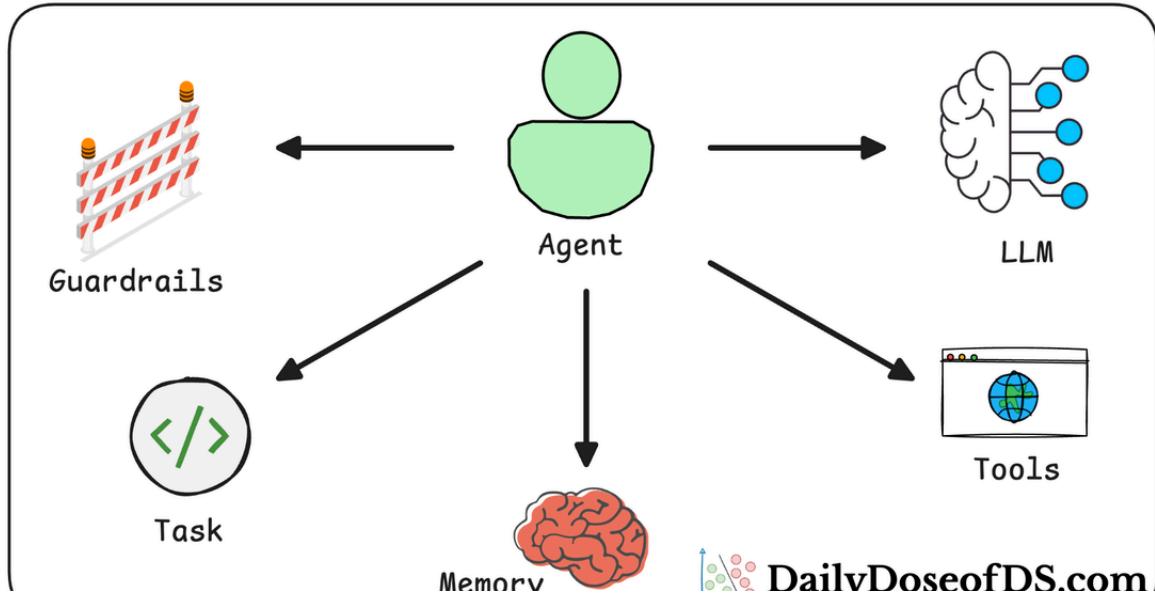
A quick recap

While AI agents are powerful, not every problem requires an agentic approach. Many tasks can be handled effectively with regular prompting or retrieval-augmented generation (RAG) solutions.

However, for problems where autonomy, adaptability, and decision-making are crucial, AI agents provide a structured way to build intelligent, goal-driven systems.

Here are three major motivations for building agentic systems:

AI Agents Crash Course



AI Agents Crash Course—Part 1 (With Implementation)

A deep dive into the fundamentals of Agentic systems, building blocks, and how to build them.

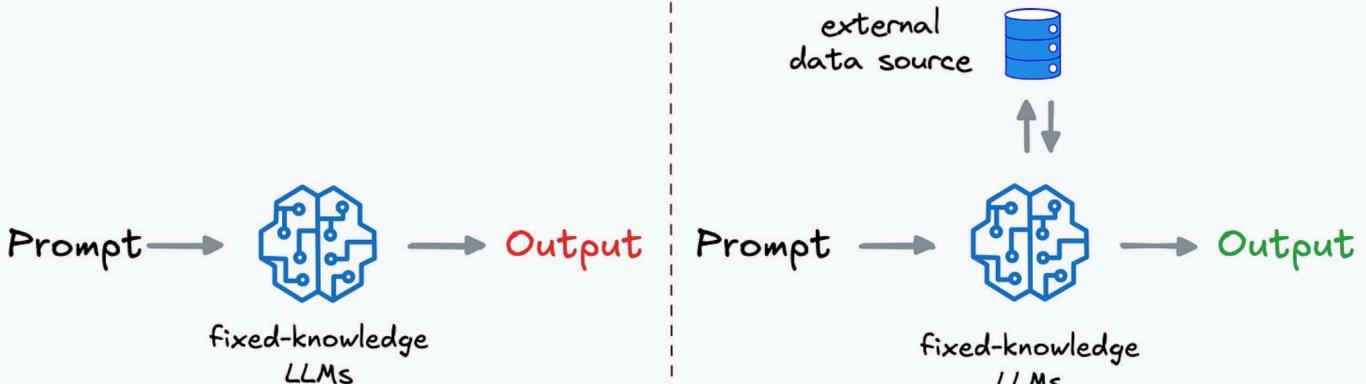


Daily Dose of Data Science • Avi Chawla

1) Beyond RAG

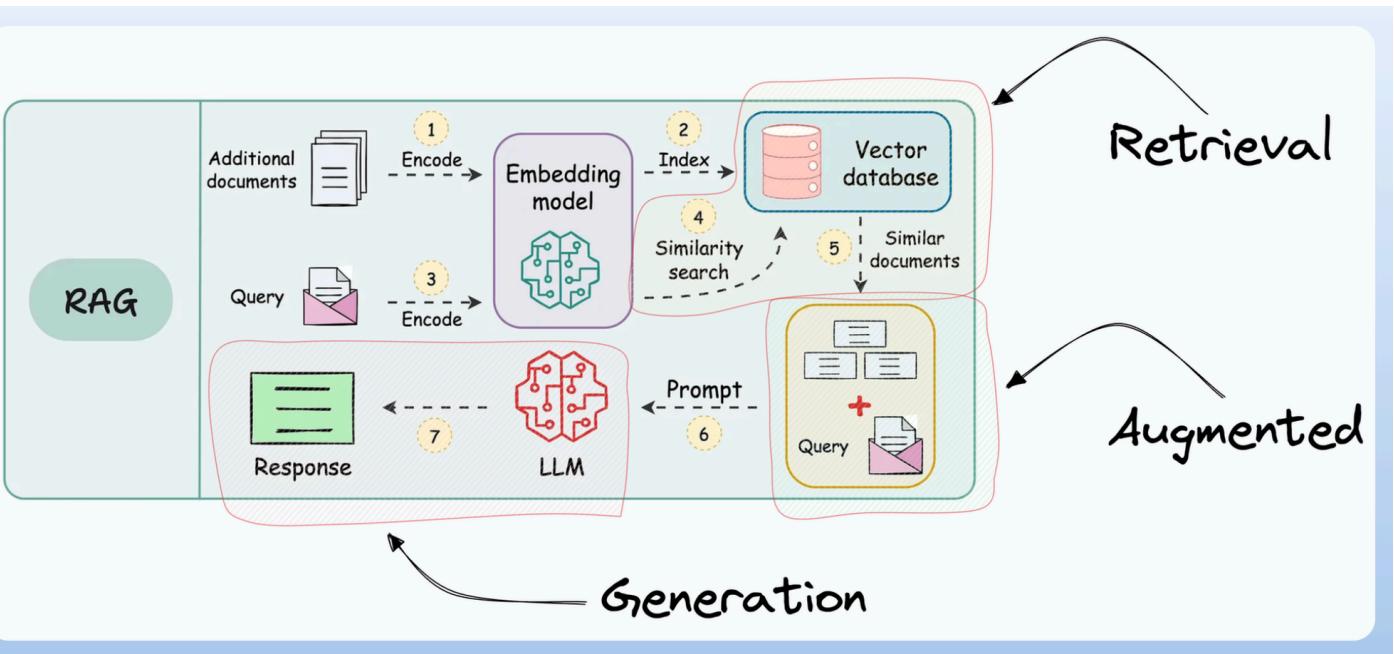
RAG was a major leap forward in how AI systems interact with external knowledge. It allowed models to retrieve relevant context from external sources rather than relying solely on static training data.

Limiting



However, most RAG implementations follow predefined steps—a programmer defines:

- Where to search (vector databases, documents, APIs)
- How to retrieve data (retrieval logic)
- How to construct the final response



This is not a limitation per se, but it does limit autonomy—the model doesn't dynamically figure out what to retrieve or how to refine queries.

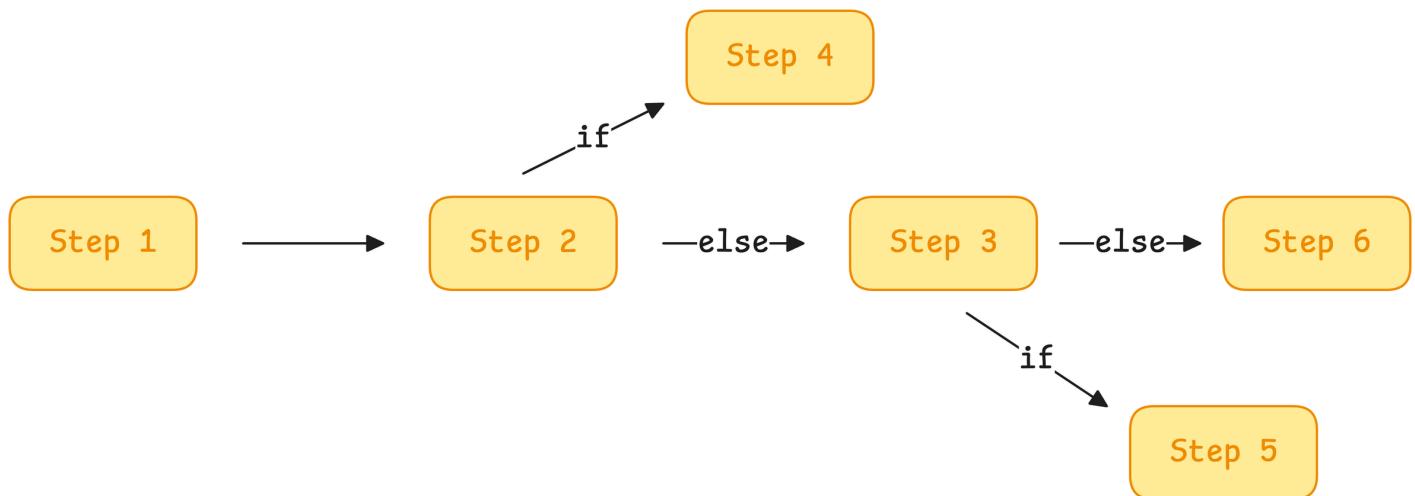
Agents take RAG a step further by enabling:

- Intelligent data retrieval—Agents decide where to search based on context.
- Context-aware filtering—They refine retrieved results before presenting them.
- Actionable decision-making—They analyze and take action based on the retrieved information.

