Feb 9, 2025

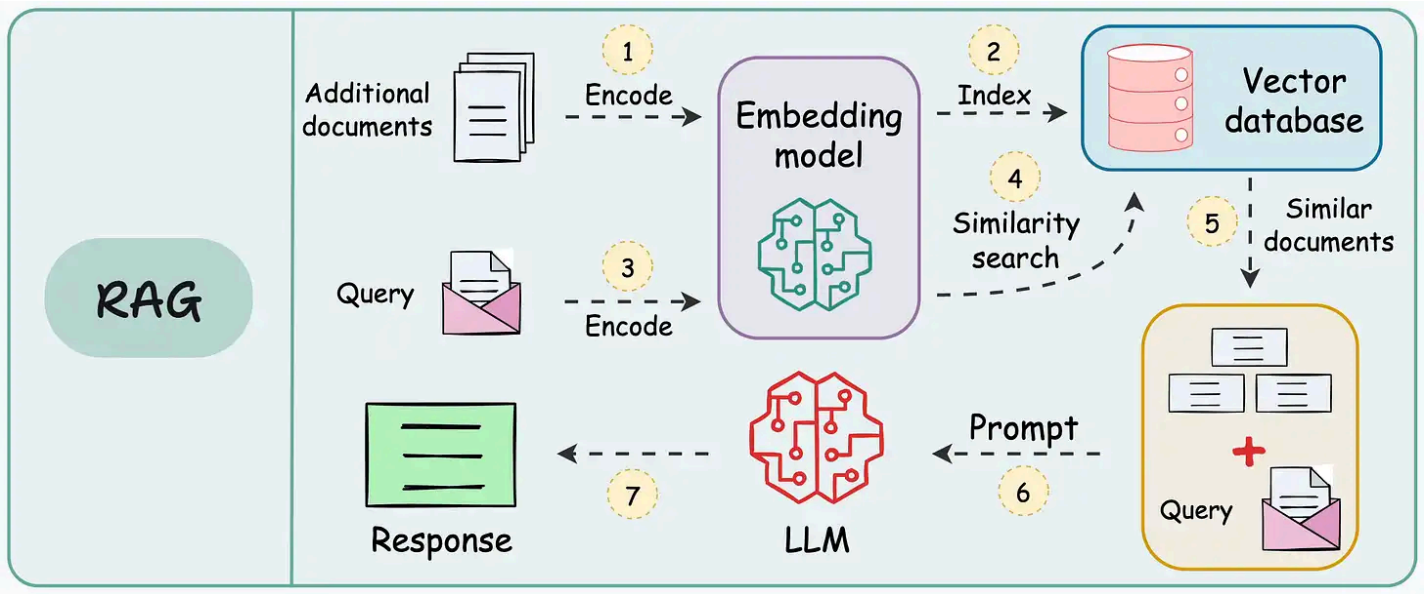
Agentic Systems 101: Fundamentals, Building Blocks, and How to Build Them (Part A)

AI Agents Crash Course—Part 1 (with implementation).

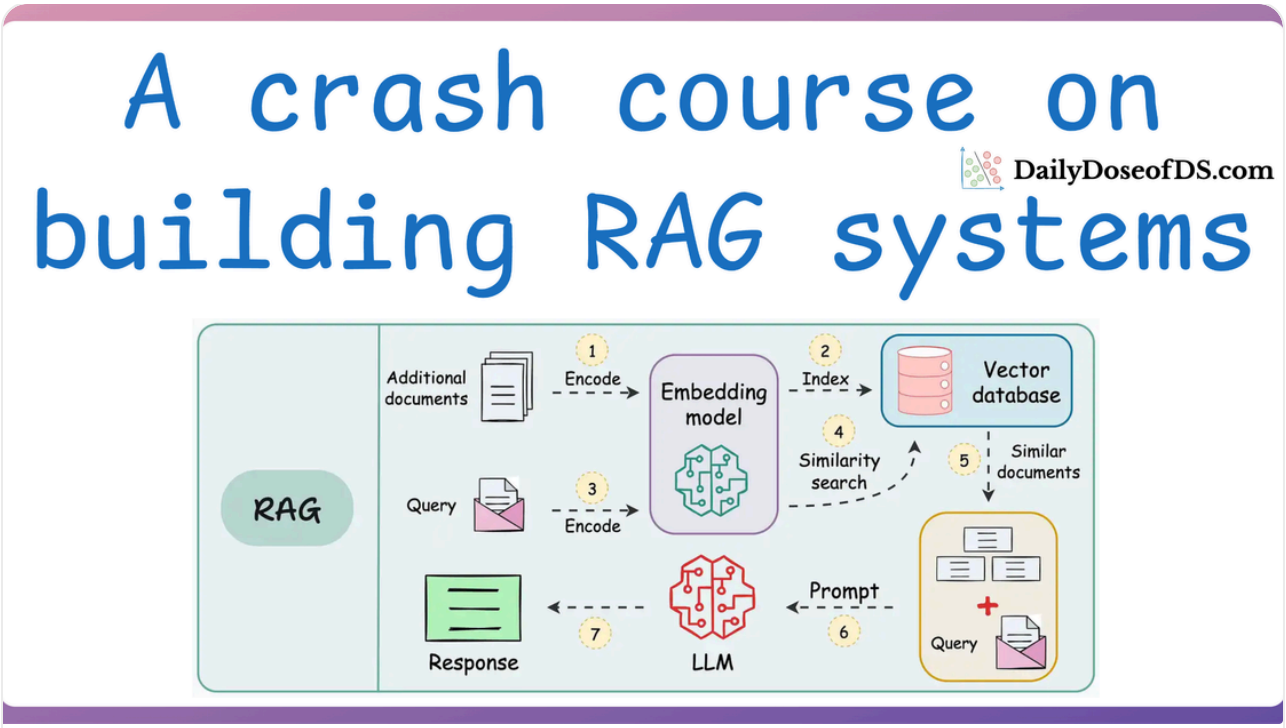
Avi Chawla, Akshay Pachaar

Introduction

Our [RAG crash course] was an incredible deep dive into practical applications, covering everything from retrieval pipelines to multimodal integration and efficient retrieval strategies.



We explored how to optimize large-scale retrieval systems, how to evaluate them, and even how to extend them beyond text into images and structured data.



RAG Crash Course - Daily Dose of Data Science

Daily Dose of Data Science • Avi Chawla

Now, it’s time to take things a step further in this new crash course on AI Agents!

If RAG systems are about retrieving and enhancing information, AI agents take it to the next level—they reason, plan, and autonomously take action based on the information they process.

Instead of simply fetching relevant content, agents can coordinate workflows, interact with external tools, and dynamically adapt to changing scenarios.

Let's dive into more details to understand the motivations for AI Agents.

Motivation

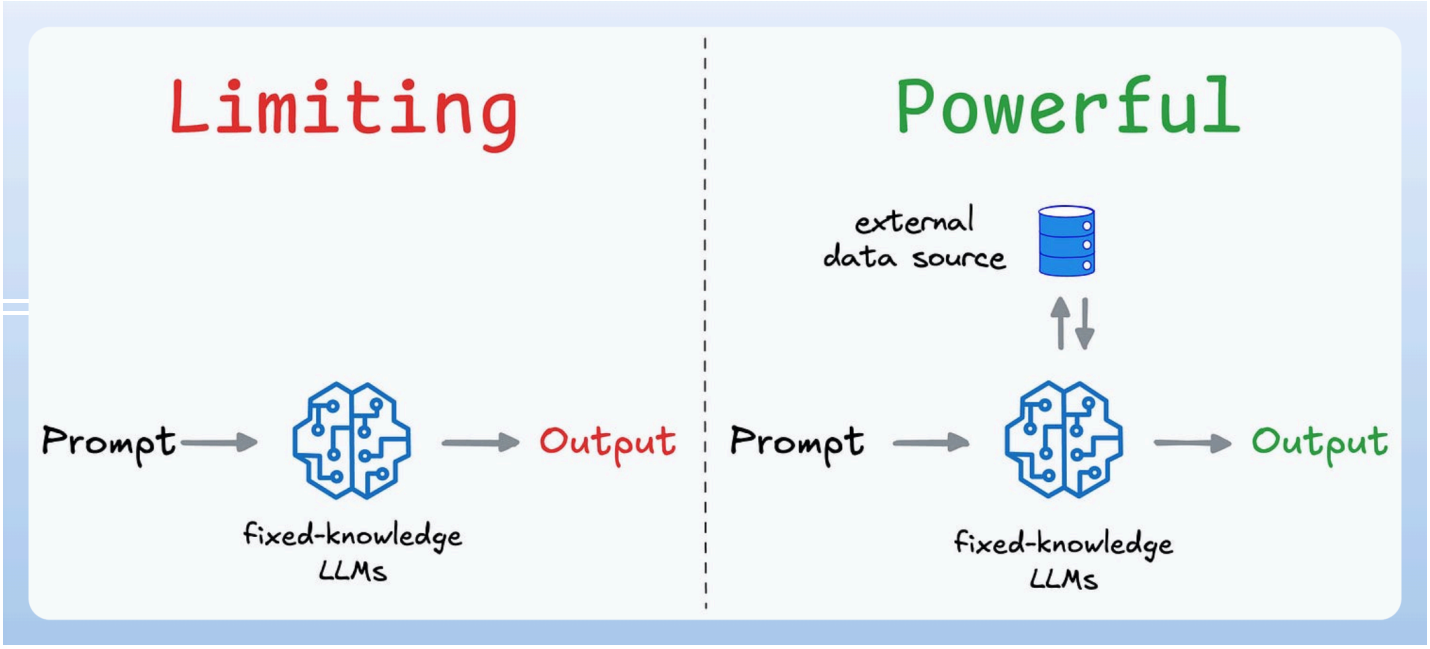
There are three perspectives to understanding the motivation behind AI Agents:

1) RAG perspective

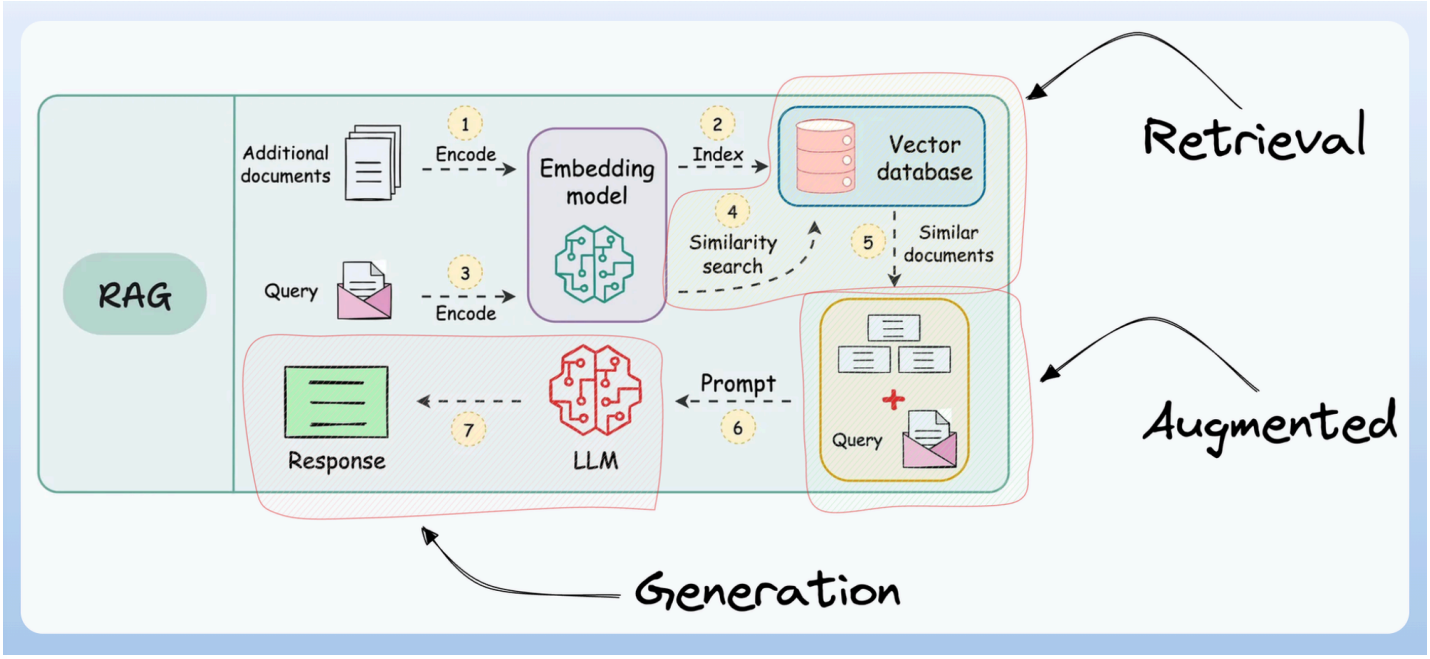
Given the scale and capabilities of modern LLMs, it feels limiting to use them as "standalone generative models" for pretty ordinary tasks like text summarization, text completion, code completion, etc.

Instead, their true potential is only realized when you build systems around these models, where they are allowed to:

- access, retrieve, and filter data from relevant sources,
- analyze and process this data to make real-time decisions and more.



RAG was a pretty successful step towards building such compound AI systems:



But since most RAG systems follow a programmatic flow (you, as a programmer, define the steps, the database to search for, the context to retrieve, etc.), it does not unlock the full autonomy one may expect these compound AI systems to possess.

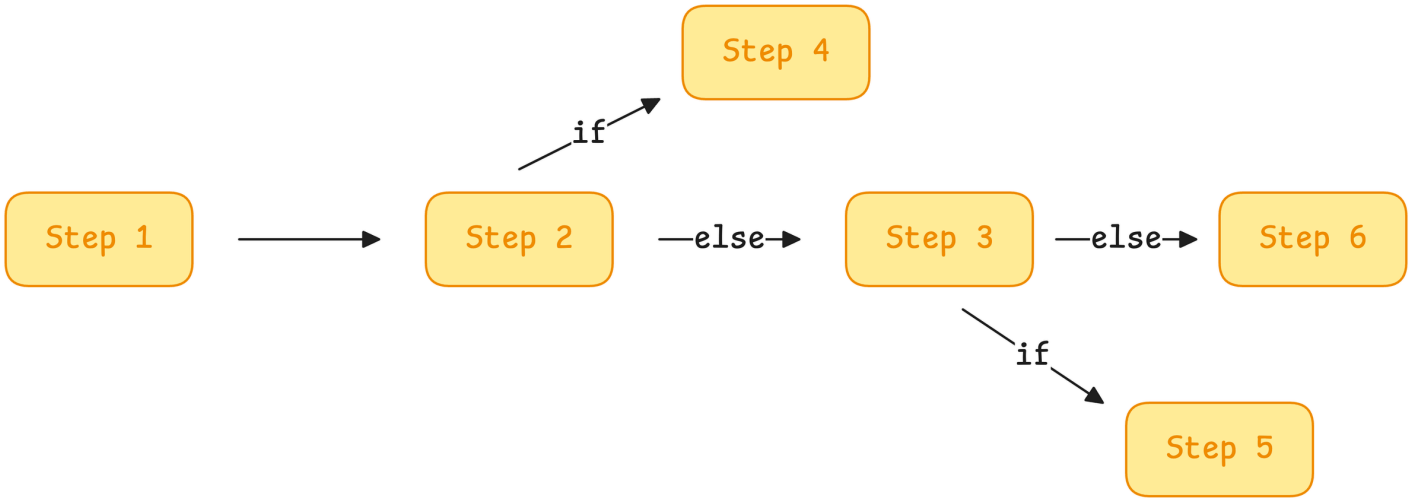
That is why the primary focus in 2024 was (and going ahead in 2025 will be) on building and shipping **AI Agents**—autonomous systems that can reason, think, plan, figure out the relevant sources and extract information from them when needed, take actions, and even correct themselves if something goes wrong.

