GENERATIVE AI SERIES

# Multi-Agent System — Crew.AI

Multi-Agent systems are LLM applications that are changing the automation landscape with intelligent bots.

A B Vijay Kumar · Follow
7 min read · Apr 17, 2024



*This blog is an ongoing series on Generative AI and introduction to multi-agent architecture and frameworks such as Autogen, Crew.ai, that help build intelligent bots, that implement multi-agent architectures. In this blog, we will explore crew.ai*

## Multi Agent System

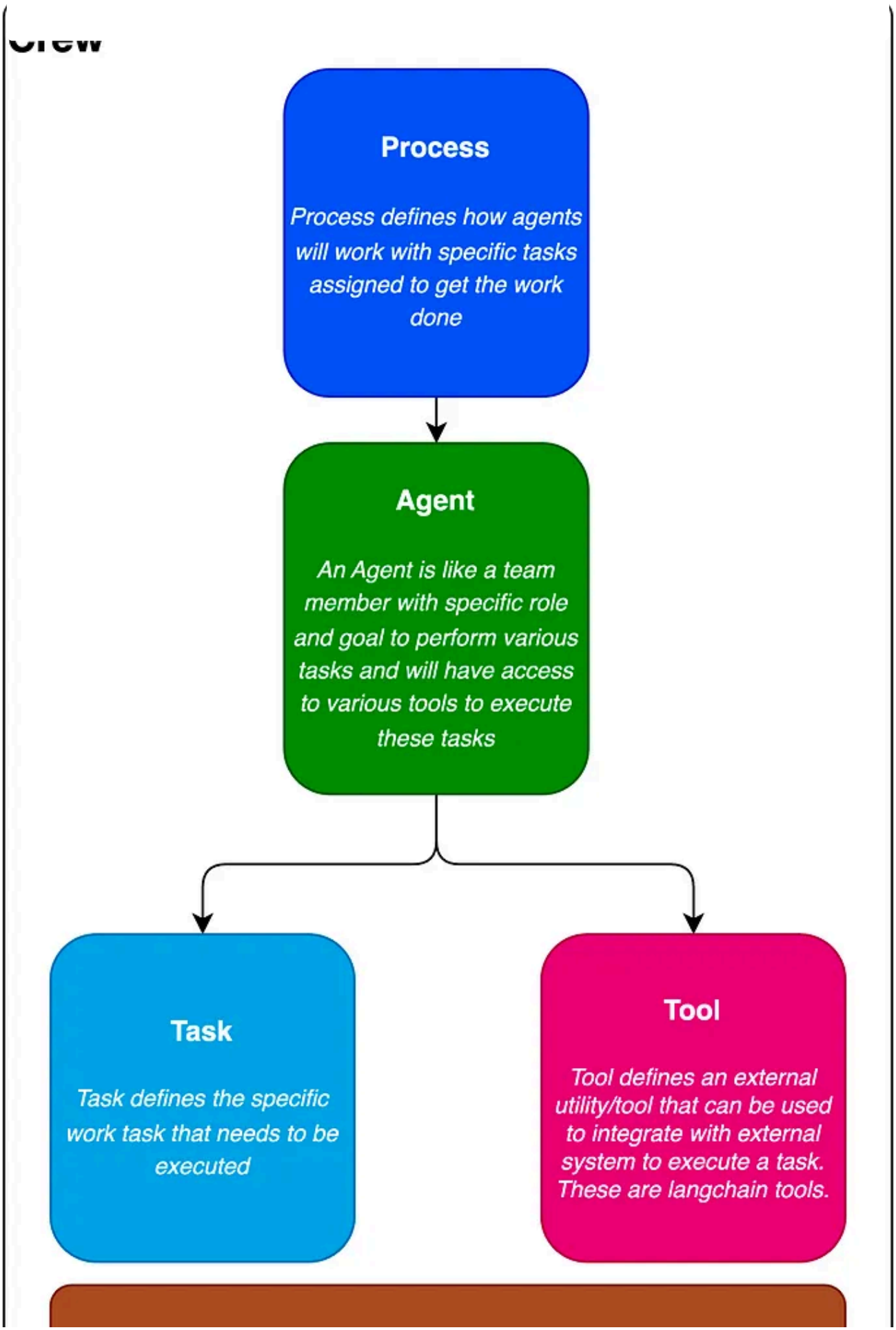
In the context of language models and AI, a **multi-agent system** involves multiple independent actors, each powered by language models, collaborating in a specific way. I did a blog on Multi-Agent Systems, and general architecture. Please read the following blog, before you proceed.

