Crewai: Combining the power of AI and Teamwork in developing lesson plan for a classroom lecture.

1. **Introduction**

In the ever-evolving landscape of education, where the convergence of technology and pedagogy drives innovation, a remarkable tool is emerging to revolutionize the way classroom lectures are developed and delivered. CrewAI, a cutting-edge framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling complex tasks. In this article, we delve into the transformative potential of CrewAI in shaping the creation process of classroom lectures, illuminating how this fusion of AI and teamwork promises to enhance educational outcomes and foster a dynamic learning environment.

# **When prompting is not enough**

Prompting alone is not sufficient for AI chat systems like ChatGPT or other AI completion models because it only provides a starting point or context for generating responses. While prompts help guide the AI in understanding the user's intent or the task at hand, they do not capture the full complexity of human language or conversation. AI models require extensive training on diverse datasets to develop a deep understanding of language patterns, semantics, and context.

Good prompting brought results that [seemed impressive](https://www.cnet.com/tech/computing/why-were-all-obsessed-with-the-mind-blowing-chatgpt-ai-chatbot/) for a while. However, we quickly became accustomed to these outcomes, and they [no longer felt magical](https://www.wearedevelopers.com/magazine/chatgpt-getting-worse-over-time). They turned into the standard. Which isn’t bad on its own, but this standard wasn’t “good enough” for a customer base accustomed to rapid, significant technological advancements. Additionally, users encountered limitations such as hallucinations [1] and [generic-sounding text](https://www.wordrake.com/blog/wordy-choppy-generative-ai) [2].

# **AI agent**

An agent, in the context of computer science and AI, is a software entity programmed to perceive its environment (like visual, auditory, and physical inputs) and take actions to achieve specific goals. AI agents utilize algorithms and data to autonomously make decisions, playing roles such as virtual assistants, autonomous vehicles, or game-playing agents, among others. They are fundamental to AI systems, enabling machines to interact intelligently with the world.

An agent is simply an entity that can act. The idea has been around for thousands of years [in some form](https://academic.oup.com/edited-volume/42642/chapter/358147424), but it has gained traction with the rise of AI and in particular self-driving cars.

A [recent research](https://arxiv.org/pdf/2309.07864.pdf) emphasized the importance of “iterative thinking” that agents seem to provide. Based on the same research, agents can be placed into 3 categories based on capabilities:

* Task-Oriented: at this level, agents are capable of automating boring, repetitive tasks like: scrape data -> extract key points -> summarize.
* Innovation-Oriented: Agents that are trained on medical or scientific data might be capable of scientific breakthroughs, and there's proof [that this is already happening](https://www.vox.com/future-perfect/23827785/artifical-intelligence-ai-drug-discovery-medicine-pharmaceutical).
* AGI or Lifecycle-Oriented: General, super intelligence, capable of planning [might emerge from a team of agents](https://amatriain.net/blog/multiagents#:~:text=Contrary%20to%20traditional%20visions%20of,a%20network%20of%20specialized%20agents.) instead of one super big LLM.

# Agent’s critical components

An LLM-based agent interacts with its environment by sensing data and taking actions, sometimes utilizing tools. In unimodal mode, the agent's input and output are purely text-based. In multi-modal mode, the agent's perception expands to include visual, auditory, and physical information, allowing it to perform actions within the environment itself.

## LLMs (the Brains)

Inside an autonomous agent, LLMs handle diverse tasks like planning, reasoning, taking actions, evaluating results, and generating summaries. To accomplish this, they use their understanding of language, general knowledge, and the ability to extract actionable steps from information. Sometimes, multiple LLMs work together, each specializing in different functions – similar to how different areas of the brain have specialized purposes. This approach improves efficiency and reduces costs.

**Input:** The LLM processes text-based input. This includes:

* **User Commands:** Specific instructions from the person using the agent.
* **System Guidelines:** Broad rules and preferences set for the agent's operation.
* **Examples:** Demonstrations of the desired output (this helps the LLM learn).
* **Conversation History:** If interacting over multiple turns, the LLM uses past exchanges to provide better responses.
* **Records of Past Actions:** When complex tasks are broken into steps, the LLM keeps track for better overall performance.