2) Beyond traditional software development

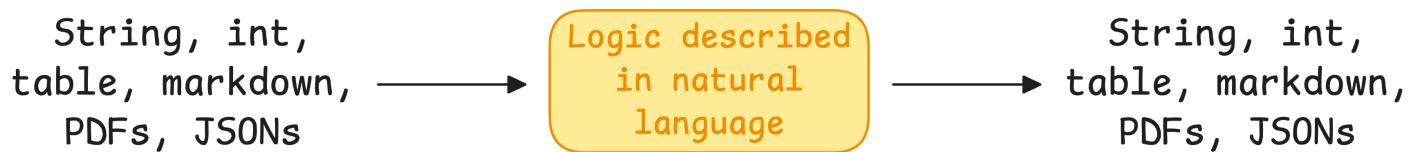
Traditional software is rule-based—developers hardcode every condition:

- IF A happens → Do X
- IF B happens → Do Y
- Else → Do Z



This works for well-defined processes but struggles with ambiguity and new, unseen scenarios.

AI agents break this rigidity by:

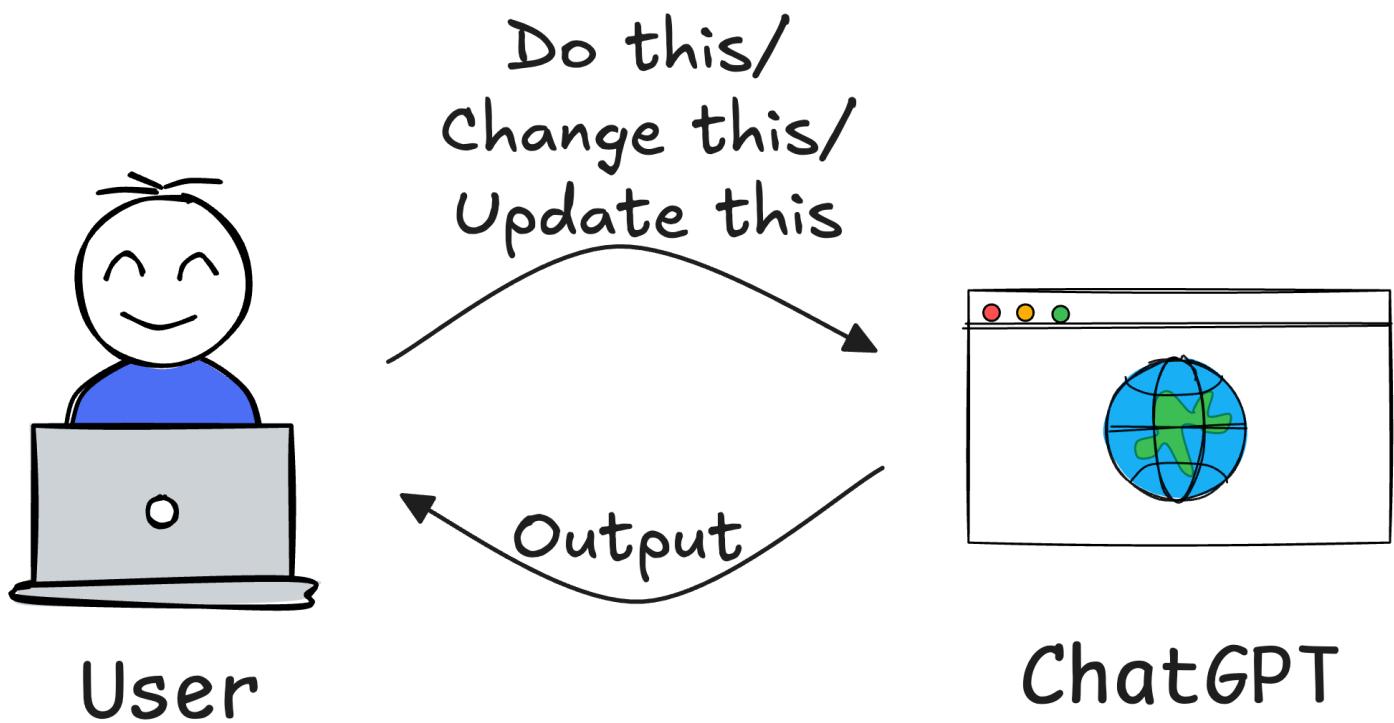


- Handling uncertain and dynamic environments (e.g., customer support, real-time analysis).
- Learning from interactions rather than following fixed rules.
- Using multiple modalities (text, images, numbers, etc.) in reasoning.

Due to the cognitive capabilities of LLMs, instead of writing hundreds of if-else conditions, AI agents learn and adapt their workflows in real-time.

3) Beyond human-interaction

One of the biggest challenges in AI-driven workflows is the need for constant human supervision. Traditional AI models—like ChatGPT—are reactive, meaning:



- You provide a query → Model generates a response.
- You review the response → Realize it needs refining.
- You tweak the query → Run the model again.
- Repeat until the output is satisfactory.

This cycle requires constant human intervention.

AI agents, however, can autonomously break down complex tasks into sub-tasks and execute them step by step without supervision.

Building Blocks of AI Agents

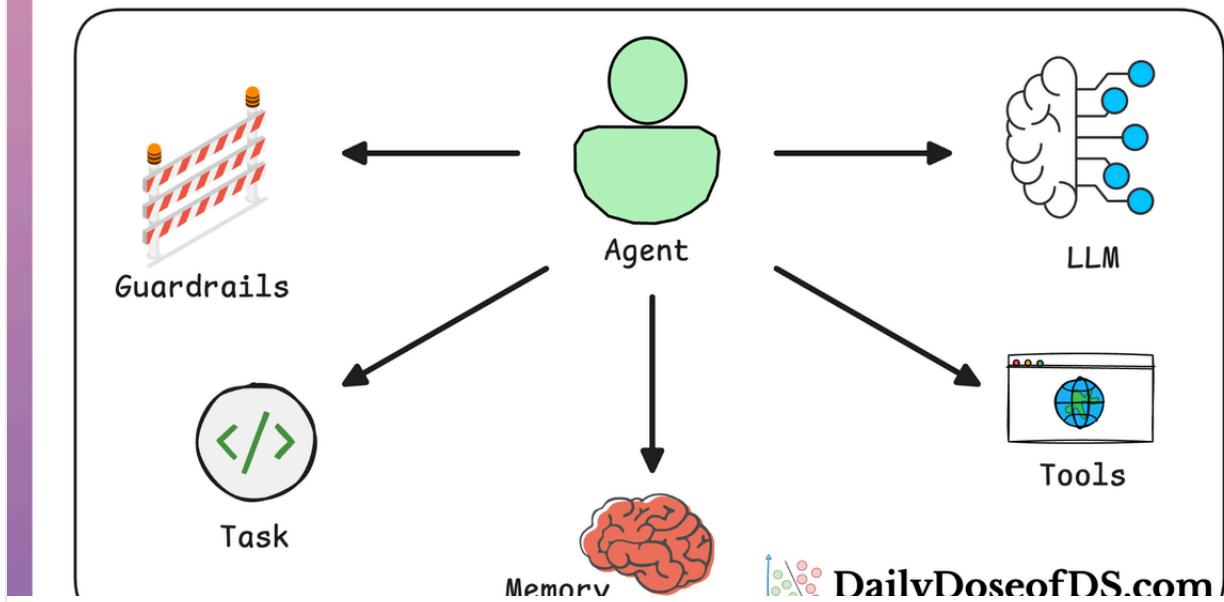
AI agents are designed to reason, plan, and take action autonomously. However, to be effective, they must be built with certain key principles in mind.

There are six essential building blocks that make AI agents more reliable, intelligent, and useful in real-world applications:

1. Role-playing
2. Focus
3. Tools
4. Cooperation
5. Guardrails
6. Memory

We won't cover them in detail once again since we have already done that in Part 1. If you haven't read it yet, we highly recommend doing so before reading further:

AI Agents Crash Course



AI Agents Crash Course—Part 1 (With Implementation)

A deep dive into the fundamentals of Agentic systems, building blocks, and how to build them.



Daily Dose of Data Science • Avi Chawla

Let's dive into the implementation now.

Building Agentic systems

Throughout this crash course, we shall be using CrewAI, an open-source framework that makes it seamless to orchestrate role-playing, set goals, integrate tools, bring any of the popular LLMs, etc., to build autonomous AI agents.



GitHub - crewAllInc/crewAI: Framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling complex tasks.

Framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling...

 GitHub • crewAllInc

To highlight more, CrewAI is a standalone independent framework without any dependencies on Langchain or other agent frameworks.

Let's dive in!

Setup

To get started, install CrewAI as follows:



n

Like the RAG crash course, we shall be using Ollama to serve LLMs locally. That said, CrewAI integrates with several LLM providers like:

- OpenAI
- Gemini
- Groq
- Azure
- Fireworks AI
- Cerebras
- SambaNova
- and many more.

 If you have an OpenAI API key, we recommend using that since the outputs may not make sense at times with weak LLMs. If you don't have an API key, you can get some credits by creating a dummy account on OpenAI and use that instead. If not, you can continue reading and use Ollama instead but the outputs could be poor in that case.

To set up OpenAI, create a `.env` file in the current directory and specify your OpenAI API key as follows:



.env

Also, here's a step-by-step guide on using Ollama:

- Go to [Ollama.com](https://ollama.com), select your operating system, and follow the instructions.