2) Software development perspective

Think about how traditional software applications work. They follow rigid, pre-defined rules:

- If you want a program to complete a task, you explicitly define every step.

- As new scenarios arise, developers must add more conditions and logic.



- Over time, the system becomes harder to scale and maintain.

In other words, traditional automation requires strict pre-defined logic:

- If condition A is met, do X.
- If condition B is met, do Y.
- Else, do Z.

That's not all, even these are mostly fixed in traditional automation:

- Inputs have a defined data type (text, numeric, etc.)
- Transformations on that input are mostly fixed.
- Output types are also fixed.

> 💡 On a side note, this should not be considered as a negative point about traditional automation and we do not intend to discourage it. If it's suitable, you should 100% stick to traditional automation and software development and don't build agents if you don't have to. We'll understand why we say so below.

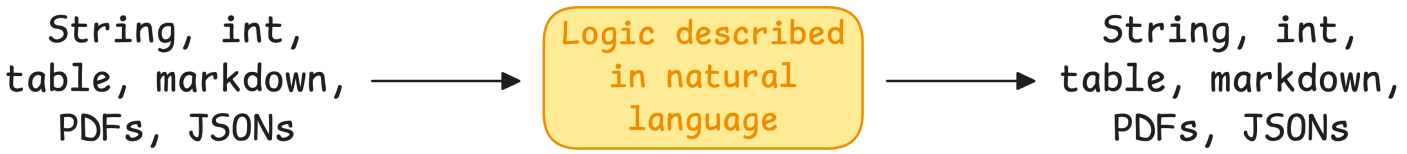
But all of that changes with AI Agents.

Firstly, AI agents don’t need explicit instructions for every case. Instead, they:

- Gather information dynamically.
- Use reasoning to handle ambiguous problems.
- Collaborate with other agents to complete tasks.
- Leverage external tools to make decisions in real time.

Moreover:

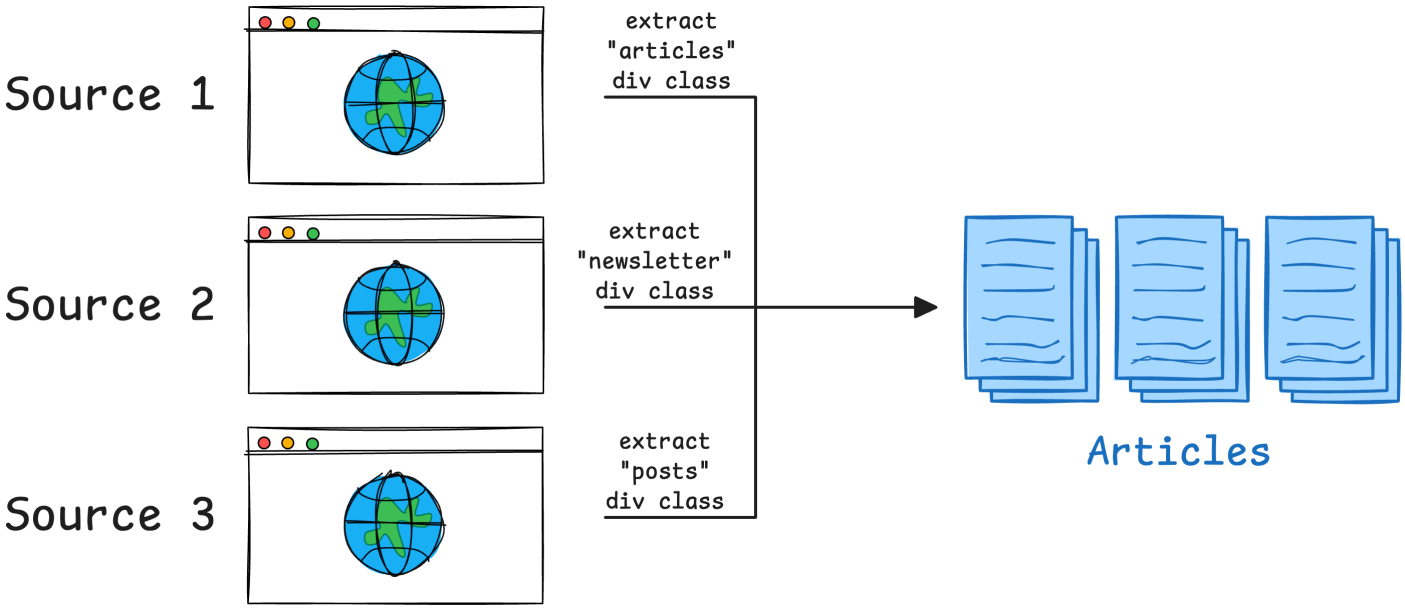
- Inputs do not have to be of any defined data type. It could be string, integer, PDFs, tables, markdown, JSON, etc.



- Transformations could be anything based on what you ask the LLM to do.
- Outputs can be anything—tokens, lists, structured output, JSON, code, etc.

Let's take an example to thoroughly understand the utility of AI Agents.

Imagine you run a news aggregation platform that pulls articles from different sources. Traditionally, you would:



1. Write scripts to scrape multiple websites.
2. Filter and categorize the articles based on hardcoded rules.
3. Manually verify whether an article is relevant.

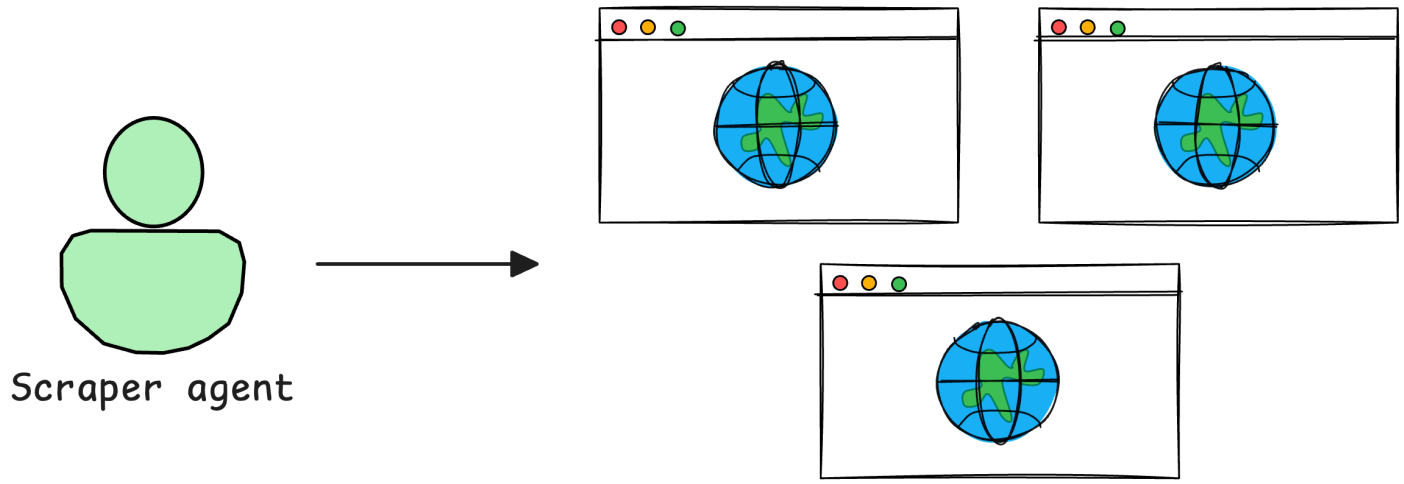
This works—but it's rigid.

Also, if a website changes its layout, your scraper breaks.

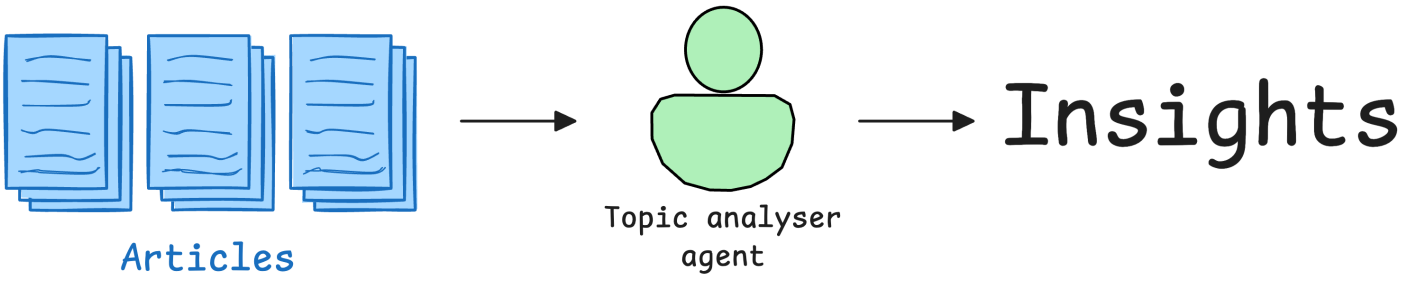
If new topics emerge, you have to update the filtering logic manually.

Now, let's see how AI agents would handle this differently:

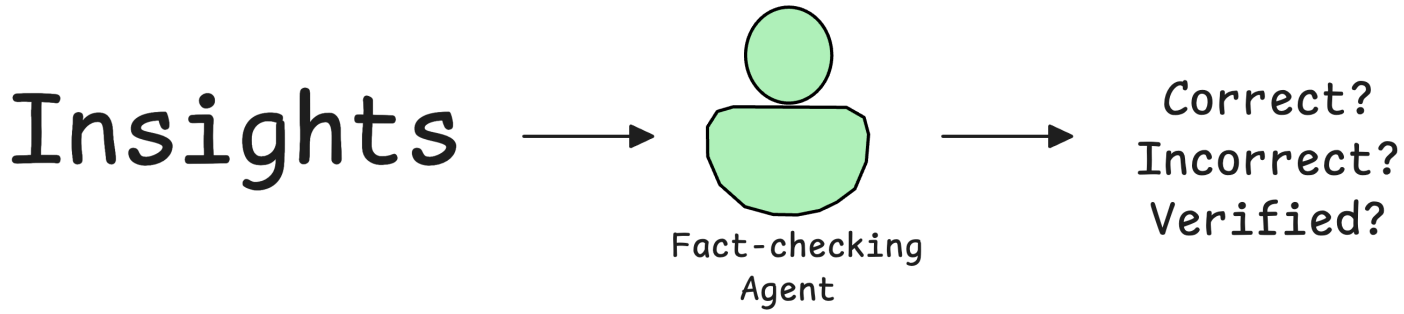
- A Web Scraper Agent autonomously finds new sources and adapts when page structures change.



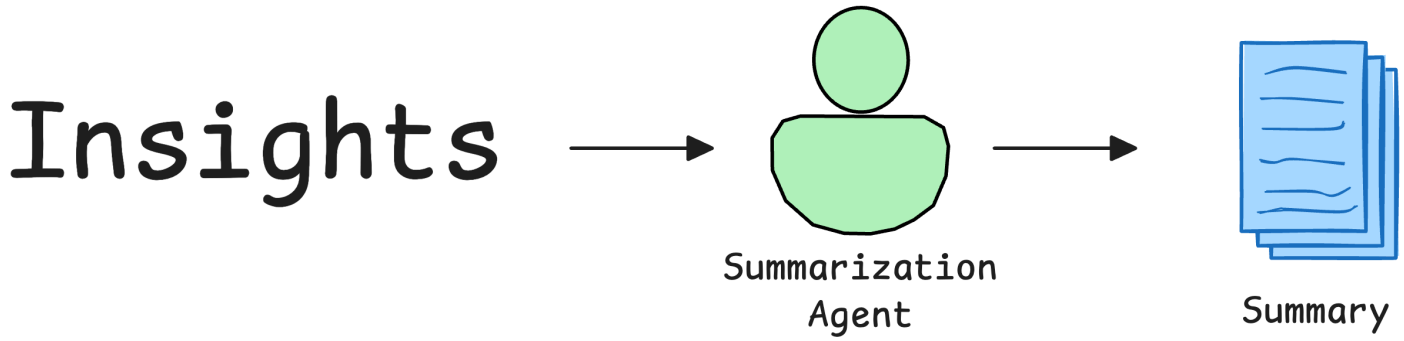
- A Topic Analysis Agent processes articles, detects emerging trends, and classifies them in real-time.



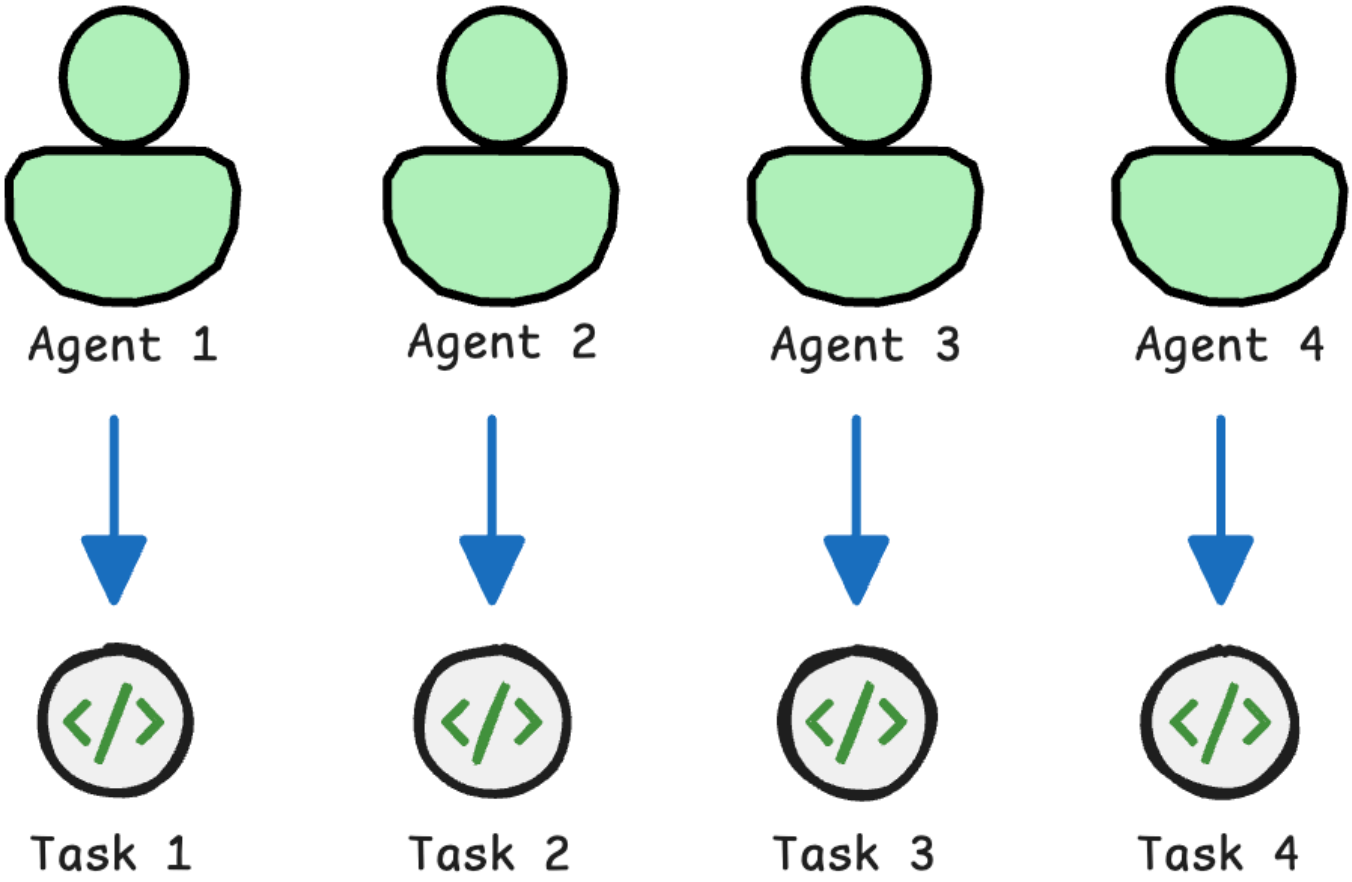
- A Fact-checking Agent verifies credibility using external data sources before surfacing an article.