**Multi-Agent Architectures**

In this blog, we will dig deeper on crew.ai, one of the emerging frameworks for building multi-agent application. Crew.ai provides a framework for building agents to work together seamlessly, tackling complex tasks with a level of sophistication that mirrors the dynamics of a well-oiled team. Crew.AI is designed to enable AI agents to assume roles, share goals, and operate in a cohesive unit.

## Architecture

The architecture of CrewAI is modular, comprising several key components working together to achieve a well-orchestrated multi-agent system. The following picture shows the key components that provide a framework to build multi-agent LLM applications.


Process
Process defines how agents will work with specific tasks assigned to get the work done
Agent
An Agent is like a team member with specific role and goal to perform various tasks and will have access to various tools to execute these tasks
Task
Task defines the specific work task that needs to be executed
Tool
Tool defines an external utility/tool that can be used to integrate with external system to execute a task. These are langchain tools.

Let's go through this picture bottom up so that we understand how they come together.

**Tool**: A tool object is a utility/equipment that the agents use to perform a specific task efficiently. For example, searching the web, loading documents & reading them, etc. Crewai is built with LangChain, we can use any existing ones from LangChain or we can write our custom tools.

**Task**: Task, as the name suggests, is the specific task that needs to be executed. An Agent performs a task. We provide various tools that are required to execute a task.

**Agents**: An agent is like a team member in the crew, with a specific role, background story, goal, and memory. The agent is the core performer of the tasks assigned within the framework. Each CrewAI Agent is a LangChain Agent, but enhanced with a `ReActSingleInputOutputParser`. This parser is specially modified to better support role-playing, incorporates a binding stop word for contextual focus, and integrates a memory mechanism, using ConversationSummaryMemory for maintaining the context.

**Crew**: Think of the crew as a team of agents, working together to achieve a particular goal. These agents have a clearly defined way to collaborate to achieve the set of tasks at hand

**Process:** Process object is the workflow or strategy the crew follows to complete tasks. 3 strategies are defined by the framework (as of the time I was writing the blog. I understand that there are plans to add more).