Blog Discord GitHub

Search models

Models Sign in

Download Ollama



macOS



Linux



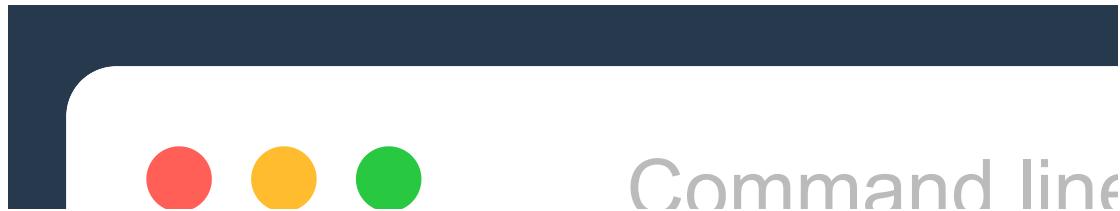
Windows

Install with one command:

```
curl -fsSL https://ollama.com/install.sh | sh
```

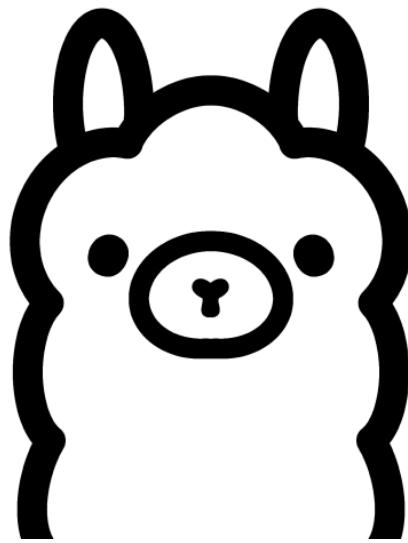
[View script source](#) • [Manual install instructions](#)

- If you are using Linux, you can run the following command:



Command line

- Ollama supports a bunch of models that are also listed in the model library:



library

Get up and running with large language models.



[Blog](#) [Discord](#) [GitHub](#)

Search models

[Models](#) [Sign in](#)

[Download](#)



Models

Filter by name...

Most popular ▾

llama3.2

Meta's Llama 3.2 goes small with 1B and 3B models.

[tools](#) [1b](#) [3b](#)

2.2M Pulls 63 Tags Updated 5 weeks ago

llama3.1

Llama 3.1 is a new state-of-the-art model from Meta available in 8B, 70B and 405B parameter sizes.

[tools](#) [8b](#) [70b](#) [405b](#)

8M Pulls 93 Tags Updated 7 weeks ago

gemma2

Google Gemma 2 is a high-performing and efficient model available in

Once you've found the model you're looking for, run this command in your terminal:

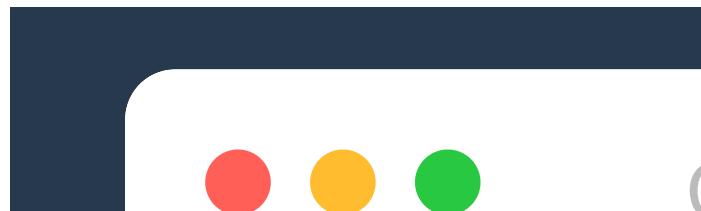


Command line

The above command will download the model locally, so give it some time to complete. But once it's done, you'll have Llama 3.2 3B running locally, as shown below which depicts Microsoft's Phi-3 served locally through Ollama:



That said, for our demo, we would be running Llama 3.2 1B model instead since it's smaller and will not take much memory:



Done!

Everything is set up now and we can move on to building our agents.

YAML config recap

- 💡 Feel free to skip this part and move to next section if you remember YAML config.

We ended Part 1 by learning about the YAML config.

More specifically, before learning about YAML config, we were defining agents—their roles, backstory, tasks, expected outputs—everything directly in Python.



```
from crewai import Agent, Task

research_agent = Agent(
    role="Internet Researcher",
    goal="Find the most relevant ai",
    backstory="""You are a skilled
```

This works well for prototyping and quick iteration. However, there's a downside to this approach:

- You are constantly interacting with the code, modifying agents, tweaking tasks, and updating configurations.
- This increases the risk of introducing errors into the pipeline, especially when transitioning from development to production.
- Maintaining large-scale workflows becomes harder as complexity increases.

To solve this, we decoupled the agent definitions, tasks, and workflows from the Python script by using YAML configuration files.

This approach allowed us to:

- Separate logic from configuration, making workflows more maintainable.
- Easily modify agents and tasks without changing the core execution logic.
- Version control YAML files for better traceability of workflow changes.

To do this, we first created a `config.yaml` file.



config.yaml

```
agents:  
    research_agent:  
        role: "Internet Researcher"  
        goal: "Find the most relevant and up-to-date i  
        backstory: "You are a skilled researcher with  
  
    summarization_agent:  
        role: "Content Summarizer"  
        goal: "Condense research findings into an easy  
        backstory: "You are an expert in breaking dow  
  
    fact_checker_agent:
```

The beauty of doing this was that now you could load this YAML file as a Python dictionary and access the exact descriptions, which can then be passed as arguments to the Agents and Tasks.

Also, notice closely that we have the "`{topic}`" parameter specified in the YAML file so that it gets populated later with values from your inputs when we kick off the Crew:



config.yaml

```
agents:  
  ...  
  
tasks:  
  research_task:  
    description: "... recent data on {topic}."  
    assigned_agent: "research_agent"  
    expected_output: "..."  
  
  ...
```

Once done, we loaded the YAML file as follows:



Printing the `config` dictionary, we got:

```

from pprint import pprint
pprint(config)

✓ 0.0s

{'agents': {'fact_checker_agent': {'backstory': "You specialize in detecting 'misinformation and validating 'claims using credible 'sources.'",
                                    'goal': "Verify research findings and 'ensure factual accuracy.'",
                                    'role': 'Fact-Checking Specialist'},
            'research_agent': {'backstory': "You are a skilled researcher with 'expertise in retrieving credible, 'real-time information from online 'sources.'",
                               'goal': "Find the most relevant and up-to-date 'information on a given topic.",
                               'role': 'Internet Researcher'},
            'summarization_agent': {'backstory': "You are an expert in 'breaking down complex 'information into clear, 'structured insights.'",
                                   'goal': "Condense research findings into 'an easy-to-read summary.",
                                   'role': 'Content Summarizer'}}},
'tasks': {'fact_checking_task': {'assigned_agent': 'fact_checker_agent',
                                 'depends_on': 'summarization_task',
                                 'description': "Cross-check the summarized 'information for accuracy and 'remove any misleading 'claims.",
                                 'expected_output': "A fact-checked and 'verified research 'summary."},
          'research_task': {'assigned_agent': 'research_agent',
                            'description': "Use the SerperDevTool to find the 'most relevant and recent data on '{topic}.",
                            'expected_output': "A detailed research report 'with key insights and source 'references."},
          'summarization_task': {'assigned_agent': 'summarization_agent',
                                 'depends_on': 'research_task',
                                 'description': "Summarize the research 'findings into a 'well-structured, concise 'report.",
                                 'expected_output': "A summary highlighting 'the key takeaways from 'the research."}}}

```

Since it's a nested dictionary, we accessed the details of any agent just like we would do in any other dictionary:



config.yaml

Next, we parameterized our entire multi-agent workflow by replacing the actual strings we defined earlier with placeholders from this config file.

For instance, here's what we did to research the Agent:



config.yaml

```
from crewai import Agent, Task

# Load YAML Configuration
with open("config.yaml", "r")
    config = yaml.safe_load(  
        config
```

Similarly, we defined our other two agents and their tasks as follows:



config.yaml

```
summarization_agent = Agent(  
    role=config["agents"]["sun"],  
    goal=config["agents"]["sun"],  
    backstory=config["agents"]["backstory"],  
    verbose=True  
)
```

```
fact_checker_agent = Agent(  
    role=config["agents"]["fact"],  
    goal=config["agents"]["fact"],  
    backstory=config["agents"]["backstory"],  
    verbose=True  
)
```

Finally, we had our Crew like before:



notebook.ipynb

```
from crewai import Crew, Process  
  
research_crew = Crew(  
    summarization_agent, fact_checker_agent)
```

And it works just like before, as depicted below:

Markdown ([result.raw](#))

✓ 0.0s

Python

The research findings on the impact of AI on job markets reveal several key insights:

1. **Job Transformation vs. Job Loss:** While there is concern that AI will lead to mass unemployment, the findings suggest that AI is more likely to transform existing jobs rather than eliminate them entirely. Many jobs will evolve, requiring workers to adapt to new technologies.
2. **Skill Shift:** There is a significant shift in required skills due to AI integration. Workers will need enhanced technological skills, and soft skills such as critical thinking and creativity will become increasingly important. Upskilling and reskilling initiatives are essential for workforce adaptation.
3. **New Job Creation:** AI is expected to create new job categories that do not currently exist. These jobs may arise in sectors such as AI system development, maintenance, and the ethical oversight of AI applications.
4. **Sector Variation:** The impact of AI varies across industries. Sectors like manufacturing and retail may see more automation, whereas healthcare and education are expected to augment human capabilities with AI, rather than replace workers outright.
5. **Economic Growth:** The integration of AI can lead to economic growth by improving productivity and efficiency. Companies that effectively utilize AI technologies may gain competitive advantages, potentially leading to job creation despite automation in specific areas.
6. **Policy Recommendations:** The report emphasizes the need for proactive policies. Governments and organizations should collaborate to ensure that workers are supported through training programs, and that social safety nets are in place to help those displaced by AI.
7. **Ethical Considerations:** There is a growing emphasis on the ethical implications of AI in the workplace. Businesses should establish guidelines to ensure fair treatment of workers and mitigate biases in AI systems.

In conclusion, while AI poses challenges to traditional job markets, it also presents opportunities for growth and innovation. Proactive adaptation strategies focusing on skill development and ethical practices are essential for harnessing the positive potential of AI.

Impressive, isn't it?

Building Modular Crews

There's one more way to make this even simpler, which is by defining modular Crews.