- A Summarization Agent extracts key points and generates short, easy-to-read summaries.

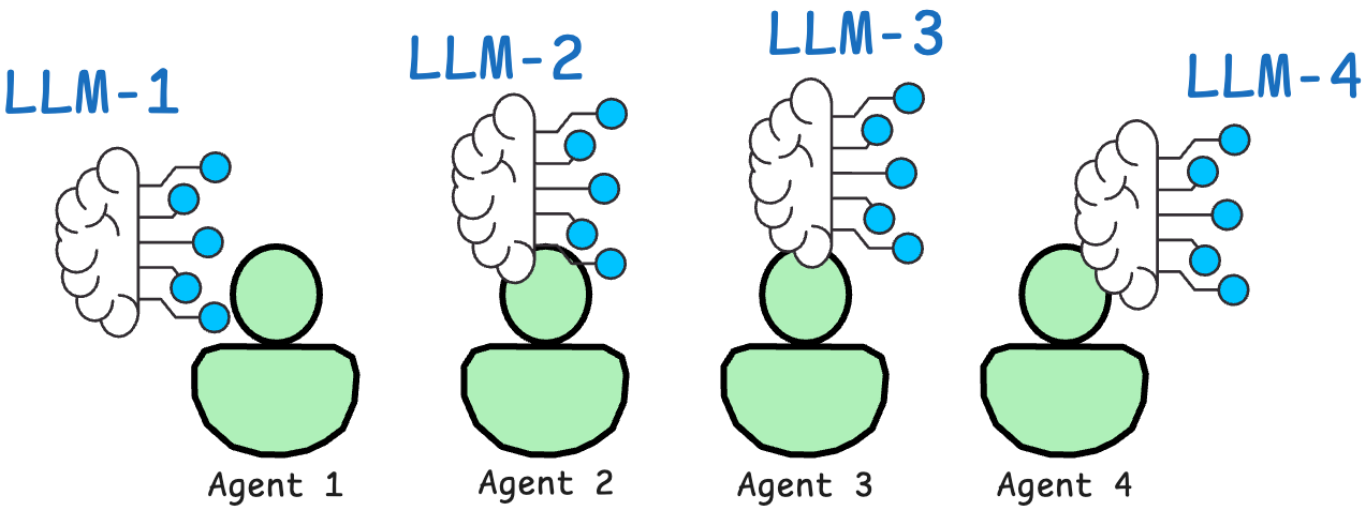


This is called a multi-agent framework, where you have multiple agents working together to achieve a common goal.



This, at least in the current state of AI, works better than building single-agent systems since each agent has a dedicated task it must complete.

Also, having multiple agents means each Agent can be backed by a different LLM.



For instance:

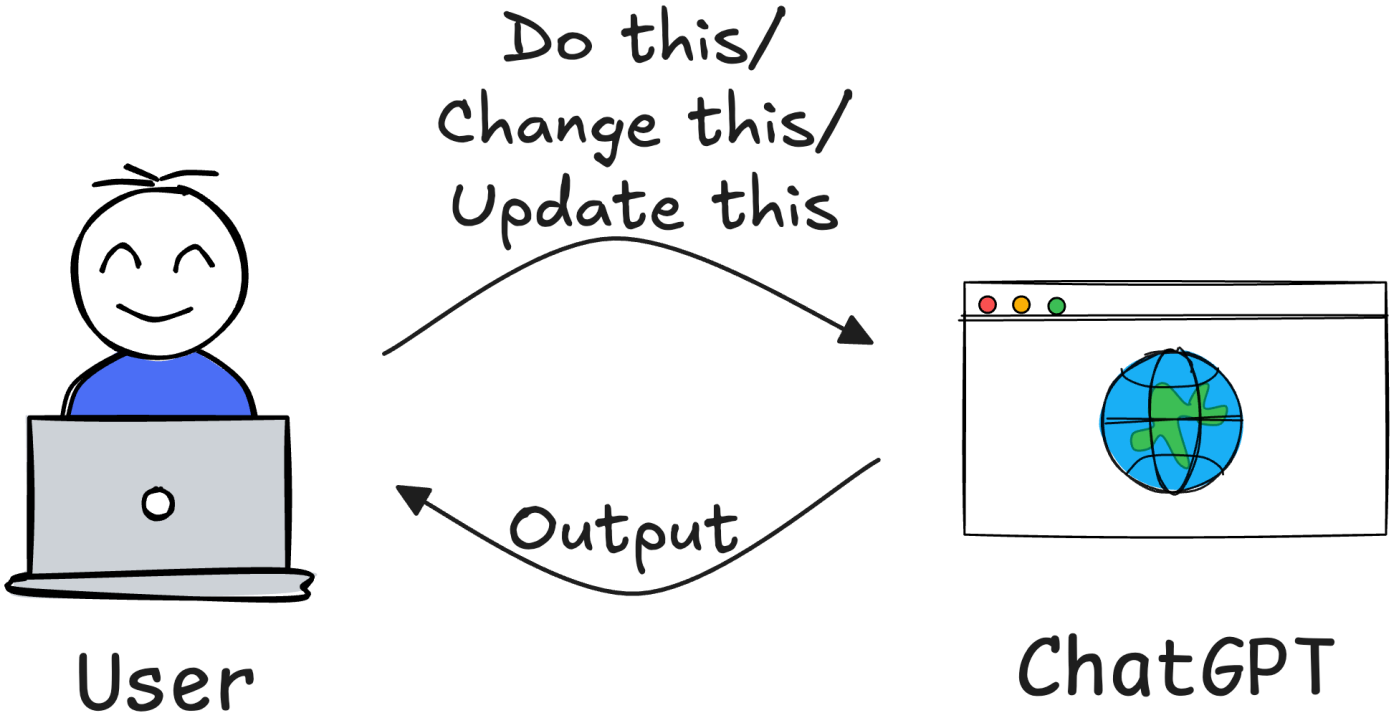
- If some LLM is better at summarization, we can use that as the summary LLM.
- Similarly, if an LLM is better at analyzing long texts, we can use it as the topic analysis Agent, and so on.

In the above news aggregation platform application, instead of relying on hard-coded rules, these agents learn, adapt, and optimize their processes dynamically. If a new trend emerges, the system automatically updates without requiring manual intervention.

This is the power of AI agents—they don't just execute predefined tasks; they discover problems, collaborate with other agents, and solve them autonomously, which takes us to the next point.

3) Autonomous system perspective

One of the biggest limitations of traditional LLM-based interactions—like ChatGPT—is that you have to constantly engage with the model to refine outputs.

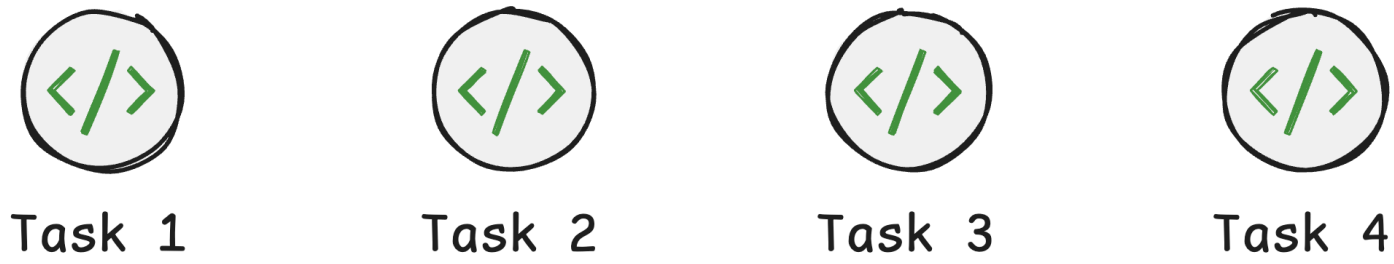


- You ask a question
- You review the response
- You tweak your prompt, ask again
- Repeat until you get the desired result.

But since LLMs are becoming significantly better at reasoning, we would want to leverage these capabilities and build systems that could autonomously complete tasks without constant user intervention.

AI Agents help us do that.

Instead of responding passively to individual queries, AI agents can break down a complex objective into sub-tasks, execute them step by step, and refine the output along the way—without needing constant feedback.



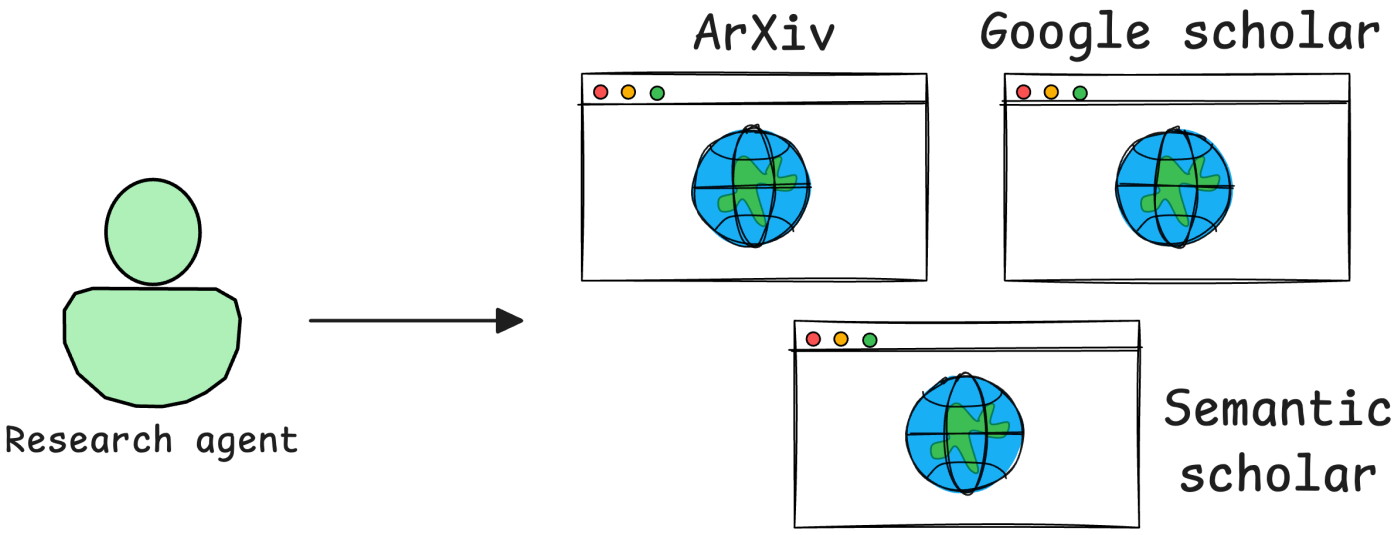
Imagine you want to generate a report on the latest trends in AI research. If you use a standard LLM, you might:

1. Ask for a summary of recent AI research papers.
2. Review the response and realize you need sources.
3. Obtain a list of papers along with citations.
4. Find that some sources are outdated, so you refine your query.
5. Finally, after multiple iterations, you get a useful output.

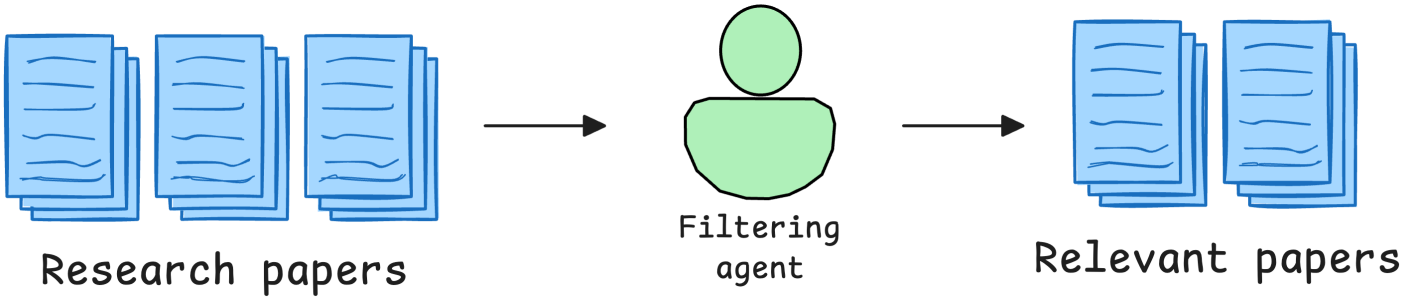
This iterative process takes time and effort, requiring you to act as the decision-maker at every step.

Now, let's see how AI agents handle this differently:

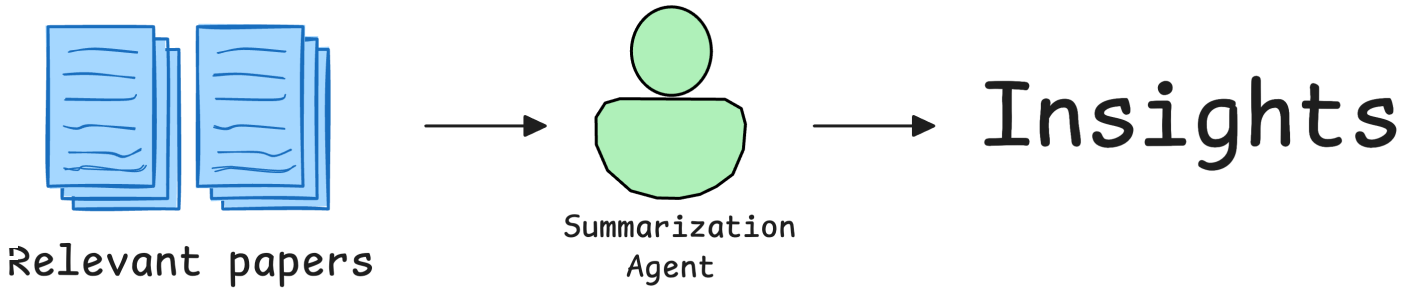
- A Research Agent autonomously searches and retrieves relevant AI research papers from arXiv, Semantic Scholar, or Google Scholar.