**Output:** The LLM also generates text-based output. This includes:

* **Solutions:** Direct answers to the user's commands.
* **Action Plans:** Step-by-step instructions for more complex tasks.
* **Structured Data:** If instructed, the LLM can provide outputs in formats like JSON to easily feed into other programs.

## Memory

In ongoing conversations, it's important for the LLM to "remember" what was said earlier. This means storing past dialogue in memory. For complex tasks broken into smaller steps, the LLM also needs to track plans, actions, and results from those steps to stay on course. However, LLMs have limits on how much information they can process at once, and storing everything costs time and money. To balance these needs, different memory strategies have been developed.

## Tools or Data resources

**Tools**: LLMs can discern the suitable API (search engines for example) to use and correct input arguments, thanks to their context learning capabilities.

**Data**: To use external data or scripts effectively with LLMs, here's a simplified breakdown of the process:

1. Segmentation: Long texts are broken down into smaller chunks to fit the LLMs input limits.
2. Embedding: Text segments are converted into numerical representations that the LLM can understand.
3. Indexing: These numerical representations are organized for quick and easy retrieval.
4. Retrieval: When needed, relevant information is pulled from a database or other storage system.

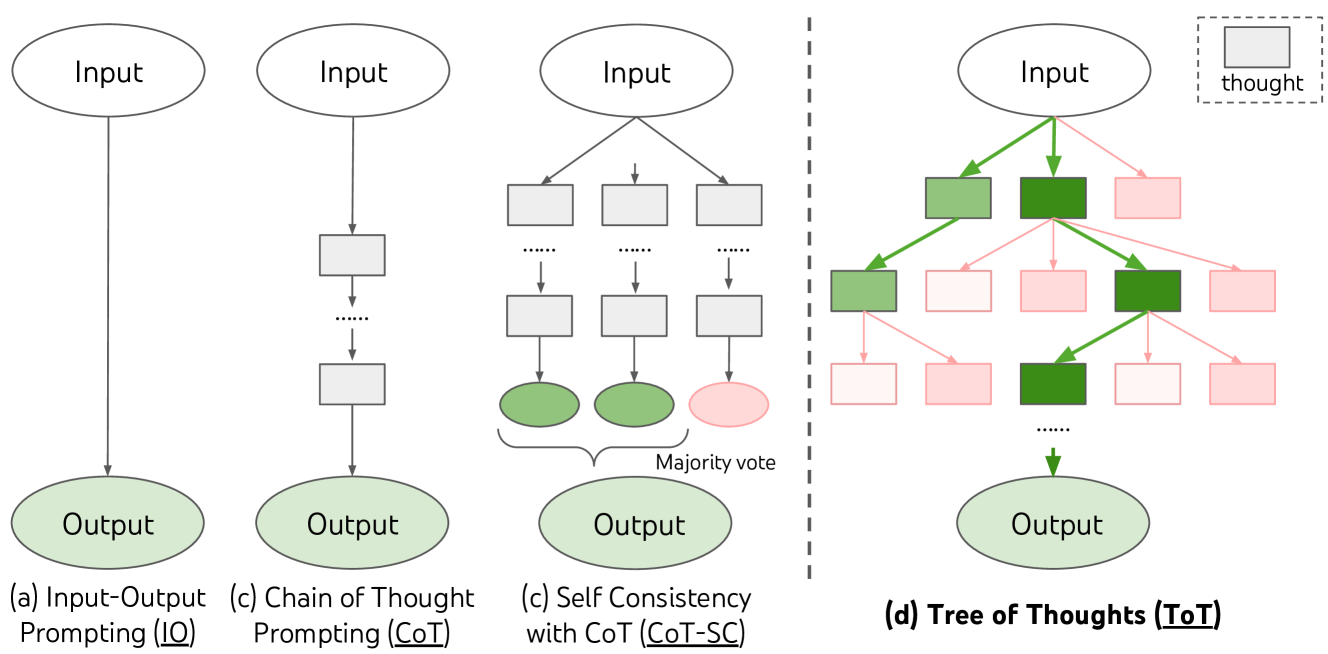
# LLMs-Based Agents

LLMs-Based Agents have evolved from simple input-output systems to complex ones powered by LLMs:

* Early Agents: Accept direct commands and provide single output (Direct Prompting, Instruction Prompting).
* Advanced agents: Can now break down tasks into steps, use logic, access external tools and information, evaluate their own progress, and change course if needed (Chain of Thought/Tree of Thought).