More specifically, one of the issues is that our code is quite cluttered with all the config key access, as you can see below from the agent definitions:



config.yaml

```
from crewai import Agent, Task  
  
# Load YAML Configuration  
with open("config.yaml", "r") as f:  
    config = yaml.safe_load(f)
```

...and also from the task definitions below:



config.yaml

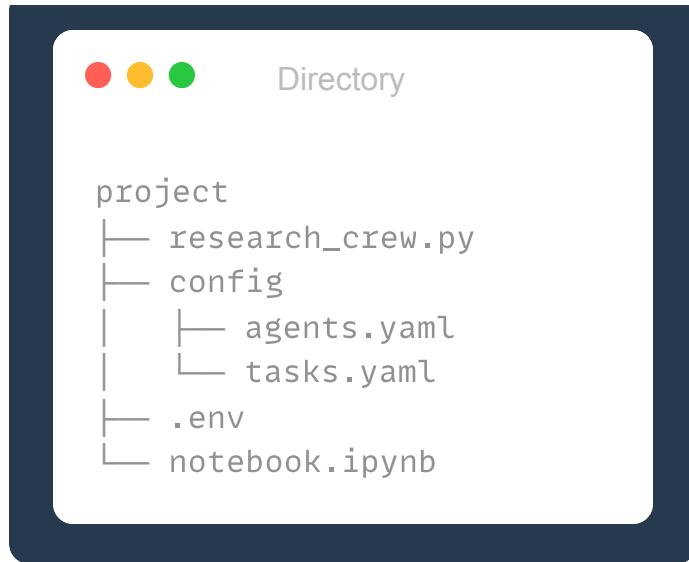
```
summarization_agent = Agent(  
    role=config["agents"]["sun"]  
    goal=config["agents"]["sun"]  
    backstory=config["agents"]  
    verbose=True  
)
```

```
fact_checker_agent = Agent(  
    role=config["agents"]["fac"]
```

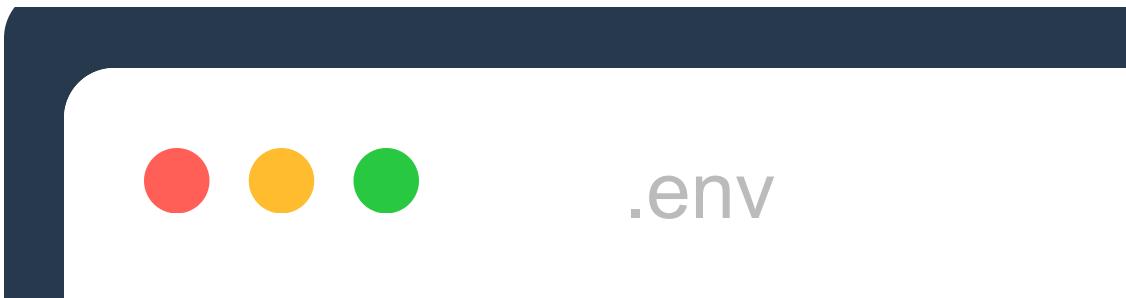


For the below demo to work, you must create a `.py` file and define the Crew there. We will import it in our Jupyter notebook.

First, ensure you have the following directory structure:



- We already have a `.env` file with the OpenAI API key and serper dev tool key:



- We create a `research_crew.py` file and we'll define our Crew here shortly.
- We also create a `config` folder with two files:
 - `agents.yaml` where we will add agent-specific config details.
 - `tasks.yaml` where we will add agent-specific config details.

Let's start with the `config` folder first.

Below, we define the `agents.yaml` file in the `config` folder with the details for every agent, like role, goal, backstory, etc.:



config/agents.yaml

```
research_agent:
  role: "Internet Researcher"
  goal: >
    Find the most relevant and
    up-to-date information on a given topic.
  backstory: >
    You are a skilled researcher with expertise in
    retrieving credible, real-time information from online sources.
  verbose: true

summarization_agent:
  role: "Content Summarizer"
  goal: >
    Condense research findings into an easy-to-read summary.
  backstory: >
    You are an expert in breaking down complex
    information into clear, structured insights.
  verbose: true

fact_checker_agent:
  role: "Fact-Checking Specialist"
  goal: >
    Verify research findings and ensure factual accuracy.
  backstory: >
    You specialize in detecting misinformation and
    validating claims using credible sources.
  verbose: true
```

Next, we define the `tasks.yaml` file in the `config` folder with the details for every agent, like role, goal, backstory, etc.:



config/tasks.yaml

```
research_task:
  description: >
    Use the SerperDevTool to find the most relevant and
    recent data on {topic}.
  agent: research_agent
  expected_output: >
    A detailed research report with key insights and
    source references.
  verbose: true

summarization_task:
  description: >
    Summarize the research findings into a well-structured,
    concise report.
  agent: summarization_agent
  expected_output: >
    A summary highlighting the key takeaways from the research.
  verbose: true

fact_checking_task:
  description: >
    Cross-check the summarized information for accuracy and
    remove any misleading claims.
  agent: fact_checker_agent
  expected_output: >
    A fact-checked and verified research summary.
  verbose: true
```

Now, instead of manually creating agents and tasks, we define a class that dynamically loads them from YAML.

We start with some basic imports:



research_crew.py

```
from crewai import Agent, Crew, Task, Process
from crewai.project import CrewBase, agent, task, crew
from crewai_tools import SerperDevTool
```

Here, we:

- Import Agent, Task, and Crew from `crewai`.
- Import `CrewBase` (a decorator that allows defining modular Crew classes).
- Import `SerperDevTool` (for web searches).

Next, we wrap everything inside a single class that dynamically loads agents and tasks.



The screenshot shows a Mac OS X window titled "research_crew.py". The window contains Python code for a modular Crew class named "ResearchCrew". The code includes a docstring, configuration variables for agent and task YAML files, and an __init__ method that initializes a SerperDevTool object.

```
@CrewBase
class ResearchCrew:
    """A crew for conducting research,
    summarizing findings, and fact-checking"""

    agents_config = 'config/agents.yaml'
    tasks_config = 'config/tasks.yaml'

    def __init__(self):
        self.search_tool = SerperDevTool()
```

- `@CrewBase`—This decorator tells CrewAI that ResearchCrew is a modular Crew class.
- `agents_config = 'config/agents.yaml'`—the class loads agent definitions from YAML.
- `tasks_config = 'config/tasks.yaml'`—The class loads task definitions from YAML.
- `self.search_tool = SerperDevTool()`—Defines a web search tool for relevant information

One good thing here is that we just need to specify the path for the YAML file. CrewAI will automatically load it.

Moving on, we define our Agents in the class:



```
@agent
def research_agent(self) → Agent:
    return Agent(
        config=self.agents_config['research_agent'],
        tools=[self.search_tool],
    )

@agent
def summarization_agent(self) → Agent:
    return Agent(
        config=self.agents_config['summarization_agent'],
    )

@agent
def fact_checker_agent(self) → Agent:
    return Agent(
        config=self.agents_config['fact_checker_agent'],
        tools=[self.search_tool],
    )
```

In the above code:

- Instead of defining mapping each parameter of the Agent from the YAML, we just specify the `config` parameter. CrewAI will automatically map YAML entries to the Agent.
- Also, each agent is dynamically created using `@agent`.

Similar to agents, we define tasks using the `@task` decorator:



```
@task
def research_task(self) → Task:
    return Task(
        config=self.tasks_config['research_task'],
        tools=[self.search_tool],
    )

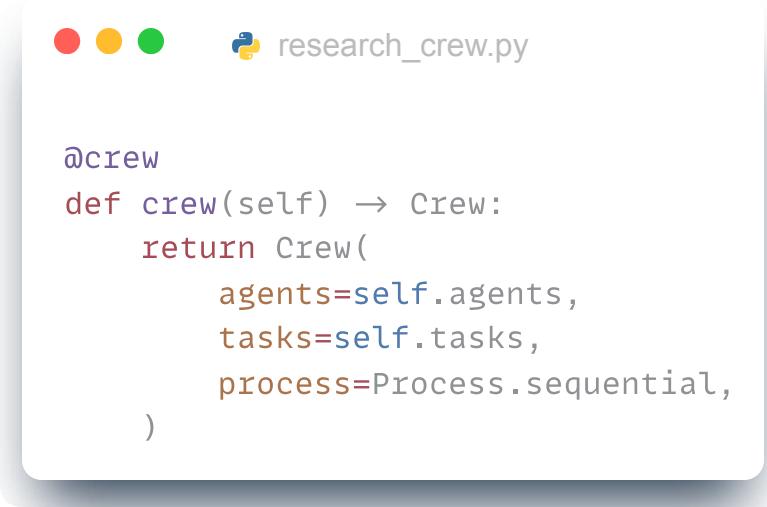
@task
def summarization_task(self) → Task:
    return Task(
        config=self.tasks_config['summarization_task']
)

@task
def fact_checking_task(self) → Task:
    return Task(
        config=self.tasks_config['fact_checking_task'],
        tools=[self.search_tool]
)
```

Yet again:

- Instead of defining mapping each parameter of the Agent from the YAML, we just specify the `config` parameter. CrewAI will automatically map YAML entries to the Task.

Finally, we define the Crew, bringing everything together.



```
research_crew.py
```

```
@crew
def crew(self) → Crew:
    return Crew(
        agents=self.agents,
        tasks=self.tasks,
        process=Process.sequential,
    )
```

- `@crew` decorator : Defines the AI workflow using CrewAI.
- `agents=self.agents` : Assigns all dynamically created agents.
- `tasks=self.tasks` : Assigns all dynamically created tasks.
- `Process.sequential` : Runs tasks one after the other (can be changed to hierarchical if needed, and we'll discuss that later in the article).

To recap, this is our final `research_crew.py` file:



research_crew.py

```
from crewai import Agent, Crew, Task, Process
from crewai.project import CrewBase, agent, task, crew
from crewai_tools import SerperDevTool

@CrewBase
class ResearchCrew:
    """A crew for conducting research, summarizing findings, and fact-checking"""

    agents_config = 'config/agents.yaml'
    tasks_config = 'config/tasks.yaml'

    def __init__(self):
        self.search_tool = SerperDevTool()

    @agent
    def research_agent(self) → Agent:
        return Agent(config=self.agents_config['research_agent'],
                     tools=[self.search_tool])

    @agent
    def summarization_agent(self) → Agent:
        return Agent(config=self.agents_config['summarization_agent'])

    @agent
    def fact_checker_agent(self) → Agent:
        return Agent(config=self.agents_config['fact_checker_agent'],
                     tools=[self.search_tool])

    @task
    def research_task(self) → Task:
        return Task(config=self.tasks_config['research_task'],
                     tools=[self.search_tool])

    @task
    def summarization_task(self) → Task:
        return Task(config=self.tasks_config['summarization_task'])

    @task
    def fact_checking_task(self) → Task:
        return Task(config=self.tasks_config['fact_checking_task'],
                     tools=[self.search_tool])

    @crew
    def crew(self) → Crew:
        return Crew(agents=self.agents,
                     tasks=self.tasks,
                     process=Process.sequential)
```

-  While we have not specified an LLM, you can add it to the `__init__` method of the class just like we have done with the serper dev tool and pass it to the Agents as the `llm` parameter. If you face issues, let us know.

With this setup, running our multi-agent system is extremely simple.