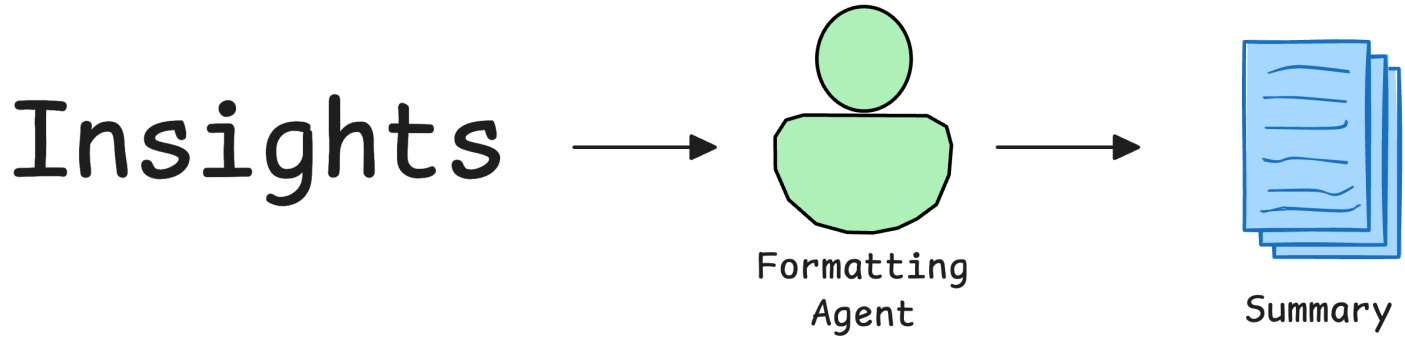
- A Filtering Agent scans the retrieved papers, identifying the most relevant ones based on citation count, publication date, and keywords.



- A Summarization Agent extracts key insights and condenses them into an easy-to-read report.



- A Formatting Agent structures the final report, ensuring it follows a clear, professional layout.



Here, the AI agents not only execute the research process end-to-end but also self-refine their outputs, ensuring the final report is comprehensive, up-to-date, and well-structured—all without requiring human intervention at every step.

This autonomous task-completion capability transforms how we interact with AI, shifting from one-time responses to goal-driven automation.

This explains the motivation behind AI Agents and why they are gaining so much traction.

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

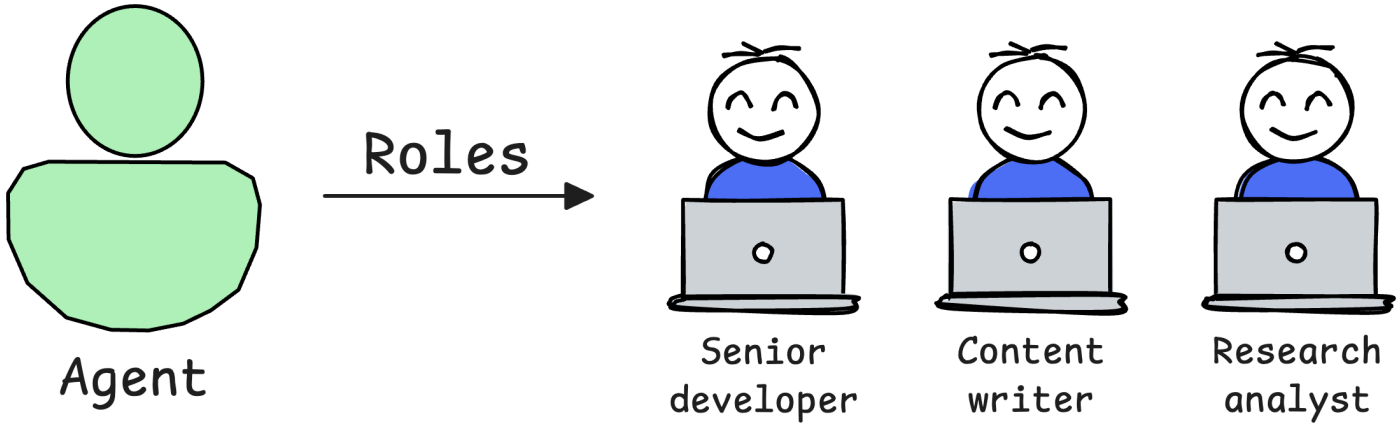
Let's explore each of these concepts and understand why they are fundamental to building great AI agents.

> 💡 We shall see all of this in action shortly through implementations.

1) Role-playing

One of the most impactful ways to improve an AI agent's effectiveness is to define its role clearly.

When an agent is given a well-defined identity, expertise, and objective, its responses become more structured, relevant, and aligned with expectations.



For example, asking a "*general AI assistant*" to summarize a legal contract may result in a vague and imprecise answer.

However, if you set the agent's role as "Senior *contract lawyer specializing in corporate law*", the response will be more precise, legally accurate, and context-aware.

This approach works because role assignment influences the reasoning process, ensuring the agent retrieves and generates information with domain-specific knowledge.

Thus, the best practice is to assign agents clear roles and objectives that align with their tasks. The more specific and contextual the role, the better the response quality.

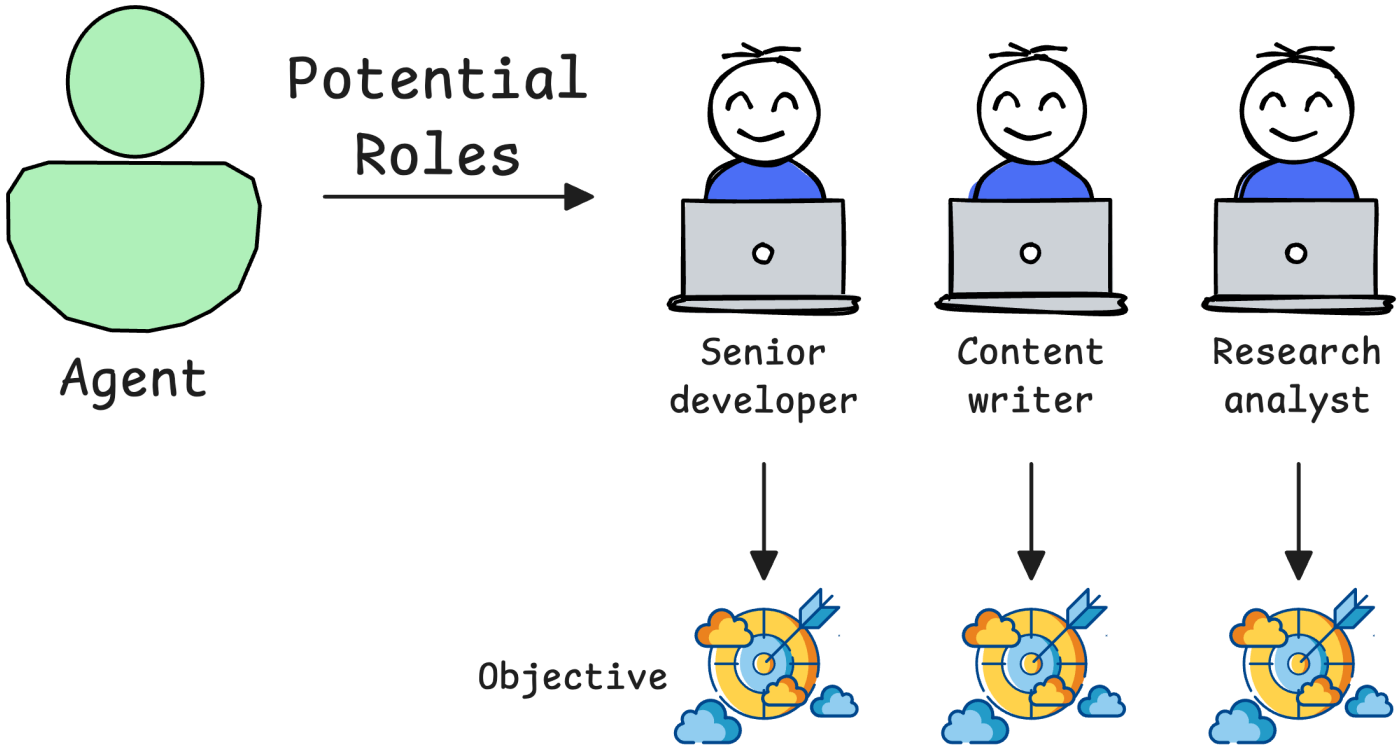
2) Focus/Tasks

Focus is critical for reducing hallucinations, improving accuracy, and ensuring consistency in multi-agent systems.

While modern LLMs come with large context windows, simply feeding them excessive data doesn't guarantee better results.

Research shows that overloading an agent with too many tasks, too much information, or conflicting objectives can cause it to lose focus and degrade performance.

For instance, if an AI agent is responsible for writing marketing copy, it should only focus on tasks like brand messaging, tone, and audience engagement.



Adding extra responsibilities—like product pricing analysis or competitive intelligence—may dilute its effectiveness.

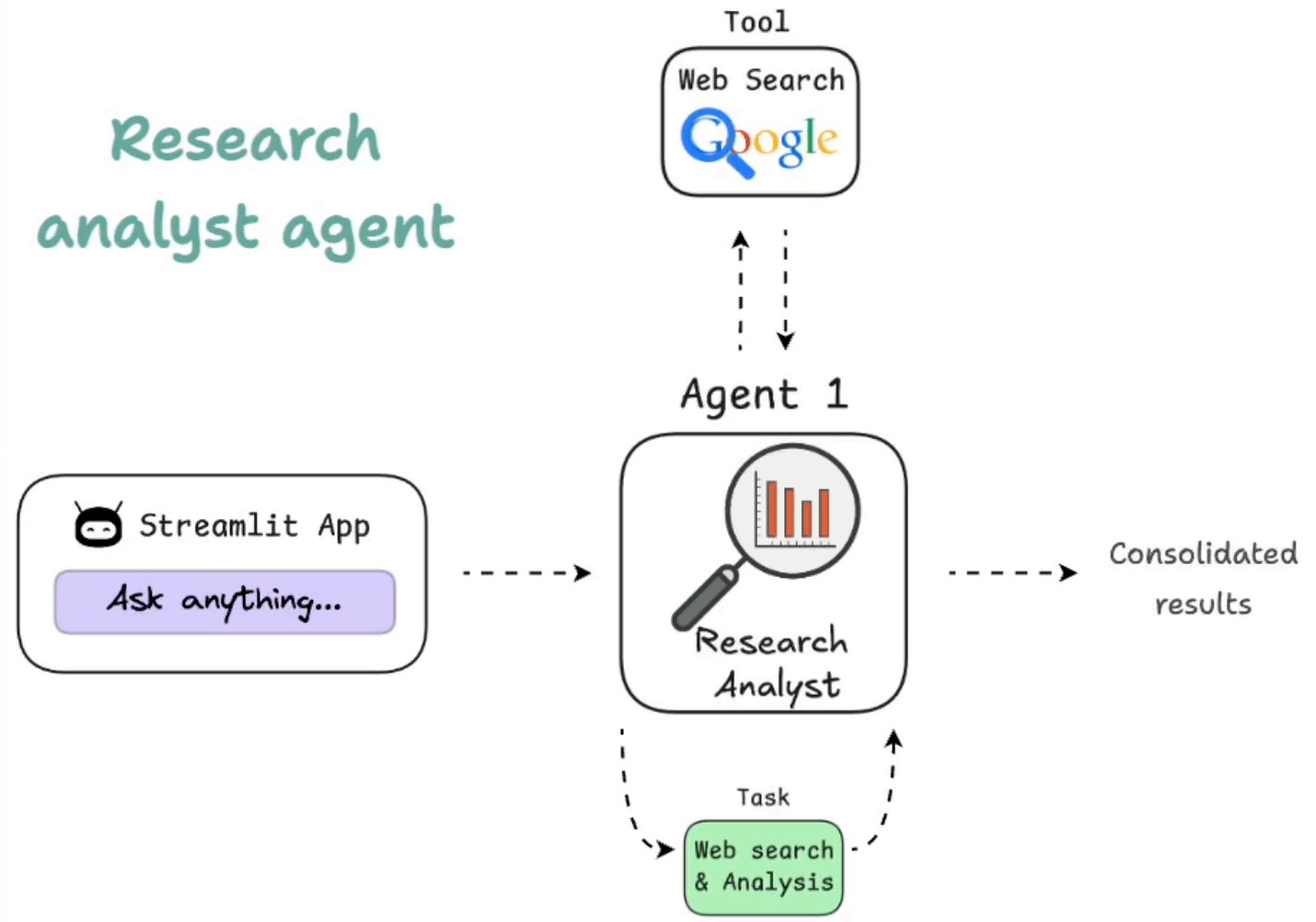
Instead of trying to make one agent do everything, a better approach is to use multiple agents, each with a specific and narrow focus.

Thus, the best practice is to keep agents as specialized as possible. A well-defined scope ensures higher accuracy and reliability.

3) Tools

AI agents are far more powerful when they can use external tools.

However, the key to success lies in choosing the right tools and not overwhelming the agent with too many options.



For example, an AI research agent could benefit from:

- A web search tool for retrieving recent publications.
- A summarization model for condensing long research papers.
- A citation manager to properly format references.

But if you add unnecessary tools—like a speech-to-text module or a code execution environment—it could confuse the agent and reduce efficiency.

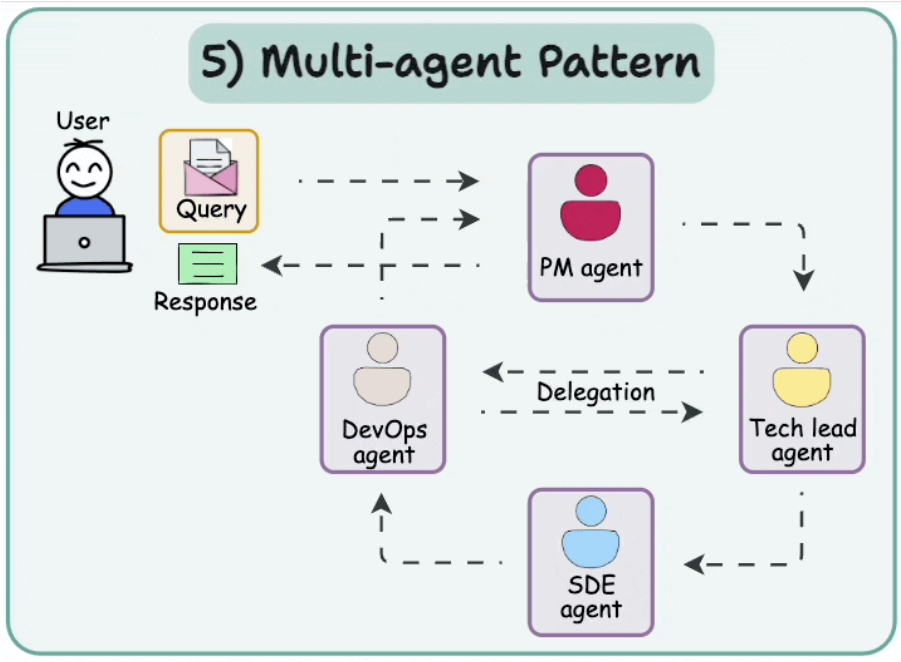
Thus, the best practice is to equip a particular agent with only the essential tools they need to complete their tasks effectively.