## Reasonable Agents:

There have recently been various attempts to make LLMs “think” reasonably like Chain of Thought (CoT) or even more advanced like Tree of Thought (ToT). This prompting technique guides the LLMs to break down complicated tasks into multiple steps, tackle individual steps sequentially, and output a final answer. However, The instructions within a prompt guide the LLM's problem-solving approach. Different strategies can be employed, each with its own strengths and weaknesses.



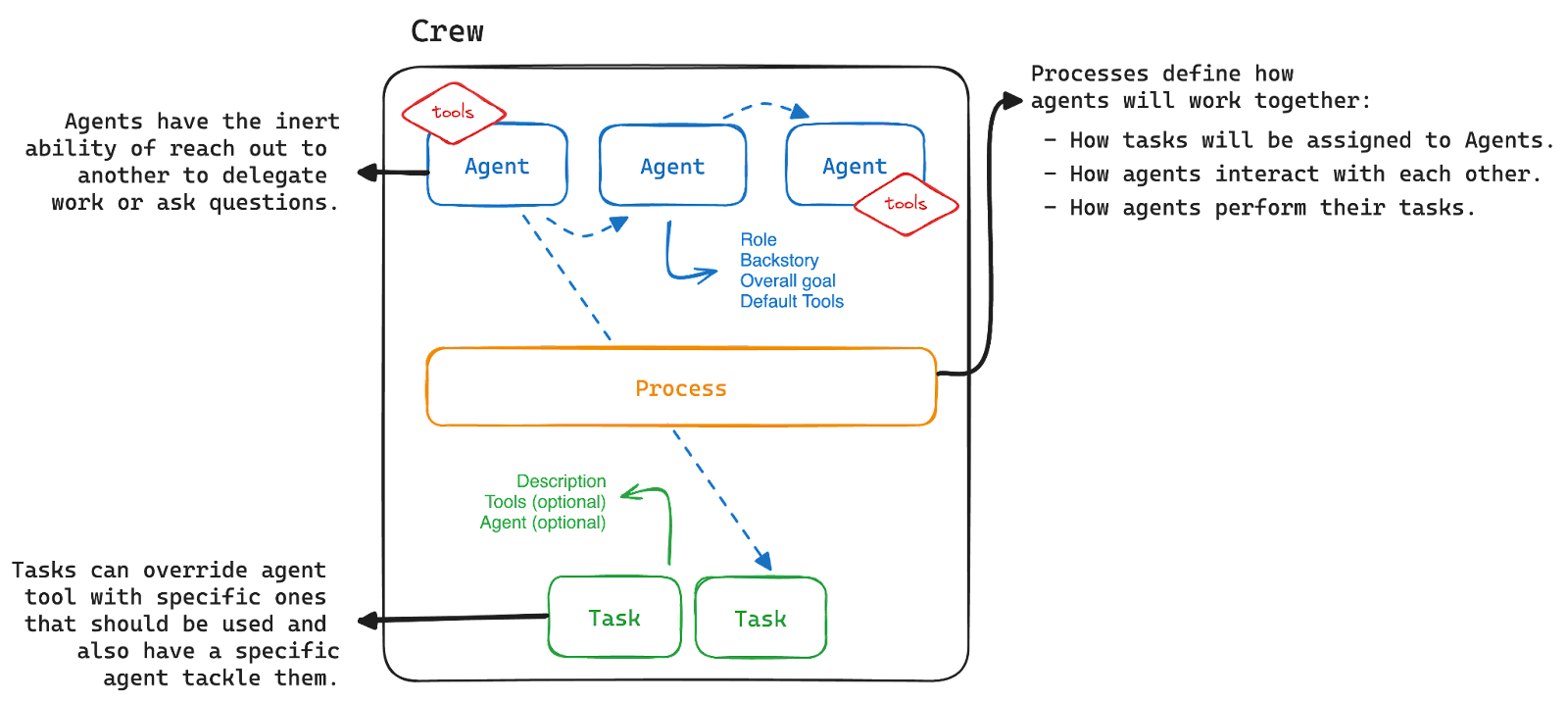
Source: [Yao et el. (2023)](https://arxiv.org/abs/2305.10601)