We import the `research_crew` module, instantiate the class, and kick off the crew as follows:

```
notebook.ipynb

from research_crew import ResearchCrew

research_crew = ResearchCrew()

result = research_crew.crew().kickoff(inputs={"topic": "The impact of AI on job markets"})
```

Let's look at the verbose output below:

The first part of the output is from the Internet Researcher Agent, which executes the assigned task. The agent has utilized the SerperDevTool, to search for real-time Google searches to fetch relevant information:

```

2025-02-11 23:53:00,023 - 8225323712 - __init__.py-__init__:537 - WARNING: Overriding of current TracerProvider is not allowed
# Agent: Internet Researcher
## Task: Use the SerperDevTool to search for the most relevant and recent data about The impact of AI on job markets. Extract the key insights.

# Agent: Internet Researcher
## Thought: I need to gather recent and relevant information about the impact of AI on job markets. This requires a comprehensive search.
## Using tool: Search the internet
## Tool Input:
"{"search_query": "impact of AI on job markets 2023"}"
## Tool Output:

Search results: Title: Research: How Gen AI Is Already Impacting the Labor Market
Link: https://hbr.org/2024/11/research-how-gen-ai-is-already-impacting-the-labor-market
Snippet: Despite concerns about job losses, AI also offers opportunities for job augmentation and productivity gains. Our findings show
---
Title: AI will have a major impact on labor markets. Here's how the US can ...
Link: https://fedscoop.com/ai-will-have-a-major-impact-on-labor-markets-heres-how-the-us-can-prepare/
Snippet: AI may have even larger effects, creating new types of work and making obsolete some of the job categories we use to understand
---
Title: The Impact of AI on the Labour Market – Tony Blair Institute
Link: https://institute.global/insights/economic-prosperity/the-impact-of-ai-on-the-labour-market
Snippet: Based on historic rates of labour shedding, we estimate 1 to 3 million jobs could ultimately be displaced by AI. Crucially how
---
Title: The Impact of AI on The Job Market: Key Insights – Upwork
Link: https://www.upwork.com/resources/ai-job-market-impact
Snippet: AI is reshaping the job market by creating new roles and enabling workers in existing roles to work more strategically.
---
Title: AI Will Transform the Global Economy. Let's Make Sure It Benefits ...
Link: https://www.imf.org/en/Blogs/Articles/2024/01/14/ai-will-transform-the-global-economy-lets-make-sure-it-benefits-humanity
Snippet: AI will affect almost 40 percent of jobs around the world, replacing some and complementing others. We need a careful balance
---
Title: How Will Artificial Intelligence Affect Jobs 2024–2030
Link: https://www.nexford.edu/insights/how-will-ai-affect-jobs
Snippet: Artificial intelligence (AI) could replace the equivalent of 300 million full-time jobs, a report by investment bank Goldman S
---
Title: Generative AI and the future of work in America | McKinsey
Link: https://www.mckinsey.com/mgi/our-research/generative-ai-and-the-future-of-work-in-america
Snippet: Generative AI has the potential to increase US labor productivity by 0.5 to 0.9 percentage points annually through 2030 in a range
---
Title: Insights on Generative AI and the Future of Work | NC Commerce
Link: https://www.commerce.nc.gov/news/the-lead-feed/generative-ai-and-future-work
Snippet: AI has the potential to influence jobs traditionally considered immune to automation, including roles requiring creativity or
---
Title: AI's Impact on Jobs: The Future of Work and Unemployment
Link: https://www.forwardfuture.ai/p/ai-and-the-job-market
Snippet: Artificial intelligence (AI) is rapidly reshaping the job market with increasing evidence that widespread job displacement is
---

```

In our case, we also asked the internet researcher to extract the key insights from multiple sources, and it did so as depicted below:

```
# Agent: Internet Researcher
## Final Answer:
**The Impact of AI on Job Markets: Key Insights and Sources**

1. **Job Displacement and Transformation**:
   AI technologies are expected to displace millions of jobs while simultaneously creating new roles. According to the Tony Blair Insti

2. **Productivity and Economic Growth**:
   McKinsey's analysis highlights that generative AI could boost US labor productivity significantly, leading to an estimated increase

3. **Job Creation in Emerging Industries**:
   A report by Upwork indicates that rather than purely replacing jobs, AI is reshaping existing job markets by creating new roles that

4. **Sector-Specific Impact**:
   Certain sectors, such as technology, healthcare, and hospitality, are witnessing substantial transformation due to AI integration. F

5. **Global Job Impact**:
   The International Monetary Fund (IMF) predicts that nearly 40% of jobs worldwide are likely to be affected by AI, indicating that wr

6. **Future Workforce Dynamics**:
   According to a report by Goldman Sachs, AI advancements could replace the equivalent of 300 million full-time jobs globally, pushing

7. **Policy Recommendations**:
   To successfully navigate the changes brought about by AI, it's essential for policymakers to implement measures that facilitate work

In conclusion, the impact of AI on job markets is multifaceted, involving both job displacement and creation across various sectors. Pc
```

```
# Agent: Content Summarizer
## Task: Summarize the research report into a concise and informative paragraph. Ensure clarity, coherence, and completeness.
```

```
# Agent: Content Summarizer
## Final Answer:
The research report on the impact of AI on job markets highlights significant transformations that are expected to result from AI advar
```

Once we had the insights, the content summarizer agent took over to summarize the insights.

Finally, we have the fact-checking specialist, which uses the Serper dev tool again to verify the accuracy of the summarized information.

```
# Agent: Fact-Checking Specialist
## Thought: I need to verify the accuracy of the summarized information regarding the impact of AI on job markets, checking statistics,
## Using tool: Search the internet
## Tool Input:
 "{\"search_query\": \"impact of AI on job markets 2023 report job displacement statistics productivity increase Goldman Sachs\"}"
## Tool Output:

Search results: Title: Generative AI could raise global GDP by 7% - Goldman Sachs
Link: https://www.goldmansachs.com/insights/articles/generative-ai-could-raise-global-gdp-by-7-percent
Snippet: They could drive a 7% (or almost $7 trillion) increase in global GDP and lift productivity growth by 1.5 percentage points over
---
Title: The Potentially Large Effects of Artificial Intelligence on Economic ...
Link: https://www.gspublishing.com/content/research/en/reports/2023/03/27/d64e052b-0f6e-45d7-967b-d7be35fabd16.html
Snippet: These results suggest that the direct effects of generative AI on labor demand could be negative in the near-term if AI affect
---
Title: Goldman Sachs Predicts 300 Million Jobs Will Be Lost Or Degraded ...
Link: https://www.forbes.com/sites/jackkelly/2023/03/31/goldman-sachs-predicts-300-million-jobs-will-be-lost-or-degraded-by-artificial-
Snippet: Goldman Sachs maintains that if generative AI lives up to its hype, the workforce in the United States and Europe will be uper
---
Title: AI is showing "very positive" signs of eventually boosting GDP and ...
Link: https://www.goldmansachs.com/insights/articles/AI-is-showing-very-positive-signs-of-boosting-gdp
Snippet: Goldman Sachs Research predicted last year that generative AI could boost GDP and raise labor productivity growth over the com
---
Title: Artificial Intelligence and the Labor Market - Sciences Po
Link: https://www.sciencespo.fr/women-in-business/en/news/article-artificial-intelligence-and-the-labor-market
Snippet: Earlier in 2023 for example, a Goldman Sachs report stated that while generative AI could create new jobs and boost global pro
---
Title: A.I. automation could impact 300 million jobs – here's which ones
Link: https://www.cnbc.com/2023/03/28/ai-automation-could-impact-300-million-jobs-heres-which-ones.html
Snippet: The use of AI technology could also boost labor productivity growth and boost global GDP by as much as 7% over time, Goldman S
---
Title: How Will Artificial Intelligence Affect Jobs 2024–2030
Link: https://www.nexford.edu/insights/how-will-ai-affect-jobs
Snippet: Artificial intelligence (AI) could replace the equivalent of 300 million full-time jobs, a report by investment bank Goldman S
---
Title: AI Replacing Jobs Statistics By AI's Impact on Job, AI Skills and Facts
Link: https://electroiq.com/stats/ai-replacing-jobs-statistics/
Snippet: Goldman Sachs via BBC predicts that AI may replace 300 million jobs worldwide, representing 9.1% of all global employment.
---
Title: Is Society Ready for the Impact of AI on the Workforce?
Link: https://www.cigionline.org/articles/is-society-ready-for-the-impact-of-ai-on-the-workforce/
Snippet: Goldman Sachs, a US investment bank, published a 2023 report estimating that AI could impact 300 million jobs globally due to
---
```

Once that's been verified, it produces the final output

```
# Agent: Fact-Checking Specialist
## Final Answer:
### Research Report Summary: The Impact of AI on Job Markets

#### 1. **Introduction**
The integration of Artificial Intelligence (AI) into various sectors is reshaping job markets globally. The Future of Work Report by LinkedIn emphasize

#### 2. **Key Insights from Recent Reports**

- **Job Creation vs. Job Displacement**
  AI and automation are expected to displace specific roles; however, the World Economic Forum's Future of Jobs Report 2023 indicates that more jobs wi

- **Increased Demand for AI Skills**
  A significant trend is the rise in demand for AI-related skills in job postings, with LinkedIn reporting a 2.4-fold increase from 2021 to 2023. Addit

- **Sectoral Impacts**
  Sectors such as technology, finance, healthcare, and manufacturing are adapting rapidly to AI implementation, leading to enhanced productivity and pr

- **Workforce Displacement of Specific Jobs**
  Pew Research Center's findings highlight that 20% of U.S. jobs are highly exposed to AI, particularly those with routine tasks. Women and college-edu

- **AI's Influence on Job Satisfaction and Wages**
  The OECD Employment Outlook 2023 discusses the dual impact of AI on job satisfaction and wages. While AI can enhance workers' job satisfaction and wa

#### 3. **Conclusion**
AI has the potential to significantly transform the labor market by fostering innovation and job creation while altering skill requirements and workfor

#### Sources:
- [Future of Work Report: AI at Work – LinkedIn's Economic Graph] (https://economicgraph.linkedin.com/research/future-of-work-report-ai)
- [The Impact of AI on the Labour Market – Tony Blair Institute] (https://institute.global/insights/economic-prosperity/the-impact-of-ai-on-the-labour-m)
- [OECD Employment Outlook 2023] (https://www.oecd.org/en/publications/oecd-employment-outlook-2023\_08785bba-en.html)
- [Which U.S. Workers Are More Exposed to AI on Their Jobs? – Pew Research] (https://www.pewresearch.org/social-trends/2023/07/26/which-u-s-workers-are-more-exposed-to-ai-on-their-jobs/)
- [AI Jobs Barometer – PwC] (https://www.pwc.com/gx/en/issues/artificial-intelligence/ai-jobs-barometer.html)
```

This compilation of insights highlights the diverse implications of AI on job markets, emphasizing the need for continuous learning and adaptation with

We can also print the final output as follows:

```
from IPython.display import Markdown  
Markdown(response.raw)
```

✓ 0.0s

Python

Research Report Summary: The Impact of AI on Job Markets

1. Introduction

The integration of Artificial Intelligence (AI) into various sectors is reshaping job markets globally. The Future of Work Report by LinkedIn emphasizes the transformative nature of AI, which not only alters existing job roles but also creates new job categories and changes skill requirements. This report evaluates the critical impacts, findings, and trends related to AI and employment.