More tools ≠ better results.

4) Cooperation

Multi-agent systems work best when agents collaborate and exchange feedback.

Instead of relying on a single, monolithic agent, a network of specialized agents can work together to improve decision-making and task execution.



Just like we discussed earlier, consider an AI-powered financial analysis system:

- A Data Collection Agent gathers real-time stock market data.
- A Risk Assessment Agent evaluates investment risks based on historical trends.

- A Portfolio Strategy Agent recommends optimal investment strategies.
- A Report Generation Agent compiles findings into a user-friendly summary.

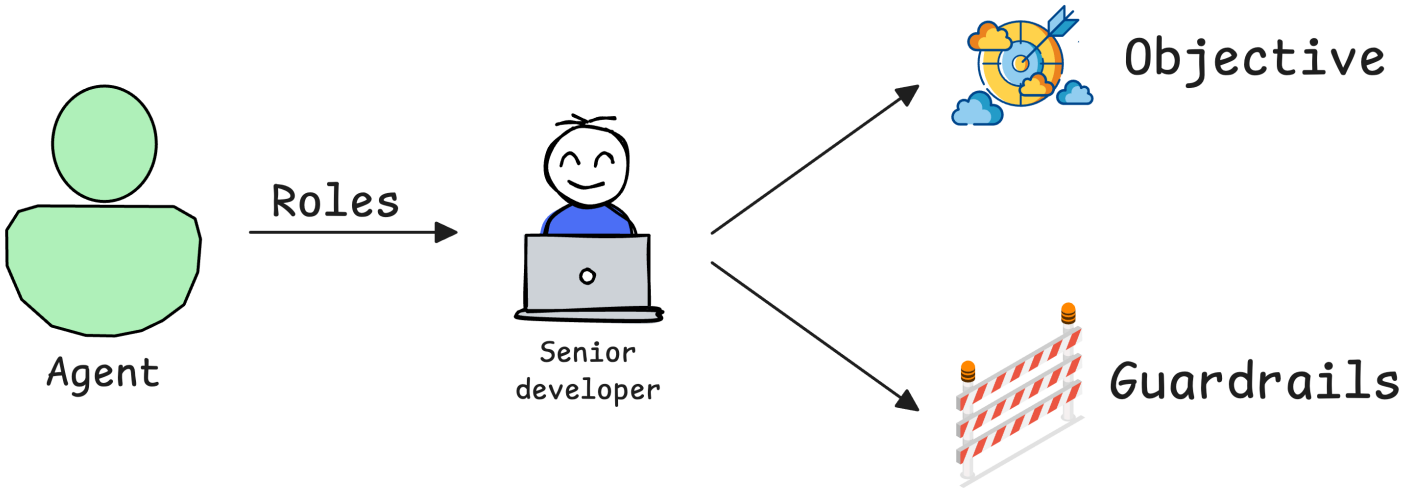
By dividing tasks and sharing context, agents can refine each other's outputs and produce more accurate and structured results.

In this case, the best practice is to enable agent collaboration by designing workflows where agents can exchange insights and refine their responses together.

5) Guardrails

AI agents are powerful, but without constraints and safeguards, they can hallucinate, enter infinite loops, or make unreliable decisions.

Guardrails ensure that agents stay on track and maintain quality standards.



Examples of useful guardrails include:

- Limiting tool usage: Prevent an agent from overusing APIs or generating irrelevant queries.
- Setting validation checkpoints: Ensure outputs meet predefined criteria before moving to the next step.

- Establishing fallback mechanisms: If an agent fails to complete a task, another agent or human reviewer can intervene.

For instance, in an AI-powered legal assistant, guardrails can prevent the system from:

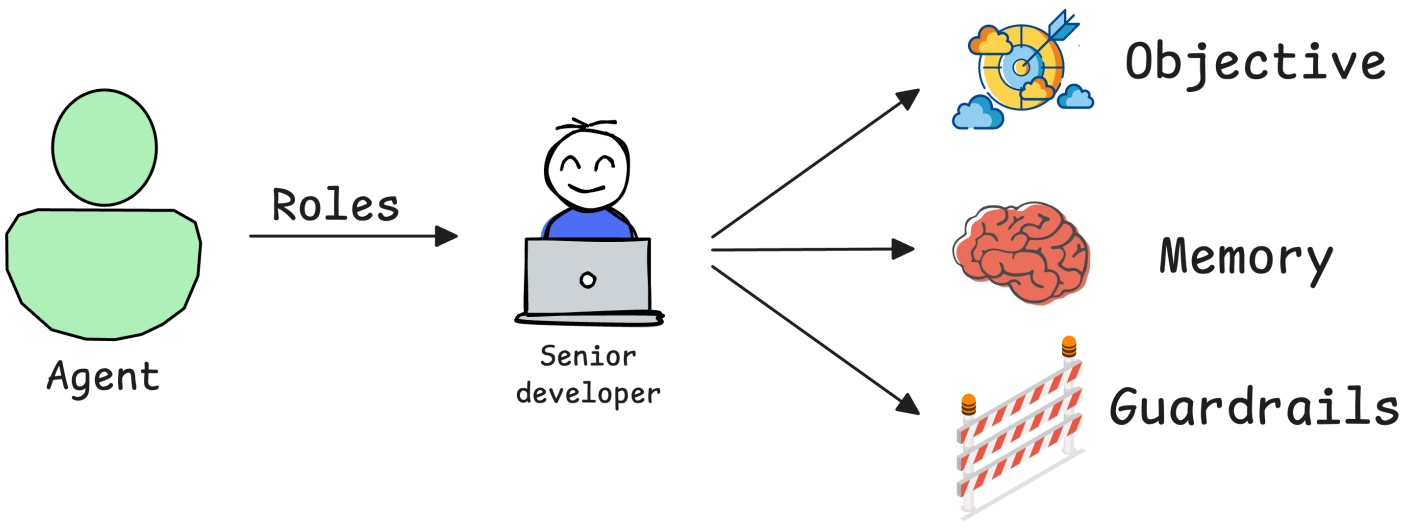
- Generating non-factual legal advice.
- Misinterpreting laws that differ across jurisdictions
- Citing outdated or incorrect legal precedents.

Thus, if you can, you should implement safeguards, validation layers, and fallback mechanisms to keep agents aligned with expected behavior.

6) Memory

Finally, we have memory, which is one of the most critical components of AI agents.

Without memory, an agent would start fresh every time, losing all context from previous interactions. With memory, agents can improve over time, remember past actions, and create more cohesive responses.



Different types of memory in AI agents include:

- Short-term memory – Exists only during execution (e.g., recalling recent conversation history).
- Long-term memory – Persists after execution (e.g., remembering user preferences over multiple interactions).
- Entity memory – Stores information about key subjects discussed (e.g., tracking customer details in a CRM agent).

For example, in an AI-powered tutoring system, memory allows the agent to:

- Remember a student's previous lessons and progress.
- Tailor future recommendations based on past learning history.
- Avoid repeating the same explanations unnecessarily.

Without memory, the agent would treat every session as a new interaction, making it far less useful.

As a takeaway, implement short-term and long-term memory so agents can learn, adapt, and provide personalized experiences over time.

> 💡 When designing agentic systems, think of yourself as a manager hiring a team.
>
> • Define the goal – What is the specific outcome you want the agent to achieve?
> • Establish the process – What steps should the agent follow to reach that goal?
> • Hire the right experts – If this were a real-world job, what kind of specialists would you bring in?
> • Give them the right tools – Once you have hired the experts, what tools would they need access to to complete their job effectively?
>
> For example, instead of a generic "researcher" agent, create a "market research analyst specializing in AI trends." Instead of a "writer" agent, design a "senior technical content strategist."

By giving agents clear roles, expertise, purpose, and tools, you make them more effective, structured, and capable of delivering high-quality results. Specificity matters.

Building AI Agent systems

With that, we are ready to dive into the implementation of multi-agent systems.

To do this, 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 - crewAIInc/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...

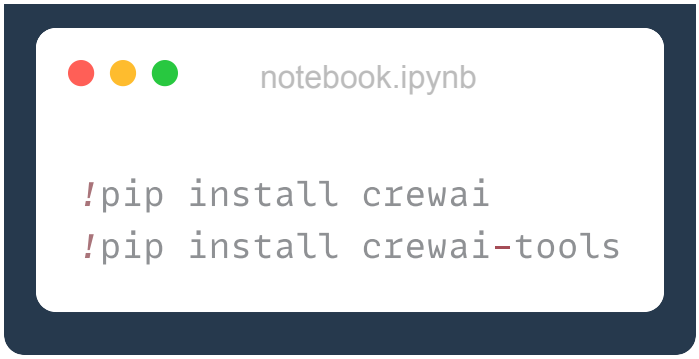
GitHub • crewAIInc

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:



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.

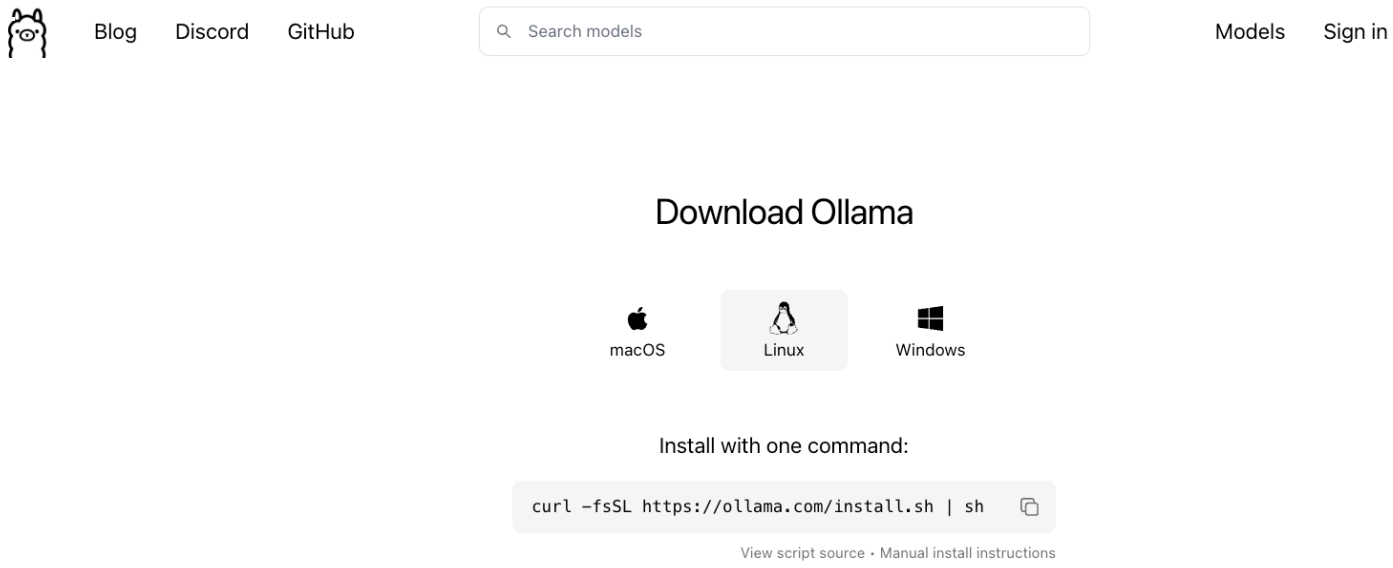
To set up OpenAI, create a `.env` file in the current directory and specify your OpenAI API key as follows:

```
.env

OPENAI_API_KEY="sk-38381...."
```

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

- Go to **Ollama.com**, select your operating system, and follow the instructions.

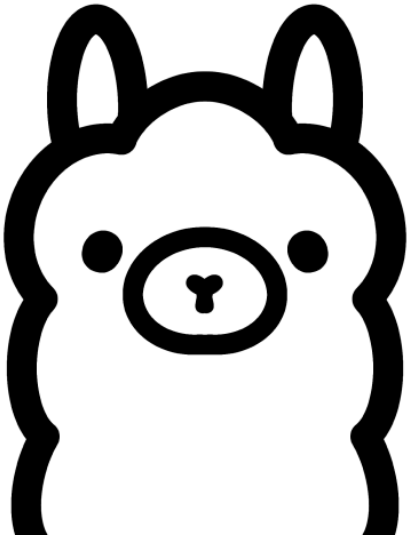


  - If you are using Linux, you can run the following command:

Command line

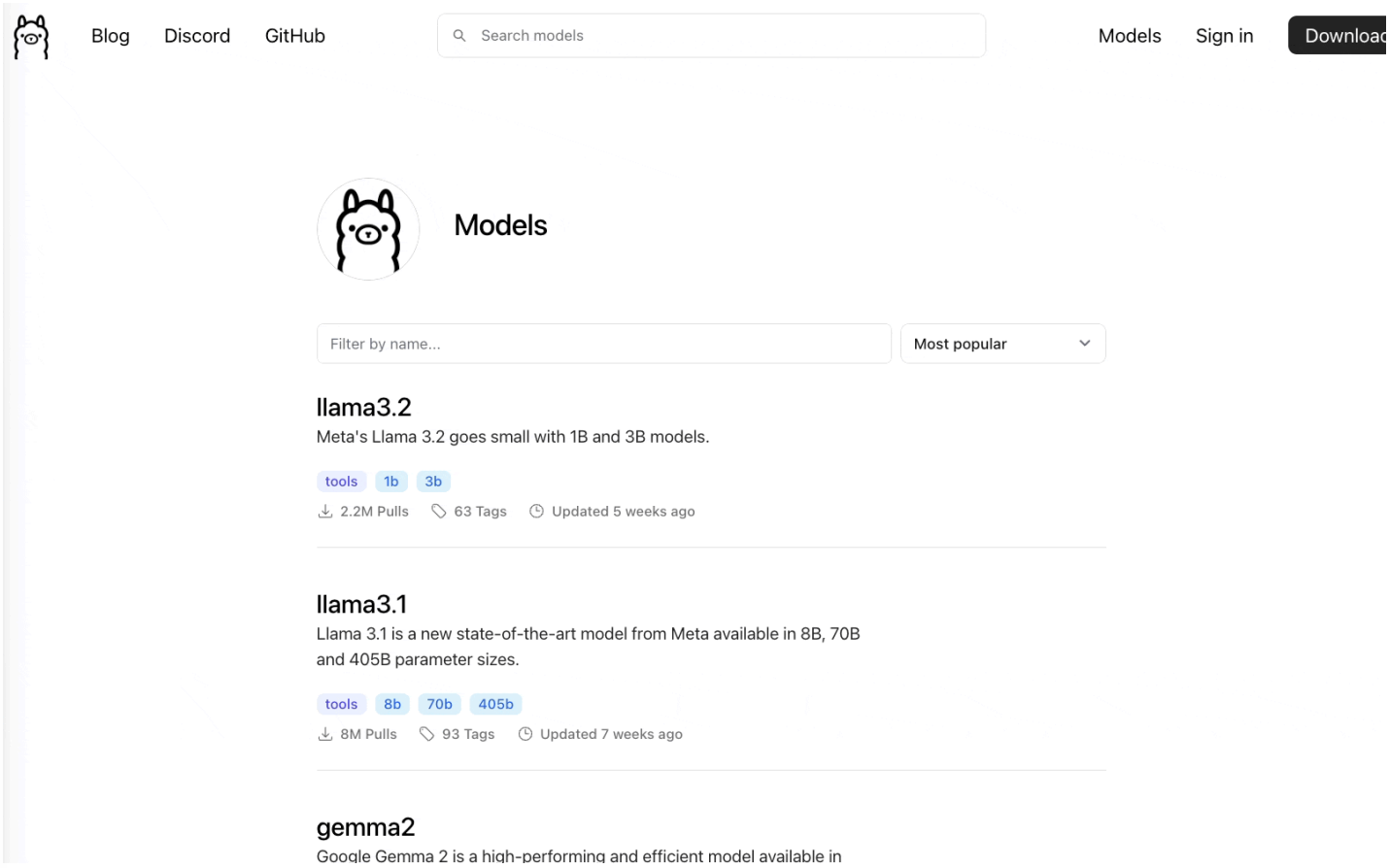
```
curl -fsSL https://ollama.com/install.sh | sh
```