# The need of multi-agent solutions:

Leveraging the impressive reasoning capabilities of LLM-based agents highlights the necessity for multi-agent solutions, a sentiment echoed in research exploring collaborative AI systems. While individual agents demonstrate proficiency in processing extensive textual data and producing coherent responses, their effectiveness can be notably augmented through collaboration within a multi-agent framework. Multi-agent systems facilitate synergistic interactions among individuals, each possessing unique expertise or specialized knowledge domains. By harnessing their collective reasoning power, these agents can address intricate tasks that surpass the capabilities of any single agent. This collaborative approach enables the inclusion of specialists in various domains, such as experts in real-time information retrieval, external tool manipulation, and intricate action planning, [CrewAI](https://blog.langchain.dev/crewai-unleashed-future-of-ai-agent-teams/) (from Joao Moura) is arguably the remarkable solution that implemented a multi-agent framework built on Langchain.

# CrewAI basics

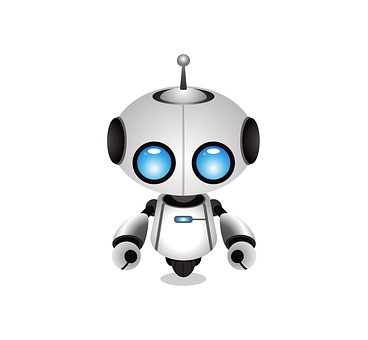
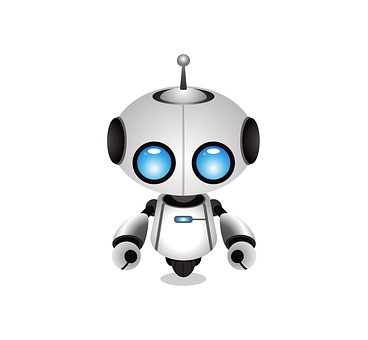
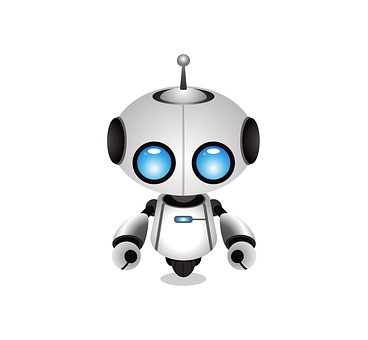
CrewAI is a framework that allows us to create AI agents where each agent has its own expertise and instructions. Together, the agents will work towards solving one large, complex problem.



Source: [CrewAI Unleashed: Future of AI Agent Teams](https://blog.langchain.dev/crewai-unleashed-future-of-ai-agent-teams/)

## CrewAI’s components and Concepts:

* **Agents**: These are like dedicated team members, each with their role. Agent is programmed to (1) perform tasks (2) make decisions and (3) communicate with other agents.



Agent

(Manager)

Agent

(Researcher)

Agent

(Writer)

Information

Task

Information

Task

Information

Task

Agent most important attributes

|  |  |
| --- | --- |
| Role | Define the agent’s function in the crew, the kind of tasks that the agent is best suited for |
| Goal | The objective that the agent aims to achieve. This will help the agent to make a decision. |
| Backstory | Provide context to the agent role and goal, fostering the interaction and collaboration |
| Tools | Functions that the agent can use to perform tasks |
| llm | The language model used by this agent to call functions, if none is passed the same main llm for each agent will be used. |
| Verbose | Allow to see what is going on during execution of the crew |

* **Tasks**: Jobs that Agents need to complete. It is designed to require collaboration between agents. For example, one agent might gather data while another analyzes it. This collaborative approach can be defined within the task properties and managed by the Crew's process.

Task attributes

|  |  |
| --- | --- |
| Description | A clear and concise statement of agent’s task |
| Agent | Specify which agent is responsible for the task |
| Backstory | Provide context to the agent role and goal, fostering the interaction and collaboration |
| Tools | Functions that the agent can use to perform tasks |
| Async Execution | (Optional) If the task is executed asynchronously |
| Expected\_ output | (Optional) Clear and detailed definition of expected output for the task |

* **Tools**: The equipment our agents use to perform their tasks efficiently, you can use existing ones from LangChain or develop your own (search the internet, scrape the web or do some calculations, . . .)