2. Key Insights from Recent Reports

- **Job Creation vs. Job Displacement**

AI and automation are expected to displace specific roles; however, the World Economic Forum's Future of Jobs Report 2023 indicates that more jobs will likely be created than eliminated, especially in emerging tech sectors like AI development and machine learning.

- **Increased Demand for AI Skills**

A significant trend is the rise in demand for AI-related skills in job postings, with LinkedIn reporting a 2.4-fold increase from 2021 to 2023. Additionally, the average number of AI skills per job applicant has surged by 177%.

- **Sectoral Impacts**

Sectors such as technology, finance, healthcare, and manufacturing are adapting rapidly to AI implementation, leading to enhanced productivity and positioning companies for long-term growth.

- **Workforce Displacement of Specific Jobs**

Pew Research Center's findings highlight that 20% of U.S. jobs are highly exposed to AI, particularly those with routine tasks. Women and college-educated workers are disproportionately affected, indicating a gender and educational disparity.

- **AI's Influence on Job Satisfaction and Wages**

The OECD Employment Outlook 2023 discusses the dual impact of AI on job satisfaction and wages. While AI can enhance workers' job satisfaction and wage conditions, it also raises concerns regarding job security and economic inequality.

3. Conclusion

AI has the potential to significantly transform the labor market by fostering innovation and job creation while altering skill requirements and workforce composition. It is crucial for workers to adapt and acquire new skills relevant to AI technologies to maintain competitiveness. Stakeholders must closely monitor these trends to navigate challenges and capitalize on the opportunities presented by AI in future job markets.

Sources:

- [Future of Work Report: AI at Work - LinkedIn's Economic Graph](#)
- [The Impact of AI on the Labour Market - Tony Blair Institute](#)
- [OECD Employment Outlook 2023](#)
- [Which U.S. Workers Are More Exposed to AI on Their Jobs? - Pew Research](#)
- [AI Jobs Barometer - PwC](#)

This compilation of insights highlights the diverse implications of AI on job markets, emphasizing the need for continuous learning and adaptation within the workforce.

Done!

Overall, this makes your code much more clean and maintainable.

Also, since it's a class, you can now instantiate it multiple times with different config files, different LLMs, and more.

Structured output

By default, LLMs generate raw text responses which is fine for conversational AI but not ideal for structured applications.

In most real-world AI workflows, we need predictable, structured outputs that can be:

- Easily parsed by downstream applications.

- Formatted consistently across multiple runs.
- Used for automation without requiring manual cleanup.

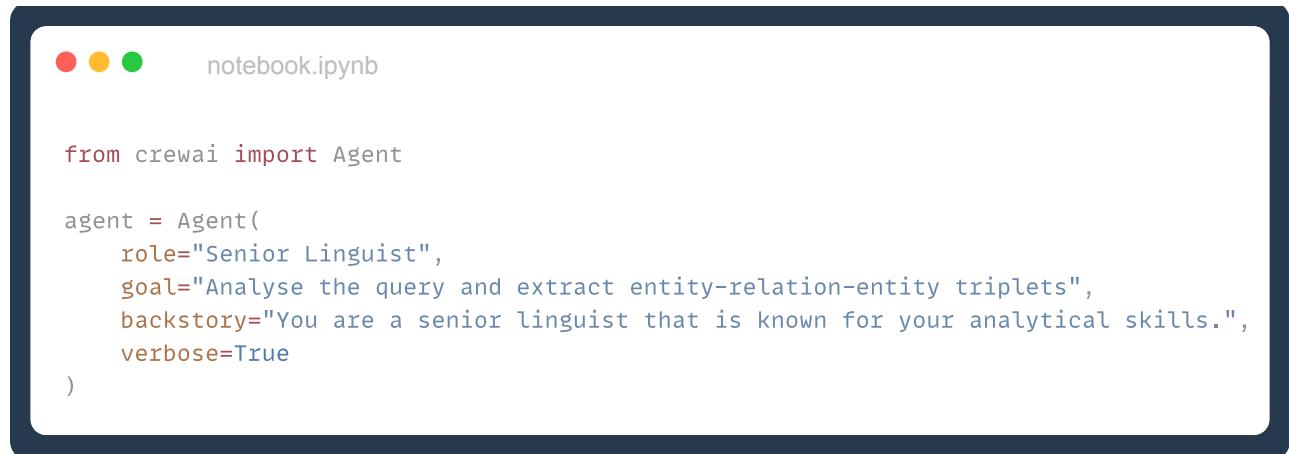
Structured outputs are especially valuable when:

- you have integrated APIs or databases that expect data in structured formats like JSON.
- you want to ensure a fixed structure for extracting key insights.
- you want to produce standardized responses and avoid inconsistencies in model-generated content.

Let's take an example to understand better.

Imagine an AI agent to analyze a natural language query and return entity-relation-entity triplets in a structured JSON format.

First, we create an Agent responsible for extracting structured triplets from text.



```
● ● ● notebook.ipynb

from crewai import Agent

agent = Agent(
    role="Senior Linguist",
    goal="Analyse the query and extract entity-relation-entity triplets",
    backstory="You are a senior linguist that is known for your analytical skills.",
    verbose=True
)
```

Next, we define a Task that instructs the agent to extract entity-relation-entity triplets from the given text.



notebook.ipynb

```
from crewai import Task

task = Task(
    description="""Analyse the query and return structured JSON
                  output in the form of
                  - entity
                  - relation
                  - entity

                  The query is: {query}
                  """,
    expected_output="A structured JSON object with the entity-relation-entity triplets",
    verbose=True,
    agent=agent
)
```

Now, we define a Crew that will manage the execution of our agent and task.

```
from crewai import Crew, Process

crew = Crew(
    agents=[agent],
    tasks=[task],
    process=Process.sequential,
    verbose=True
)
```

Finally, we execute the Crew with a sample query containing multiple entity relationships.



```
response = crew.kickoff(inputs={
    "query": """Paris is the capital of France.
    The Eiffel Tower is in Paris."""
}
)
```

The verbose output shows this:

```
# Agent: Senior Linguist
## Task: Analyse the query and return structured JSON
        output in the form of
        - entity
        - relation
        - entity

The query is: Paris is the capital of France. The Eiffel Tower is in Paris.
```

```
# Agent: Senior Linguist
## Final Answer:
{
  "triplets": [
    {
      "entity1": "Paris",
      "relation": "is the capital of",
      "entity2": "France"
    },
    {
      "entity1": "The Eiffel Tower",
      "relation": "is in",
      "entity2": "Paris"
    }
  ]
}
```

And if we print the response object, we get:

```
print(response.raw)
✓ 0.0s

{
  "triplets": [
    {
      "entity1": "Paris",
      "relation": "is the capital of",
      "entity2": "France"
    },
    {
      "entity1": "The Eiffel Tower",
      "relation": "is in",
      "entity2": "Paris"
    }
  ]
}
```

Now, of course, this works but at times, this approach can produce some issues/.

Consider this example:

```
notebook.ipynb

query = "Paris is the capital of France."
response = crew.kickoff(inputs={"query":query})
```

The verbose output shows this, which appears good in the first look:

```
# Agent: Senior Linguist
## Task: Analyse the query and return structured JSON
          output in the form of
          - entity
          - relation
          - entity
```

The query is: Paris is the capital of France.

```
# Agent: Senior Linguist
## Final Answer:
```json
{
 "triplets": [
 {
 "entity1": "Paris",
 "relation": "is the capital of",
 "entity2": "France"
 }
]
}
```
...
```

However, if we print the response, notice that it has the "json" keyword after the backticks, indicating that it's not exactly the exact format we would want our output in.

```
    print(response.raw)
✓ 0.0s

```json
{
 "triplets": [
 {
 "entity1": "Paris",
 "relation": "is the capital of",
 "entity2": "France"
 }
]
}
```

```

This would either require one more step of parsing (which is not desired since the dedicated agent should do it) or a much more strict instructions in the prompt—which means more token usage.

Thankfully, we can enforce structured outputs easily in LLMs using Pydantic.

Pydantic is a data validation library in Python that lets us define precise output schemas.

Here's how it helps in this case.

First, we define a structured schema for entity-relation-entity triplets.

```
● ● ● notebook.ipynb

from pydantic import BaseModel, Field

class EntityRelationEntity(BaseModel):
    entity: str = Field(description="The first entity in the triplet")
    relation: str = Field(description="The relation between the first and second entity")
    entity: str = Field(description="The second entity in the triplet")
```

- 💡 Whenever you define structured outputs, ALWAYS add descriptions to each field for better model guidance.

Now, we define the AI Agent responsible for extracting structured triplets from text.

```
● ● ● notebook.ipynb

from crewai import Agent

agent = Agent(
    role="Senior Linguist",
    goal="Analyse the query and extract entity-relation-entity triplets",
    backstory="You are a senior linguist that is known for your analytical skills.",
    verbose=True
)
```

Now, we modify the task definition to force the model to return structured JSON using Pydantic.



notebook.ipynb

```
from crewai import Task

task = Task(
    description="""Analyse the query and return structured JSON
                  output in the form of
                  - entity
                  - relation
                  - entity

                  The query is: {query}
                  """,
    expected_output="""A structured JSON object with the
                      entity-relation-entity triplets""",
    output_pydantic=EntityRelationEntity,
    verbose=True,
    agent=agent
)
```

In the above code, we enforce JSON output by specifying the `output_pydantic` parameter. This prevents freeform text responses by making sure the model strictly adheres to the schema.