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

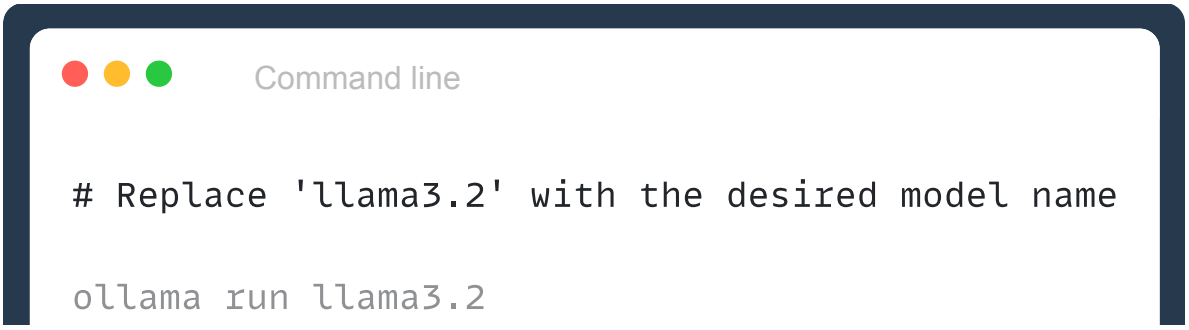


library

Get up and running with large language models.



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



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:

0:00 / 0:29

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

Define agents

In CrewAI, an `Agent` is defined as an autonomous entity that:

- Has a `Role` —its function and expertise within the system, like a "*Senior Technical Writer*"
- Has a `Goal` —its individual objective within the system, like "*Craft a publication-ready article*"
- Has a `Backstory` —it provides context and personality to the agent, enriching interactions, like "*You're a seasoned researcher with a knack for uncovering the latest developments in AI. Known for your ability to find the most relevant information and present it in a clear and concise manner.*"

Moreover, an Agent can:

- Use tools to accomplish the Goal.
- Make decisions based on the Goal and the Role.
- Collaborate with other agents.
- Maintain memory of interactions.
- Delegate tasks if needed (and if allowed to do so).

We can define an `Agent` in CrewAI as follows:

notebook.ipynb

```python
from crewai import Agent

senior_technical_writer = Agent(
    role="Senior Technical Writer",
    goal="""Craft clear, engaging, and well-structured
            technical content based on research findings""",
    backstory="""You are an experienced technical writer
                 with expertise in simplifying complex
                 concepts, structuring content for readability,
                 and ensuring accuracy in documentation.""",
    verbose=True  # Enable logging for debugging
)
```

Agent's Role

Agent's Goal

Agent's Backstory

💡 The `verbose` parameter is used to enable detailed execution logs for debugging. It is set to `False` by default. We'll understand more parameters shortly.

We haven't specified the `llm` yet, which determines the language model used. It defaults to OpenAI's GPT-4. But we can use Ollama as follows:

notebook.ipynb

```python
from crewai import LLM

llm = LLM(
    model="ollama/llama3.2:1b",
    base_url="http://localhost:11434"
)
```

Once done, we can pass this `llm` object as the `llm` parameter of the `Agent` class:

notebook.ipynb

```
from crewai import Agent

senior_technical_writer = Agent(

    role="Senior Technical Writer",

    goal="""Craft clear, engaging, and well-structured
            technical content based on research findings""",

    backstory="""You are an experienced technical writer
                 with expertise in simplifying complex
                 concepts, structuring content for readability,
                 and ensuring accuracy in documentation.""",

    llm=llm,  LLM for the agent

    verbose=True
)
```

Done!

We have defined our Agent.

> 👉 We'll look at integrating tools shortly.

Similarly, you can define any other Agent as needed according to your use case. The key is to carefully craft their role, goal, and backstory so they behave as intended within an entire system.

👉 Spend good amount of time in defining the Goal, Role and Backstory. That is what controls most of the behaviour of the Agent. Also, in Role parameter specifically, keywords like "Senior", "Expert", "Head" are extremely useful in our observation.

Here are some more examples:

1) Research Analyst Agent: This agent is responsible for conducting in-depth research and summarizing findings concisely.

notebook.ipynb

```
from crewai import Agent

research_analyst = Agent(
    role="Senior Research Analyst",
    goal="""Find, analyze, and summarize information
            from various sources to support technical
            and business-related inquiries.""",
    backstory="""You are a skilled research analyst with expertise
                 in gathering accurate data, identifying key trends,
                 and presenting insights in a structured manner.""",
    llm=llm,
    verbose=True
)
```

2) Code Review Agent: This agent specializes in reviewing code for errors, optimizations, and best practices.

notebook.ipynb

```
from crewai import Agent

code_reviewer = Agent(
    role="Senior Code Reviewer",
    goal="""Review code for bugs, inefficiencies, and
            security vulnerabilities while ensuring adherence
            to best coding practices.""",
    backstory="""You are a seasoned software engineer with years of
                 experience in writing, reviewing, and optimizing
                 production-level code in multiple programming languages.""",
    llm=llm,
    verbose=True
)
```

In this case, if you know the programming language, then it is even recommended to get more specific and specify that in the backstory.

3) Legal Document Reviewer: This agent specializes in analyzing legal contracts and ensuring compliance.

notebook.ipynb

```
from crewai import Agent

legal_reviewer = Agent(
    role="Legal Document Expert Reviewer",
    goal="""Review contracts and legal documents to
            ensure compliance with applicable laws and
            highlight potential risks.""",
    backstory="""You are a legal expert with deep knowledge
                 of contract law, regulatory frameworks,
                 and risk mitigation strategies.""",
    llm=llm,
    verbose=True
)
```

Hope you have understood the point.

Define tasks

Agents can be thought of as professionals/characters in our AI workflow—Senior Writer, Research Analyst, etc.

To actually build and run a workflow, we need to define tasks for these characters (Agents).

In CrewAI, a `Task` is a specific assignment given to an AI agent. Tasks define what needs to be done, who is responsible, and how it should be executed, ensuring that agents can work efficiently, either independently or collaboratively.

A `Task` in CrewAI consists of:

- A Clear Description – Defines the action an agent must complete.
- An Assigned Agent – Specifies which agent is responsible.
- Execution Strategy – Determines whether tasks run sequentially or hierarchically.
- Dependencies (if any) – Defines whether a task requires input from another agent.

A simple task for the Senior Technical Writer Agent created earlier is defined below:

notebook.ipynb

```
from crewai import Task

writing_task = Task(
    description="""Write a well-structured, engaging,
                   and technically accurate article
                   on {topic}.""",
    agent=senior_technical_writer,
    expected_output="""A polished, detailed, and easy-to-read
                       article on the given topic.""",
)
```

Task description

The Agent it belongs to

The expected output

- Description: Clearly defines what the agent must accomplish.
- Assigned Agent: Specifies who is responsible for executing the task (`senior_technical_writer`).
- Expected output: Ensures there's a well-defined goal for evaluating success.

Here, if you notice closely, we have specified a parameter in the above task:

notebook.ipynb

```
from crewai import Task

writing_task = Task(
    description="""Write a well-structured, engaging,
                   and technically accurate article
                   on {topic}.""",

    agent=senior_technical_writer,


    expected_output="""A polished, detailed, and easy-to-read
                       article on the given topic.""",
)
```

Topic parameter

As we shall see below, this is a parameter that is passed to the workflow when we run the Agent application.

Let's see this below.

Create Crew

So far, we've defined our Senior Technical Writer Agent and created a task for it. However, AI agents don't work in isolation—they often need to collaborate with other agents to complete complex workflows.

That's why we create a `Crew`, which is a structured team of agents that work together to accomplish a goal. It orchestrates how tasks are assigned, executed, and passed between agents.

For our single-agent app, we do this as follows:

notebook.ipynb

```
from crewai import Crew

crew = Crew(
    agents=[senior_technical_writer],
    tasks=[writing_task],
    verbose=True
)

response = crew.kickoff(inputs={"topic":"AI Agents"})
```

Topic parameter

In this code:

- Agents: A list of all AI agents that will work together in the Crew.
- Tasks: A list of all tasks that need to be completed.
- Verbose mode: Provides debugging information on execution.

Also, once the Crew has been defined, we run it using the kickoff method as depicted above.

The verbose mode produces the output you see in green and in purple, which provides detailed information on the execution.

```python
from crewai import Crew

crew = Crew(
    agents=[senior_technical_writer],
    tasks=[writing_task],
    verbose=True
)

response = crew.kickoff(inputs={"topic":"AI Agents"})
```

✓ 31.3s Python

```
Overriding of current TracerProvider is not allowed
# Agent: Senior Technical Writer
## Task: Write a well-structured, engaging, and
                  technically accurate article on AI Agents.


# Agent: Senior Technical Writer
## Final Answer:
# Understanding AI Agents: A Comprehensive Overview

## Introduction to AI Agents

Artificial Intelligence (AI) Agents are autonomous or semi-autonomous systems that perceive their environment, reason about what they perceive, and take actions to achieve sp

## What is an AI Agent?

An AI agent is defined as a system that perceives its environment through sensors and acts upon that environment with actuators based on a set of rules or algorithms. The ter

### Components of AI Agents

AI agents typically consist of the following components:

1. **Perception**: This is the agent's ability to sense or acquire information about its environment using sensors. For example, a robotic vacuum uses lidar sensors to map ou

2. **Decision-Making**: With the collected data, the AI agent processes information through algorithms (such as decision trees, neural networks, etc.) to decide how to act.

3. **Action**: Once a decision is made, the agent takes action using actuators, which can be motors, speakers, or any other interface that interacts with the environment.

4. **Learning**: Many AI agents incorporate machine learning techniques to adapt their actions over time based on past experiences and outcomes.

## Types of AI Agents
```

Due to verbose mode

💡 We are using a weak LLM so you may not get a perfect response the first time. Consider replacing it with some powerful models like DeepSeek, GPT-4, etc. We discussed the steps earlier in the article.

Printing the response, we get:

```python
from IPython.display import Markdown

Markdown(response.raw)
```

✓ 0.0s Python

Understanding AI Agents: A Comprehensive Overview

Introduction to AI Agents

Artificial Intelligence (AI) Agents are autonomous or semi-autonomous systems that perceive their environment, reason about what they perceive, and take actions to achieve specific goals. From voice assistants like Siri and Alexa to sophisticated self-driving cars, AI agents are massively transforming industries and enhancing user experience across various domains. This article aims to provide a thorough understanding of AI agents, their functionalities, types, and real-world applications.

What is an AI Agent?

An AI agent is defined as a system that perceives its environment through sensors and acts upon that environment with actuators based on a set of rules or algorithms. The term "agent" implies autonomy; that is, the system can make decisions without human intervention based on the defined criteria and environmental input.

Components of AI Agents

AI agents typically consist of the following components:

1. **Perception**: This is the agent's ability to sense or acquire information about its environment using sensors. For example, a robotic vacuum uses lidar sensors to map out the room.
2. **Decision-Making**: With the collected data, the AI agent processes information through algorithms (such as decision trees, neural networks, etc.) to decide how to act.
3. **Action**: Once a decision is made, the agent takes action using actuators, which can be motors, speakers, or any other interface that interacts with the environment.
4. **Learning**: Many AI agents incorporate machine learning techniques to adapt their actions over time based on past experiences and outcomes.

Types of AI Agents

AI agents can be categorized based on their capabilities and functionalities:

1. **Reactive Agents**: These agents operate solely based on the current state of the environment without memory of past states. An example is a chess AI that evaluates possible moves without recalling past games.
2. **Deliberative Agents**: These agents maintain a representation of the world and plan their actions based on future goals. They can predict the outcomes of actions, which enables them to choose the best course of action.
3. **Learning Agents**: These agents use machine learning models to continuously improve their performance. They learn from interaction and modify their actions based on feedback.
4. **Hybrid Agents**: These combine various approaches from both reactive and deliberative agents, allowing them to adapt to rapidly changing environments while strategically planning.

How AI Agents Work

AI agents operate on principles derived from artificial intelligence, computational theories, and cognitive sciences. Their functioning involves several stages:

1. **Observation**: The agent gathers data from the environment. This data can be continuous (like temperature readings) or discrete (like sensor inputs on a vehicle).
2. **Data Processing**: The collected data is processed using algorithms, which help filter noise and identify significant patterns or trends.
3. **Decision-Making**: Based on the analyzed data, the agent formulates a response. In a dynamic environment, it might adapt its decision-making in real-time.
4. **Action Execution**: The final step involves executing the actions to influence the environment, such as speaking back to a user or changing the course of a vehicle.
5. **Feedback Loop**: After executing actions, agents often evaluate the outcomes, allowing them to learn and refine future decisions based on success or failure.

Applications of AI Agents

The utility of AI agents spans multiple industries, including:

- **Healthcare**: AI agents assist in diagnostic processes, patient management, and personalized treatment recommendations.
- **Finance**: They help in fraud detection, algorithmic trading, and portfolio management.
- **Customer Service**: AI-powered chatbots provide instant customer support, handling inquiries and solving issues round-the-clock.
- **Autonomous Vehicles**: Self-driving cars function as AI agents that perceive traffic conditions, navigate safely, and make real-time driving decisions.
- **Smart Homes**: Devices like smart thermostats and voice assistants help automate home tasks based on user preferences and behavior patterns.