The following code creates user-defined tools usearch\_tool based on google serper, the results will then feed to uscrap\_tool to scrape and summarize.

List of tools then created with load\_tools and append method.

usearch\_tool = Tool(

name='my\_search',

func= SearchTools.google\_search,

description="This tool uses google serper as the search engine."

)

uscrap\_tool = Tool(

name = "scraping",

func = BrowserTools.scrape\_and\_summarize,

description= "Scrap web and extract text with scrapingant, beautiful soup then summarize."

)

tools = load\_tools([ ])

tools.append(usearch\_tool)

tools.append(uscrap\_tool)

* **Process**: The workflow or strategy that the crew follows to complete tasks. It has two implementations:
  + Sequential: Tasks are executed one after another, ensuring a linear and orderly progression and the output of one task is automatically used as context into the next task.
  + Hierarchical: Tasks don't need to be pre-assigned to agents, the manager will decide which agent perform each task, review the output and decide if the task is completed or not
* **Crew**: Where Agents, Tasks and a Process meet, this is the container layer where the work happens.

Most import attributes:

|  |  |
| --- | --- |
| Tasks | List of task assigned to the crew |
| Agents | List of agents that are part of the crew |
| Process | Process flow, can be sequential or hierarchical |
| Tools | Functions that the agent can use to perform tasks |
| Manager llm | The language model used by the manager agent in a hierarchical process. |
| Verbose | Verbosity level logging during execution |
| Full output | Whether the crew returns all tasks outputs or just the final output. |

## To set up a Crew using CrewAI, you need to follow these steps:

## Define Your Agents: Start by creating agents that will be part of your crew. Each agent should have a defined role, goal, backstory, and optionally, tools.

## Create Tasks: Define tasks that you want your agents to perform. These tasks are associated with specific agents and should clearly describe what is expected.

## Instantiate the Crew: Create a crew by providing a list of agents and tasks. You also need to specify the process flow (e.g., sequential), and other optional configurations such as verbosity, a unique ID, and maximum requests per minute (RPM).

## Kickoff the Crew: Once the crew is set up, you can start it with the kickoff() method. This will initiate the execution of the tasks by the agents according to the defined process.

## The code

# Code to define a teacher agent

topic = “HTTP protocol, header dissection and investigation with scapy code”

teacher = Agent(

role = ”Senior Lecturer”,

goal=f”Design lesson plan on {topic} for a class of diverse knowledge level students. “,

backstory="""

You're an experienced lecturer with expertise in creating comprehensive lesson plans about networking technologies, services, especially on TCP/IP protocol suite.

""",

tools=tools,

verbose=True

)

# Code to define task of the teacher agent

lesson\_planning = Task (

description="Write draft of the lesson plan for a 2-hour lecture, write 1000-2000 words",

agent=teacher)

# Code instantiates the crew

classroom\_crew = Crew (

agents = [teacher, advanced\_student1, advanced\_student2,

intermediate\_student1, intermediate\_student2,

Beginner\_student

],

Tasks = [lesson\_planning, question1, question2, question3,

question4, question5, finalize\_plan],

verbose=2,

process = Process.sequential,

)

# kickoff the crew

classroom\_result = classroom\_crew.kickoff()

## Designing a lesson plan with crewAI

To take advantage of the agent reasoning capability, I have created a crewAI of agents to simulate a classroom environment.

The crew will consist of the lecturer, a coding\_expert, 2 advanced-level students, 2 intermediate-level students, and 1 basic level student.

The lecturer will draft the lesson plan first, advanced-level students will raise questions based on that plan, in the same way for other students. Lastly, the lecturer will rewrite the lesson plan, incorporating ideas from questions of the students to make up the final plan.

[1] <https://arxiv.org/abs/2311.05232>

[2] [Why AI-Generated Text Sounds Wordy and Choppy (wordrake.com)](https://www.wordrake.com/blog/wordy-choppy-generative-ai)