Now, we define the Crew and execute it:



notebook.ipynb

```
from crewai import Crew, Process

crew = Crew(
    agents=[agent],
    tasks=[task],
    process=ProcessSEQUENTIAL,
    verbose=True
)

response = crew.kickoff(inputs={"query": "Paris is the capital of France."})
```

Now, instead of getting raw text that requires further processing, the AI directly returns a structured JSON response like this:

```
# Agent: Senior Linguist
## Task: Analyse the query and return structured JSON
          output in the form of
          - entity
          - relation
          - entity
```

The query is: Paris is the capital of France.

```
# Agent: Senior Linguist
## Final Answer:
{
  "entity": "Paris",
  "relation": "is the capital of",
  "entity": "France"
}
```

Let's also print the `response` object to validate this:

```
print(response.raw)
✓ 0.0s
{
  "entity": "Paris",
  "relation": "is the capital of",
  "entity": "France"
}
```

Perfect!!

This output is perfectly structured and requires no additional parsing, guaranteeing consistency across outputs.

The same idea can be extended to multi-agent workflows as well. All you need to do is define the structured output schema, give descriptions for each field, and pass it in the task definition.

Custom tools

As we saw earlier in Part 1, while LLM-powered agents are great at reasoning and generating responses, they lack direct access to real-time information, external systems, and specialized computations.

Tools allow the Agent to:

- Search the web for real-time data.
- Retrieve structured information from APIs and databases.
- Execute code to perform calculations or data transformations.
- Analyze images, PDFs, and documents beyond just text inputs.

CrewAI supports several tools that you can integrate with Agents, as depicted below:

12 Powerful *crewai* Tools to Build AI Agents

 join.DailyDoseofDS.com

1) File Read tool

```
from crewai_tools  
import FileReadTool
```

Read and extract
data from files

2) File Writer tool

```
from crewai_tools  
import FileWriterTool
```

Write contents
to files

3) Code Interpreter tool

```
from crewai_tools  
import CodeInterpreterTool
```

Execute Python code
generated by Agent

4) Scrape Website tool

```
from crewai_tools  
import ScrapeWebsiteTool
```

Extract and read
content of website

5) Serper Dev tool

```
from crewai_tools  
import SerperDevTool
```

Retrieve search results
from the internet

6) Directory Read Tool

```
from crewai_tools  
import DirectoryReadTool
```

List the contents of
a directory

7) Firecrawl Search tool

```
from crewai_tools  
import FirecrawlSearchTool
```

Search and convert
websites to markdown

8) Browserbase Load tool

```
from crewai_tools  
import BrowserbaseLoadTool
```

Run, manage, and monitor
headless browsers

9) PDF Search tool

```
from crewai_tools  
import PDFSearchTool
```

RAG tool for
searching in PDFs

10) Github Search tool

```
from crewai_tools  
import GithubSearchTool
```

RAG tool for searching
in GitHub repos

11) TXT Search tool

```
from crewai_tools  
import TXTSearchTool
```

RAG tool for searching
in text files

12) NL2SQL tool

```
from crewai_tools  
import NL2SQLTool
```

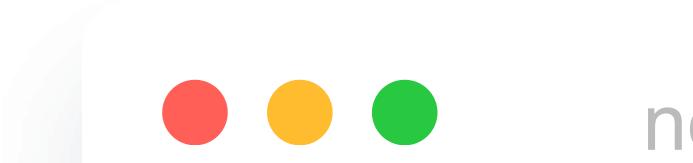
Convert natural language
to SQL Queries

However, you may need to build custom tools at times.

In this example, we're building a real-time currency conversion tool inside CrewAI. Instead of making an LLM guess exchange rates, we integrate a custom tool that fetches live exchange rates from an external API and provides some insights.

Below, let's look at how you can build one for your custom needs in the CrewAI framework.

Firstly, make sure the tools package is installed:



You would also need an API key from here: <https://www.exchangerate-api.com/> (it's free). Specify it in the `.env` file as shown below:

A screenshot of a terminal window showing the contents of a `.env` file. The file contains three environment variables: `OPENAI_API_KEY="sk-4...`, `SERPER_API_KEY="42131..."`, and `EXCHANGE_RATE_API_KEY="753..."`. The terminal interface includes a light gray header bar with three colored dots (red, yellow, green) and a dark blue footer bar.

Once that's done, we start with some standard import statements:

A screenshot of a terminal window showing the contents of a `notebook.ipynb` file. The code block imports several modules: `dotenv` for loading environment variables, `os` for operating system functions, `requests` for making HTTP requests, `typing` for type annotations, `BaseTool` from `crewai.tools`, and `BaseModel` and `Field` from `pydantic`. The terminal interface includes a light gray header bar with three colored dots (red, yellow, green) and a dark blue footer bar.

Next, we define the input fields the tool expects using Pydantic.

```
● ● ● notebook.ipynb

class CurrencyConverterInput(BaseModel):
    """Input schema for CurrencyConverterTool."""
    amount: float = Field(..., description="The amount to convert.")
    from_currency: str = Field(..., description="The source currency code (e.g., 'USD').")
    to_currency: str = Field(..., description="The target currency code (e.g., 'EUR').")
```

Now, we define the CurrencyConverterTool by inheriting from `BaseTool`:

```
● ● ● notebook.ipynb

class CurrencyConverterTool(BaseTool):
    name: str = "Currency Converter Tool"
    description: str = "Converts an amount from one currency to another."
    args_schema: Type[BaseModel] = CurrencyConverterInput
    api_key: str = os.getenv("EXCHANGE_RATE_API_KEY")
```

Every tool class should have the `_run` method which we will executed whenever the Agents wants to make use of it.

For our use case, we implement it as follows:



notebook.ipynb

```
def _run(self, amount: float, from_currency: str, to_currency: str) -> str:
    url = f"https://v6.exchangerate-api.com/v6/{self.api_key}/latest/{from_currency}"
    response = requests.get(url)

    if response.status_code != 200:
        return "Failed to fetch exchange rates."

    data = response.json()
    if "conversion_rates" not in data or to_currency not in data["conversion_rates"]:
        return f"Invalid currency code: {to_currency}"

    rate = data["conversion_rates"][to_currency]
    converted_amount = amount * rate
    return f"{amount} {from_currency} is equivalent to {converted_amount:.2f} {to_currency}."
```

In the above code, we fetch live exchange rates using an API request. We also handles errors if the request fails or the currency code is invalid.

Now, we define an agent that uses the tool for real-time currency analysis and attach our `CurrencyConverterTool`, allowing the agent to call it directly if needed:



notebook.ipynb

```
from crewai import Agent

currency_analyst = Agent(
    role="Currency Analyst",
    goal="Provide real-time currency conversions and financial insights.",
    backstory=(
        "You are a finance expert with deep knowledge of global exchange rates."
        "You help users with currency conversion and financial decision-making."
    ),
    tools=[CurrencyConverterTool()], # Attach our custom tool
    verbose=True
)
```

We assign a task to the `currency_analyst` agent.



notebook.ipynb

```
from crewai import Task

currency_conversion_task = Task(
    description=(
        "Convert {amount} {from_currency} to {to_currency} "
        "using real-time exchange rates."
        "Provide the equivalent amount and "
        "explain any relevant financial context."
    ),
    expected_output=("A detailed response including the "
                    "converted amount and financial insights."),
    agent=currency_analyst
)
```

Finally, we create a Crew, assign the agent to the task, and execute it.



notebook.ipynb

```
from crewai import Crew, Process

crew = Crew(
    agents=[currency_analyst],
    tasks=[currency_conversion_task],
    process=Process.sequential
)

response = crew.kickoff(inputs={"amount": 100,
                                 "from_currency": "USD",
                                 "to_currency": "EUR"})
```

Printing the response, we get the following output:

```
from IPython.display import Markdown  
Markdown(response.raw)
```

✓ 0.0s

Python

Converting 100 USD to EUR using real-time exchange rates results in approximately 95.40 EUR.

In the financial context, it's worth noting that exchange rates can fluctuate due to various factors like economic indicators, interest rates, and geopolitical events. As of now, the conversion reflects current market conditions, which are influenced by the latest economic data releases and monetary policies in both the United States and the Eurozone. Given the recent trends, if you're planning a trip to Europe or making an investment, these rates may change, so it's beneficial to monitor them regularly.

Works as expected!

Here, note that in the above implementation, we directly passed structured input to the Crew.

While this approach works well in controlled environments, real-world user queries are often unstructured. A user might input:

How much is 100 dollars in euros today?

Instead of expecting users to manually structure their input, we can introduce a Query Parsing Agent that:

- Extracts relevant details (amount, source currency, target currency).
- Converts natural language into a structured format.
- Passes the structured query to the Currency Analyst Agent.

This will be similar to what we learned in the previous section on producing structured outputs.

Can you build this multi-agent system as an exercise? You need to implement these steps:

- First, define a Pydantic model for the structured output.

- An agent will analyze user input (e.g., "Convert 100 dollars to euros today.") and output a structured JSON object following the Pydantic model from Step 1.
- Next, our currency Analyst Agent will receive this structured input from the Query Parsing Agent. It should still fetch live exchange rates using the `CurrencyConverterTool` and return the conversion result.
- Use CrewAI's sequential process to ensure smooth execution.

Here is the solution:

notebook

 notebook.ipynb • 10 KB

Let us know if you face any issues with the implementation.

Conclusion, takeaways and next steps

With that, we come to an end of Part 2 of the Agents crash course.

In this part, we worked on structuring AI workflows by integrating custom tools and multi-agent collaboration.

Instead of relying on LLMs for everything, we offloaded well-defined tasks to dedicated tools.

We also learned how to build modular Crews for better organization and scalability. By defining agents and tasks in YAML, we separated configuration from execution, making our codebase cleaner and more maintainable.

Try implementing the exercise we mentioned in custom tools section! If you haven't already, build the multi-agent pipeline from scratch and refine it further.

Also, experiment with more tools. Extend this approach beyond currency conversion—try integrating weather APIs, stock market data, or document summarization tools.