Done!

Wasn't that simple?

Integrate tools

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 bridge this gap by allowing agents 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.

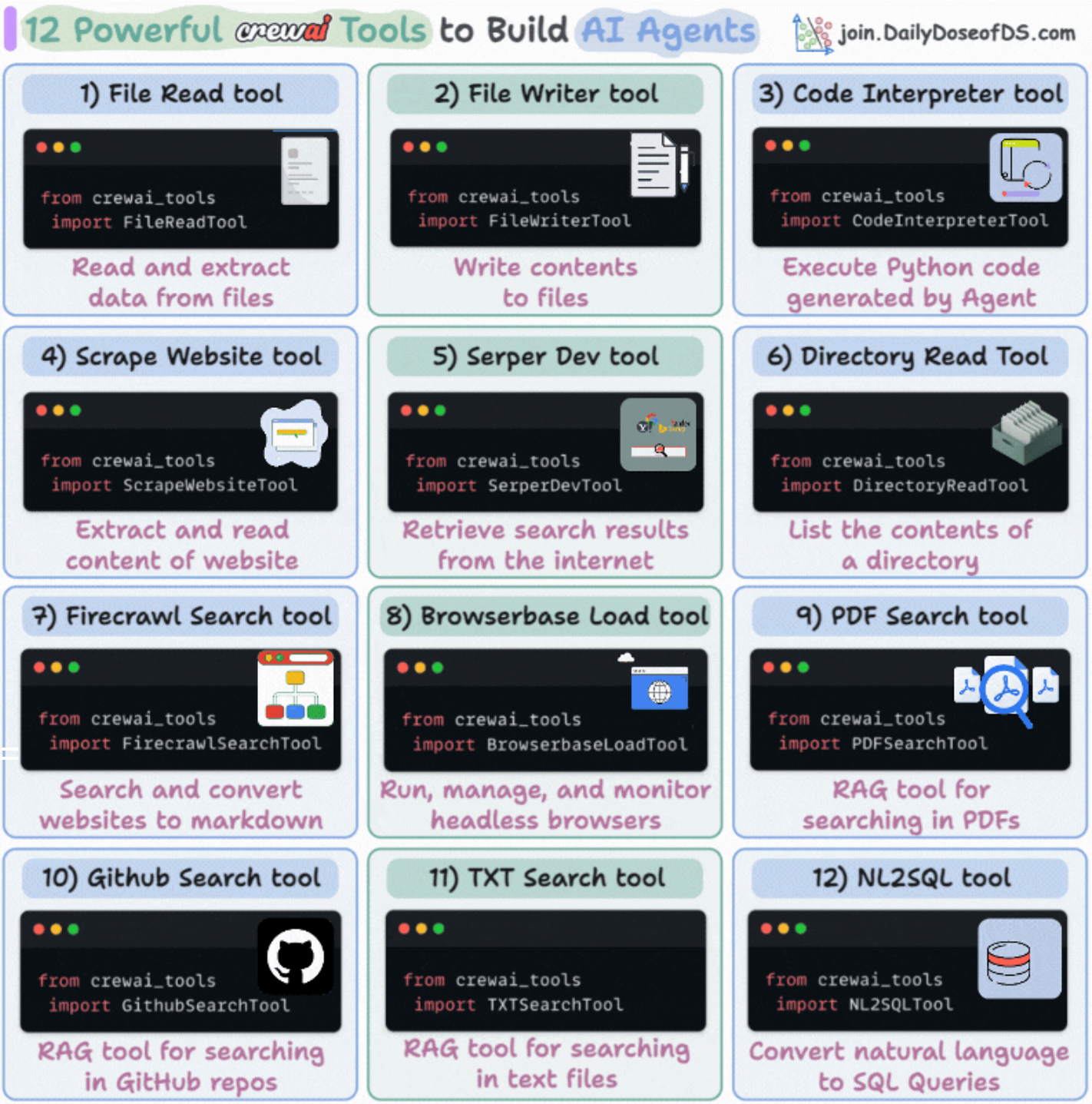
In short, tools empower AI agents to interact with the world, making them more dynamic, intelligent, and actionable.

For instance, consider an agent responsible for market research. Without tools, it can only generate insights based on its training data (which may be outdated).

However, with access to a real-time search tool, it can:

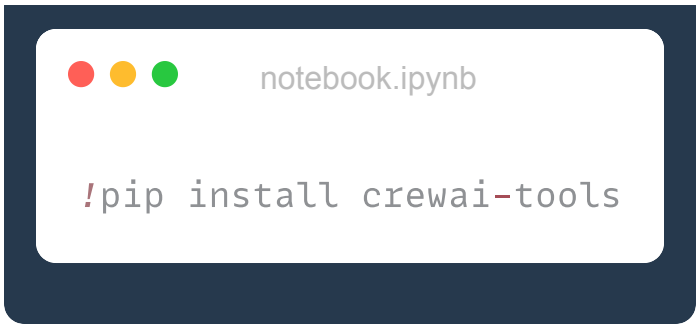
- Retrieve the latest stock prices and financial reports.
- Monitor news updates about companies and industries.
- Compare market trends with historical data.

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



Let's look at how to integrate tools into these Agentic workflows.

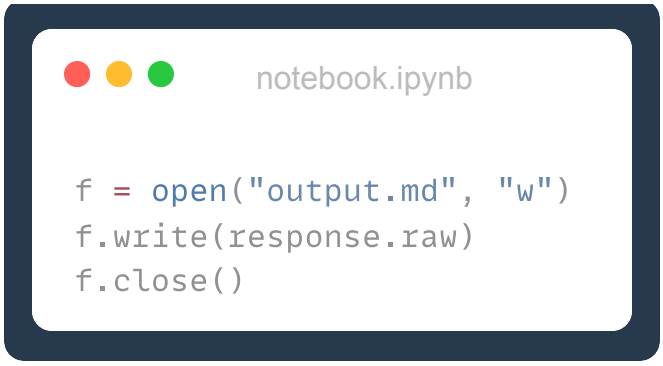
Firstly, make sure the tools package is installed:



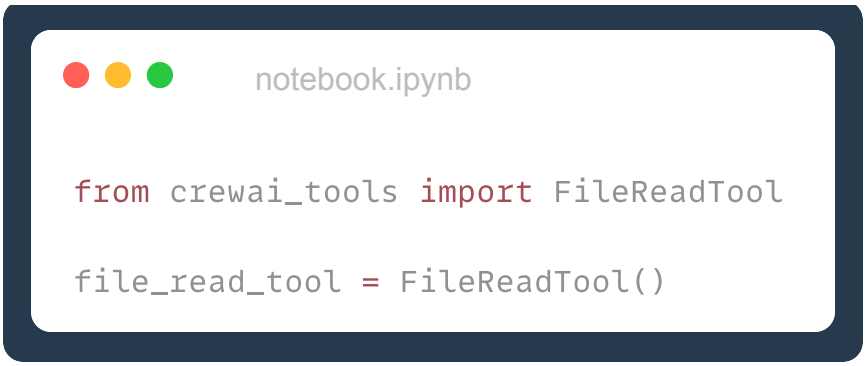
We'll cover some advanced tools like internet search shortly, but for now, let's look at how to use the file read tool.

In this case, we shall ask the Agent to read the context of the file and summarize it.

As a document, let's take the output we obtained above and store it in a markdown file:



To begin, we define a file reader tool as follows:



Next, we define our Agent that will be responsible for reading and summarizing files:

notebook.ipynb

```
from crewai import Agent

summarizer_agent = Agent(
    role="Senior Document Summarizer",
    goal="Extract and summarize key insights from provided files in 20 words or less.",
    backstory="""You are an expert in document analysis, skilled at extracting
                 key details, summarizing content, and identifying critical
                 insights from structured and unstructured text.""",
    tools=[file_read_tool],
    verbose=True
)
```

Specify tool for Agent

Now, we create a task that instructs the agent to read and analyze a file:

notebook.ipynb

```
from crewai import Task

summarizer_task = Task(
    description=(
        "Use the FileReadTool to read the contents of {file_path}"
        "and provide a summary in 20 words or less. "
        "Ensure the summary captures the key insights "
        "and main points from the document."
    ),
    agent=summarizer_agent,
    tools=[file_read_tool],
    expected_output="A concise 20-word summary of the key points from the file.",
)
```

Finally, we assemble a Crew that executes the file-processing workflow:

notebook.ipynb

```
from crewai import Crew

summarizer_crew = Crew(
    agents=[summarizer_agent],
    tasks=[summarizer_task],
    verbose=True
)

# Run the crew with the file input
result = summarizer_crew.kickoff(inputs={"file_path": "output.md"})
```

Printing the output, we get:

```
from IPython.display import Markdown

Markdown(result.raw)
```

✓ 0.0s Python

AI agents autonomously perform tasks. They enhance industries by improving automation, personalization, but face ethical and technical challenges.

Perfect, isn't it?

Building multi-agent systems

So far, we've explored how single-agent systems work in CrewAI—where one AI agent is assigned a task, executes it, and produces an output.

While this works well for simple use cases, it’s not scalable or robust for complex workflows, because:

- A single agent trying to do everything results in diluted focus and lower accuracy, which we also discussed earlier in the Building blocks section.
- A single agent cannot delegate tasks or refine outputs based on expert knowledge.
- A single agent executing multiple steps sequentially can be slow and inefficient.

To solve this, we build multi-agent systems where agents specialize, collaborate, and dynamically execute workflows.

Let's look at a demo of this.

Imagine you want to create an AI-powered research assistant that can:

- Search the internet for the latest information on a given topic.
- Summarize findings into a structured, easy-to-read format.
- Refine and verify the information before delivering the final output.

Instead of using a single agent, we’ll build a multi-agent system with:

- A Research Agent – Uses an internet search tool to find relevant information.
- A Summarization Agent – Processes and condenses key insights.
- A Fact-Checking Agent – Verifies accuracy and removes misleading information.

As you can tell, together, they create a robust and scalable research workflow.

Setting up the Agents

As discussed above, we define three specialized AI agents, each with a distinct role.

First and foremost, to enable real-time information retrieval, we integrate the Serper Dev Tool, which allows agents to search the web dynamically.

You also need to get an API key from here: https://serper.dev/ (it's free) and set in your `.env` file:

.env

```
OPENAI_API_KEY="sk-38381...."
SERPER_API_KEY="96f29..."
```

Once you have done that, load it in your environment as follows:

notebook.ipynb

```
import os
from dotenv import load_dotenv

load_dotenv()
```

Next, we define our Serper Dev tool:

notebook.ipynb

```
from crewai_tools import SerperDevTool

serper_dev_tool = SerperDevTool()
```

We start by defining the Research Agent, which will be responsible for finding the latest insights on a given topic using the Serper Dev Tool.

notebook.ipynb

```
from crewai import Agent, Task

research_agent = Agent(
    role="Internet Researcher",
    goal="Find the most relevant and recent information about a given topic.",
    backstory="""You are a skilled researcher, adept at navigating the internet
                 and gathering high-quality, reliable information.""",
    tools=[serper_dev_tool],
    verbose=True
)

research_task = Task(
    description="""Use the SerperDevTool to search for the
                   most relevant and recent data about {topic}."""
                "Extract the key insights from multiple sources.",
    agent=research_agent,
    tools=[serper_dev_tool],
    expected_output="A detailed research report with key insights and source references."
)
```

In the above code:

- We define an AI agent whose goal is to search the internet for relevant information.
- We equip it with `SerperDevTool`, enabling real-time web search.
- We assign the research task, ensuring that it extracts key insights from multiple sources.

Once the Research Agent has gathered the information, the Summarization Agent will take over. This agent is responsible for condensing the research into a concise and structured summary.

notebook.ipynb

```
summarizer_agent = Agent(
    role="Content Summarizer",
    goal="Condense the key insights from research into a short and informative summary.",
    backstory="""You are an expert in distilling complex information into concise,
                 easy-to-read summaries.""",
    verbose=True
)

summarization_task = Task(
    description="Summarize the research report into a concise and informative paragraph. "
                "Ensure clarity, coherence, and completeness.",
    agent=summarizer_agent,
    expected_output="A well-structured summary with the most important insights."
)
```

- The Summarization Agent ensures that the research findings are structured, easy to read, and clear.

Finally, to prevent misinformation, we introduce a Fact-Checking Agent that will cross-check all summarized information with credible sources:

notebook.ipynb

```
fact_checker_agent = Agent(
    role="Fact-Checking Specialist",
    goal="Verify the accuracy of information and remove any misleading or false claims.",
    backstory="""You are an investigative journalist with a knack for validating facts,
                 ensuring that only accurate information is published.""",
    tools=[serper_dev_tool],
    verbose=True
)

fact_checking_task = Task(
    description="Verify the summarized information for accuracy using the SerperDevTool. "
                "Cross-check facts with reliable sources and correct any errors.",
    agent=fact_checker_agent,
    tools=[serper_dev_tool],
    expected_output="A fact-checked, verified summary of the research topic."
)
```

- The Fact-Checking Agent is responsible for validating the summarized information.
- The Serper Dev Tool is used again to cross-check facts with external sources.

Now that we have three specialized agents and their respective tasks, we organize them into a Crew and execute the workflow sequentially.

notebook.ipynb

```
from crewai import Crew, Process

research_crew = Crew(
    agents=[research_agent, summarizer_agent, fact_checker_agent],
    tasks=[research_task, summarization_task, fact_checking_task],
    process=Process.sequential,
    verbose=True
)

result = research_crew.kickoff(inputs={"topic": "The impact of AI on job markets"})
print("\nFinal Verified Summary:\n", result)
```

Ensures tasks run in order: Research → Summarization → Fact-Check

In the above code:

- All three agents are grouped into a Crew, each assigned their specific task.
- Tasks are executed sequentially in a structured workflow:
  Research → Summarization → Fact-Checking.
- The topic is dynamically provided at runtime, making the workflow flexible for any research topic.

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 i


# Agent: Internet Researcher
## Thought: I need to gather recent and relevant information about the impact of AI on job markets. This requires a comprehensive searc
## 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 understar
---
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 m
---
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 wh

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 ove
---
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:
The research report on the impact of AI on job markets highlights significant transformations that are expected to result from AI advar

Generative AI, in particular, is anticipated to enhance U.S. labor productivity, driving a GDP increase of approximately 7% (almost $7

Overall, the displacement of jobs is significant, but there is recognition of the dual effect of AI in job creation and productivity er
```

We can also print the final output as follows:

```
Markdown(result.raw)
```

✓ 0.0s Python

The research report on the impact of AI on job markets highlights significant transformations that are expected to result from AI advancements. AI technologies may displace an estimated 300 million jobs globally, as reported by Goldman Sachs, which predicts a potential impact on job markets. However, this shift does not merely result in job elimination; it also leads to the creation of new roles that blend AI capabilities with human skill sets, thereby reshaping job markets.

Generative AI, in particular, is anticipated to enhance U.S. labor productivity, driving a GDP increase of approximately 7% (almost $7 trillion) by 2030, which is more substantial than earlier estimates of 0.5% to 0.9% annually. This increase is enabled by improved workflow optimization. Various sectors, including technology, healthcare, and hospitality, are experiencing profound changes as AI fills labor gaps while also raising concerns over traditional job losses.

Overall, the displacement of jobs is significant, but there is recognition of the dual effect of AI in job creation and productivity enhancement. To navigate these changes effectively, policymakers are encouraged to implement educational and reskilling initiatives to ensure equitable access to the economic advantages that AI technologies offer. A strategic approach is necessary to address these challenges and capitalize on the opportunities presented by AI advancements.

Done!

Using YAML config

So far, we've defined agents, tasks, and workflows directly in Python, which 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 can decouple agent definitions, tasks, and workflows from the Python script by using YAML configuration files. This approach allows 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.

Let's look at how to do this below:

First, we create a `config.yaml` file.

config.yaml

```yaml
agents:
  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."