12 Powerful *crewai* Tools to Build AI Agents

join.DailyDoseofDS.com

| | | |
|---|--|--|
| 1) File Read tool

from crewai_tools
import FileReadTool

Read and extract
data from files | 2) File Writer tool

from crewai_tools
import FileWriterTool

Write contents
to files | 3) Code Interpreter tool

from crewai_tools
import CodeInterpreterTool

Execute Python code
generated by Agent |
| 4) Scrape Website tool

from crewai_tools
import ScrapeWebsiteTool

Extract and read
content of website | 5) Serper Dev tool

from crewai_tools
import SerperDevTool

Retrieve search results
from the internet | 6) Directory Read Tool

from crewai_tools
import DirectoryReadTool

List the contents of
a directory |
| 7) Firecrawl Search tool

from crewai_tools
import FirecrawlSearchTool

Search and convert
websites to markdown | 8) Browserbase Load tool

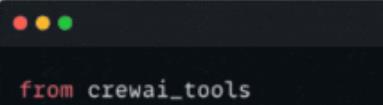
from crewai_tools
import BrowserbaseLoadTool

Run, manage, and monitor
headless browsers | 9) PDF Search tool

from crewai_tools
import PDFSearchTool

RAG tool for
searching in PDFs |
| 10) Github Search tool

from crewai_tools
import GithubSearchTool

RAG tool for searching
in GitHub repos | 11) TXT Search tool

from crewai_tools
import TXTSearchTool

RAG tool for searching
in text files | 12) NL2SQL tool

from crewai_tools
import NL2SQLTool

Convert natural language
to SQL Queries |

In the upcoming parts, we have several other advanced agentic things planned for you:

- Using Flows for better multi-agent coordination.

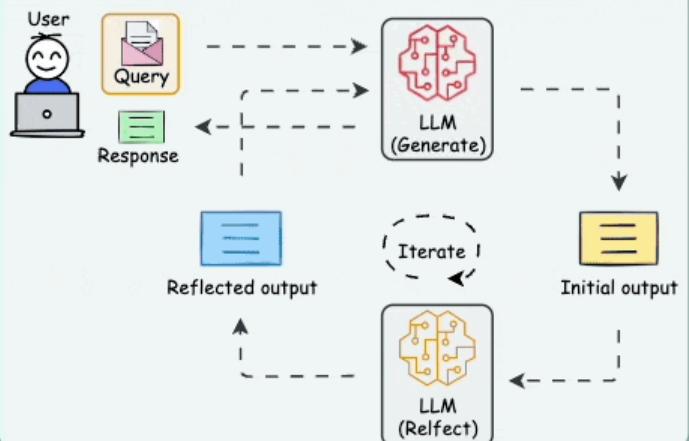
- Building production-ready agentic pipelines that scale.
- Creating and integrating custom tools for enhanced agent capabilities.
- Agentic RAG (Retrieval-Augmented Generation) – combining RAG with AI agents.
- Optimizing agents for real-world applications in business and automation.
- Building Agents around the Agentic patterns depicted below:

5 Most Popular Agentic AI Design Patterns

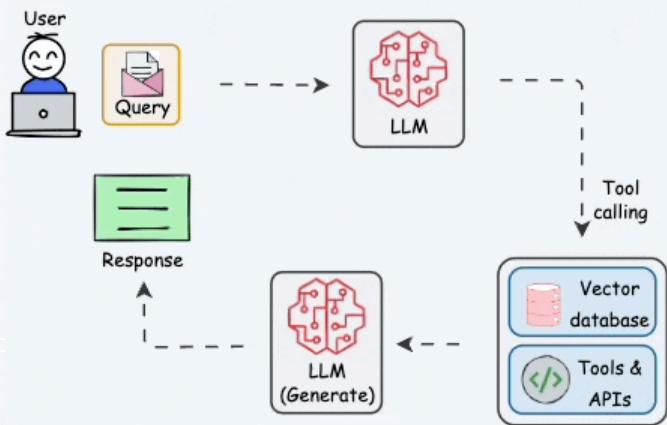


join.DailyDoseofDS.com

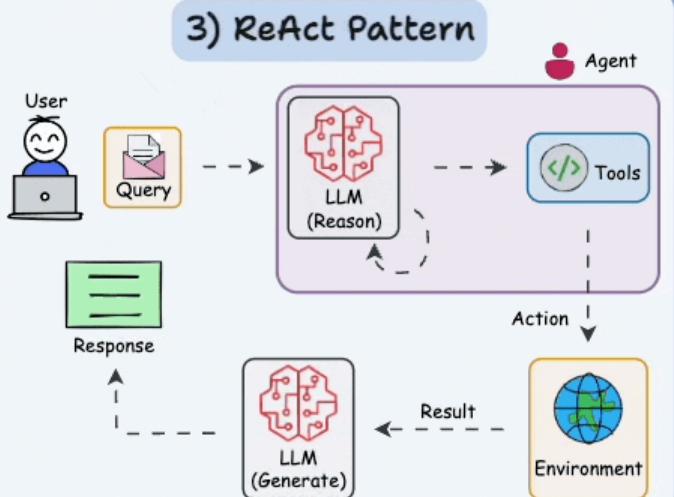
1) Reflection Pattern



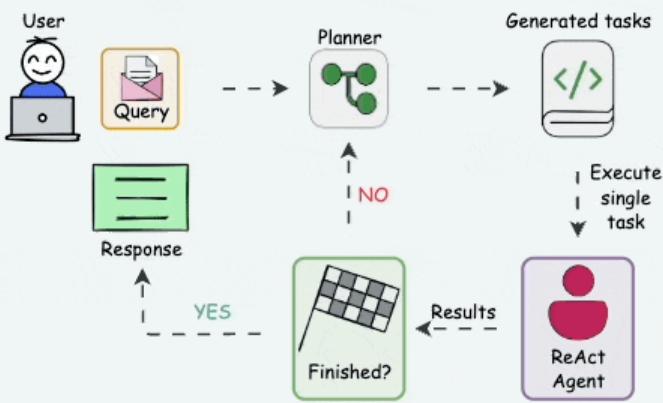
2) Tool Use Pattern



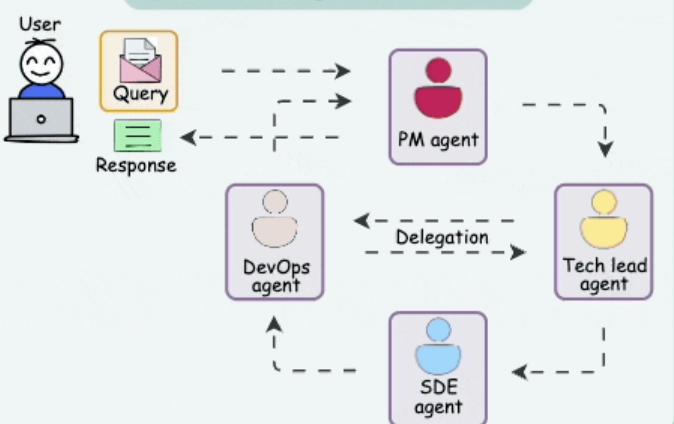
3) ReAct Pattern



4) Planning Pattern



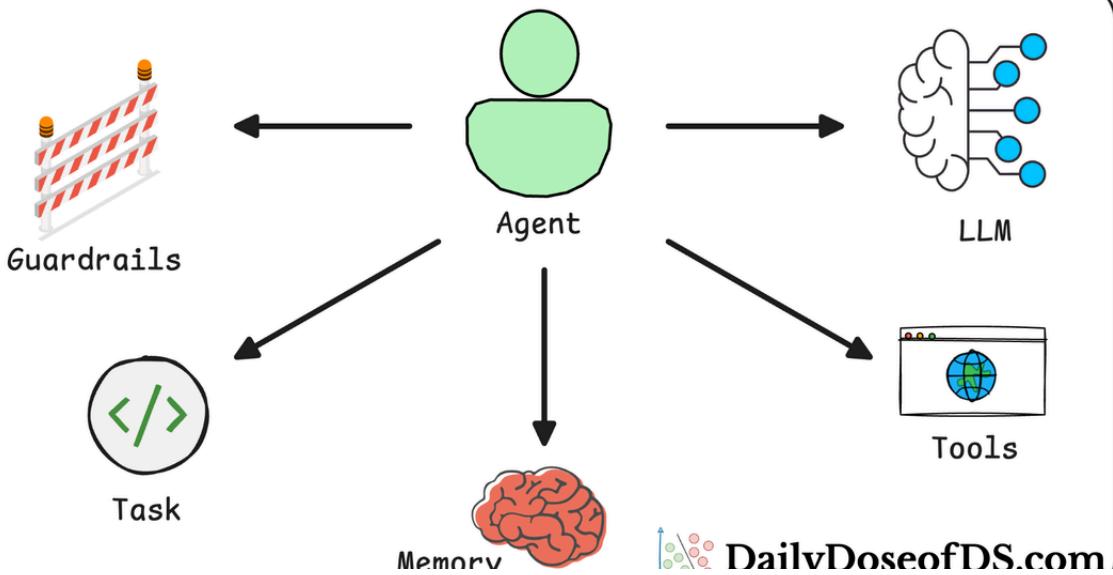
5) Multi-agent Pattern



- and many many more.

Read the next part of this crash course here:

AI Agents Crash Course



DailyDoseofDS.com

Building Flows in Agentic Systems (Part A)

AI Agents Crash Course—Part 3 (with implementation).



Daily Dose of Data Science • Avi Chawla

As always, thanks for reading!

Any questions?

Feel free to post them in the comments.

Or

If you wish to connect privately, feel free to initiate a chat here:

Chat Icon



A daily column with insights, observations, tutorials and best practices on python and data science. Read by industry professionals at big tech, startups, and engineering students, across:



Navigation

[Newsletter](#)

[Contact](#)

[Search](#)

[Blogs](#)

[FAQs](#)

[More](#)

[About](#)

Connect via chat

Agents

LLMs

AI Agent Crash Course

Share this article



Read next

MCP Jun 15, 2025 • 22 min read



The Full MCP Blueprint: Building a Full-Fledged MCP Workflow using Tools, Resources, and Prompts

Model context protocol crash course—Part 4.

Avi Chawla, Akshay Pachaar

Agents Jun 8, 2025 • 20 min read



The Full MCP Blueprint: Building a Custom MCP Client from Scratch

Model context protocol crash course—Part 3.

Avi Chawla, Akshay Pachaar

Agents

Jun 1, 2025 • 22 min read



The Full MCP Blueprint: Background, Foundations, Architecture, and Practical Usage (Part B)

Model context protocol crash course—Part 2.

Avi Chawla, Akshay Pachaar

A daily column with insights, observations, tutorials and best practices on python and data science. Read by industry professionals at big tech, startups, and engineering students, across:



Navigation

Sponsor

Newsletter

More

Contact