  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."

  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."

tasks:
  research_task:
    description: "Use the SerperDevTool to find the most relevant and recent data on {topic}."
    assigned_agent: "research_agent"
    expected_output: "A detailed research report with key insights and source references."

  summarization_task:
    description: "Summarize the research findings into a well-structured, concise report."
    assigned_agent: "summarization_agent"
    expected_output: "A summary highlighting the key takeaways from the research."
    depends_on: "research_task"

  fact_checking_task:
    description: "Cross-check the summarized information for accuracy and remove any misleading claims."
    assigned_agent: "fact_checker_agent"
    expected_output: "A fact-checked and verified research summary."
    depends_on: "summarization_task"
```

The beauty of doing this is that now you can 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.

To do this, we first load the YAML file as follows:

config.yaml

```
import yaml

with open("config.yaml", "r") as file:
    config = yaml.safe_load(file)
```

Printing the `config` dictionary, we get:

```
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 can access the details of any agent just like we would do in any other dictionary:

config.yaml

```
>>> config["agents"]["fact_checker_agent"]["goal"]

"Verify research findings and ensure factual accuracy."
```

Now, I hope you understand where we are going with this.

More specifically, we can now parameterize 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 can do to research agent:

config.yaml

```
from crewai import Agent, Task

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

research_agent = Agent(
    role=config["agents"]["research_agent"]["role"],
    goal=config["agents"]["research_agent"]["goal"],
    backstory=config["agents"]["research_agent"]["backstory"],
    tools=[serper_dev_tool],
    verbose=True
)

research_task = Task(
    description=config["tasks"]["research_task"]["description"],
    agent=research_agent,
    tools=[serper_dev_tool]
    expected_output=config["tasks"]["research_task"]["expected_output"]
)
```

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

config.yaml

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

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

summarization_task = Task(
    description=config["tasks"]["summarization_task"]["description"],
    agent=summarization_agent,
    expected_output=config["tasks"]["summarization_task"]["expected_output"],
)

fact_checking_task = Task(
    description=config["tasks"]["fact_checking_task"]["description"],
    agent=fact_checker_agent,
    tools=[serper_dev_tool]
    expected_output=config["tasks"]["fact_checking_task"]["expected_output"],
)
```

Finally, we have our Crew like before:

notebook.ipynb

```
from crewai import Crew, Process

research_crew = Crew(
    agents=[research_agent, summarizer_agent, fact_checker_agent],
    tasks=[research_task, summarization_task, fact_checking_task],
    process=Process.sequential,
    verbose=True
)

result = research_crew.kickoff(inputs={"topic": "The impact of AI on job markets"})
print("\nFinal Verified Summary:\n", result)
```

Ensures tasks run in order: Research → Summarization → Fact-Check

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?

There's one more way to make this even simpler, but we think it would be better if we discussed it in the upcoming part.

Conclusion and key takeaways

Before we wrap up, here are a few important considerations when working with AI agents:

1) Don't build AI Agents if you don’t have to:

While AI agents can be powerful, not every problem requires an agentic solution. If a simple function or script can get the job done more efficiently, stick to that.

Overcomplicating workflows with unnecessary agents can lead to:

- Increased execution time due to multiple agents interacting.
- Higher compute costs, especially when calling LLMs multiple times.
- More debugging and maintenance overhead for complex agent coordination.

As a rule of thumb → If a single prompt or API call suffices, don't use an AI agent.

2) Prompting, role definitions, etc., are critical.

The best way to learn AI agents is by experimenting with them in different ways. Try:

- Modifying agent roles and goals to see how they behave differently.
- Creating Agents in CrewAI is pretty standard. As long as your code is fine, any changes will have little influence on how well your Agent will perform. Thus, mostly the only two or three controlling factors are the details you provide in descriptions like roles, goals, etc, and the tools you give access to.

The more you experiment and write better descriptions, the better you'll understand their capabilities and limitations.

Whenever we build agents, we often give as much detail as possible, as you can see below:

```python
from crewai import Agent

# Second Agent: Content Writer
content_writer = Agent(
    role="Content Writer",
    goal="Transform research findings into engaging blog posts while maintaining accuracy",
    backstory="You're a skilled content writer specialized in creating "
              "engaging, accessible content from technical research. "
              "You work closely with the Senior Research Analyst and excel at maintaining \n"
              "the perfect balance between informative and entertaining writing, "
              "while ensuring all facts and citations from the research "
              "are properly incorporated. You have a talent for making "
              "complex topics approachable without oversimplifying them.",
    allow_delegation=False,
    verbose=True,
    llm=llm
)
```

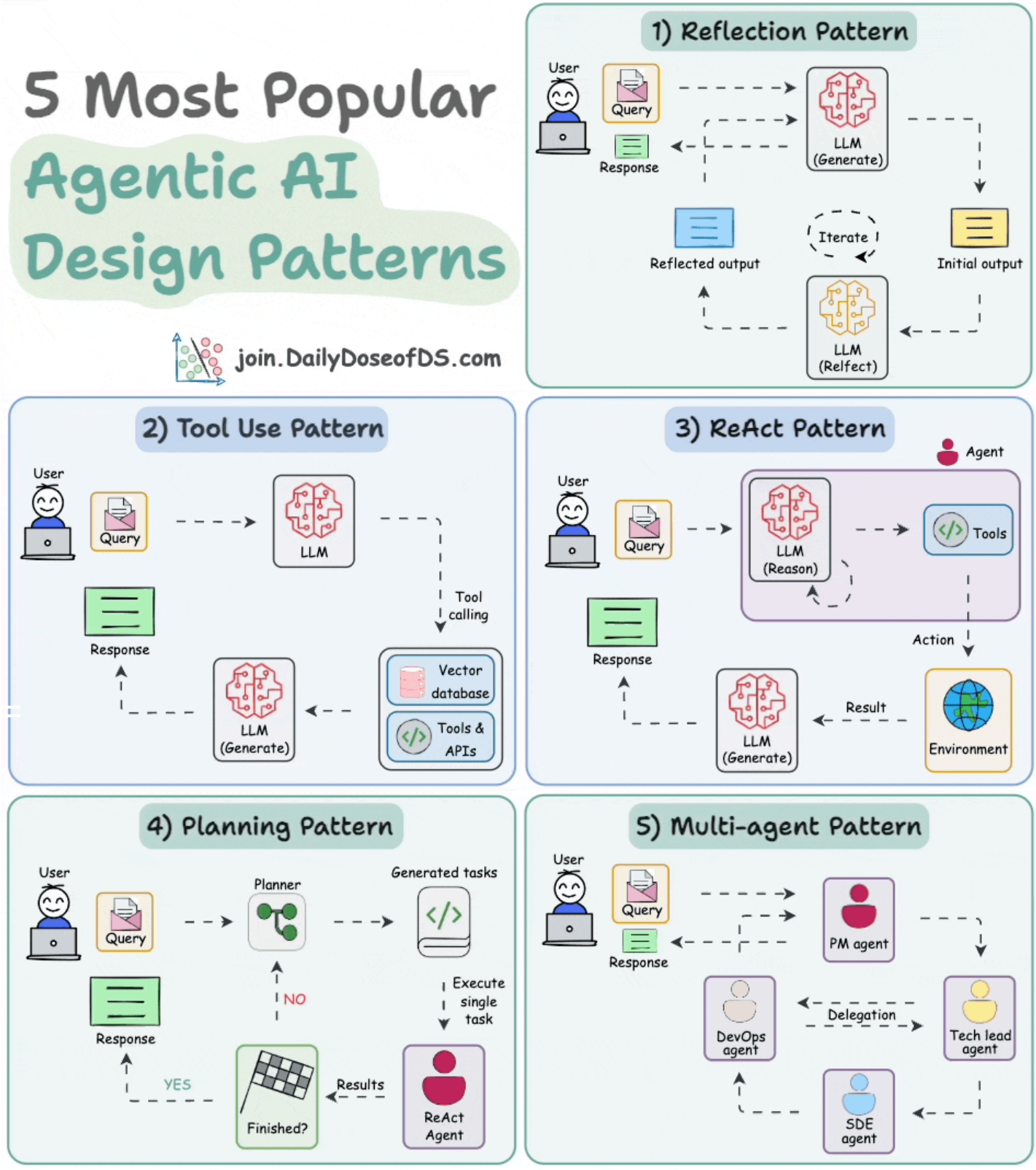

```python
from crewai import Task

# Writing Task
writing_task = Task(
    description=("""
        Using the research brief provided, create an engaging blog post that:
        1. Transforms technical information into accessible content
        2. Maintains all factual accuracy and citations from the research
        3. Includes:
            - Attention-grabbing introduction
            - Well-structured body sections with clear headings
            - Compelling conclusion
        4. Preserves all source citations in [Source: URL] format
        5. Includes a References section at the end
    """),
    expected_output="""A polished blog post in markdown format that:
        - Engages readers while maintaining accuracy
        - Contains properly structured sections
        - Includes Inline citations hyperlinked to the original source url
        - Presents information in an accessible yet informative way
        - Follows proper markdown formatting, use H1 for the title and H3 for the sub-sections""",
    agent=content_writer
)
```

So far, we've only scratched the surface!

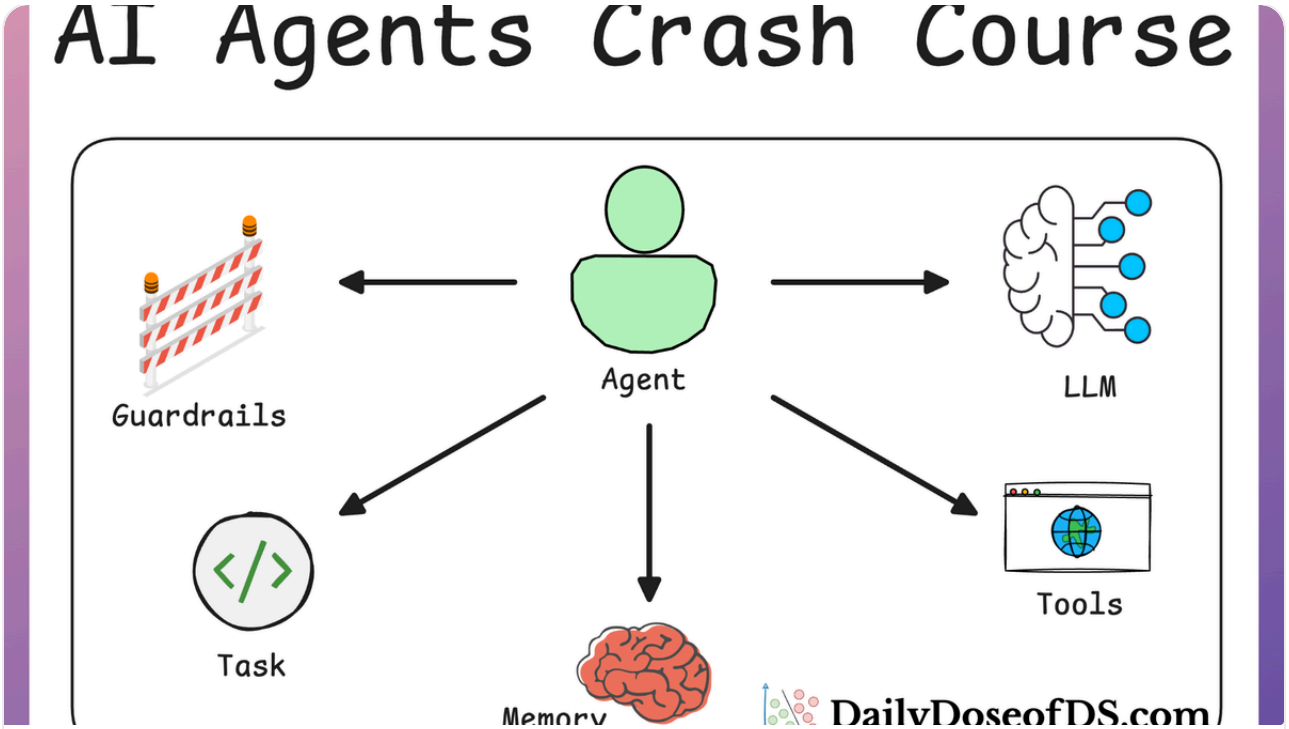
In the upcoming parts, we have several other advanced agentic things planned for you:

- 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.
- Using Flows for better multi-agent coordination.
- Generating structured outputs instead of plain text responses.
- Optimizing agents for real-world applications in business and automation.
- Building Agents around the Agentic patterns depicted below:



- and many many more.

Read the next part of this crash course here:



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

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

Daily Dose of Data Science • Avi Chawla

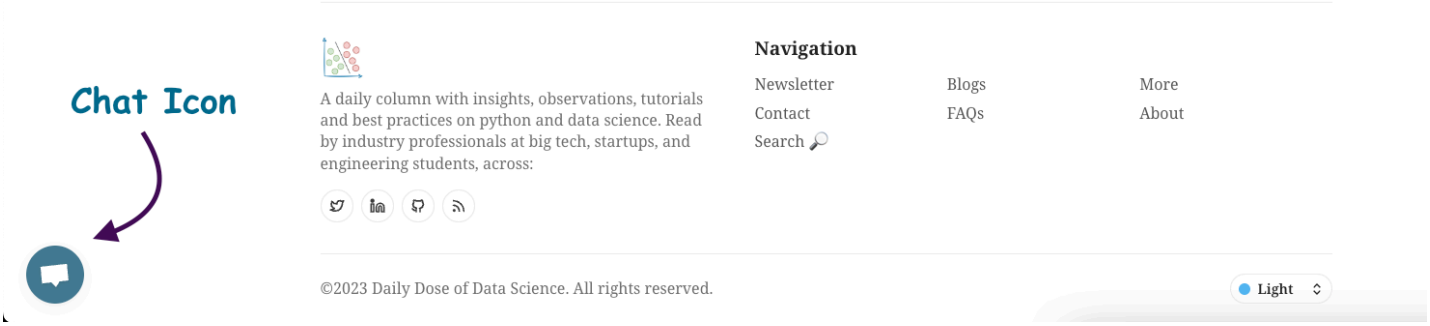
As always, thanks for reading!

Any questions?

Feel free to post them in the comments.

Or

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



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

Navigation