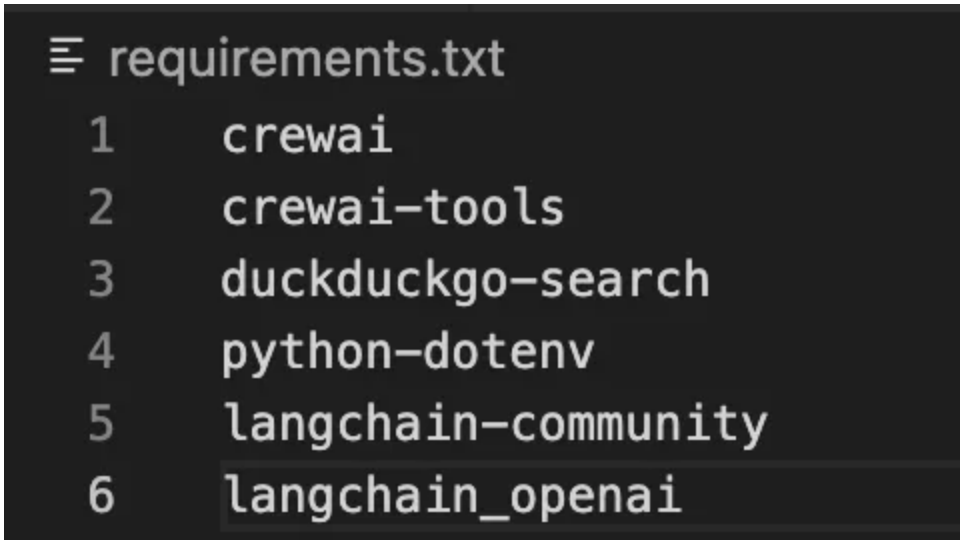
- **Sequential:** This strategy executes the given "tasks" in sequence, and in a particular defined order. This strategy is ideal for any pipeline type of work, where each agent performs a specific task and passes it to the next agent. We will use this strategy in our example today to write a blog, for a given topic.
- **Hierarchical:** This strategy organizes tasks in a hierarchy, and these tasks are delegated in a hierarchical fashion and executed based on chain of command. This is remarkably close to an orchestrator kind of pattern. This is like a manager assigning work to various agents, and validating the results, before completing the work. When we use this strategy, we need to configure a `manager_llm`, that helps take the right decision. We will be building a solution using this strategy in the next blog.
- **Consensual Process (Planned):** This is one of the most popular strategies, which is yet to be released, where we have agents talking to each other to get the work done. This is based on collaborative decision-making and the decisions are taken democratically. This is not yet released, but we have other multi-system frameworks, and have already implemented this strategy. We will explore this in future blogs

You can read more about these objects and API in crewai documentation.

Let's now jump into building our first multi-agent system that build a blog in markup format, for a given topic.

## Code

Before we start the work, let's install all the dependencies. find below my requirements.txt that has all the python dependencies. I run `pip install -r requirements.txt` (after creating my own virtual environment), to install all the dependencies.




To build out blogger multi-agent application, we will be implementing 2 agents

- `researcher` : Researcher agent will search the web for a given topic and collect all the information. We set the context of the llm, to think like a researcher, and get all the material that we need, on the topic.
- `blogger` : Blogger agent will convert the content that is collected by researcher into a blog.

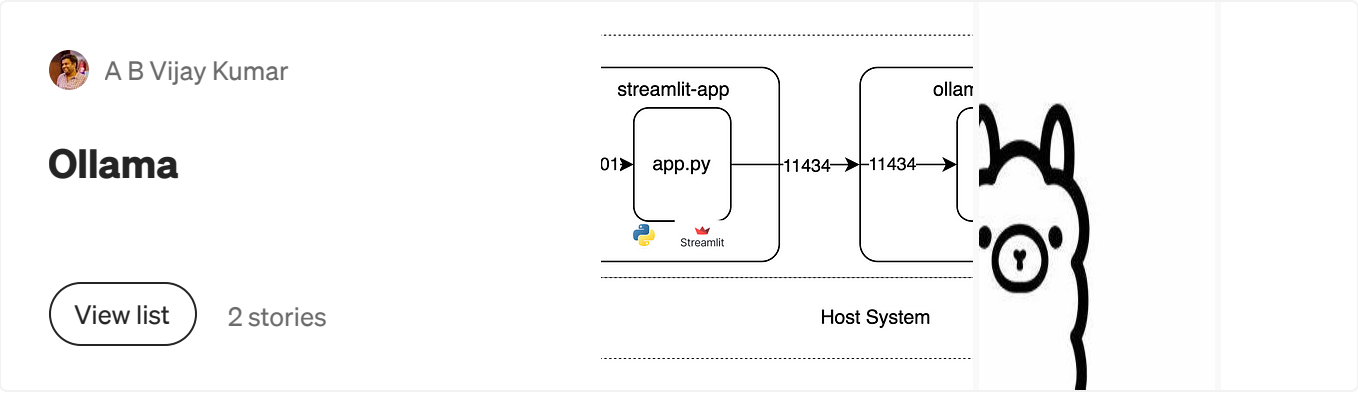
These agents will run the following tasks

- `task_search` : This task is about searching the web for all the relevant content for the given topic. We will be providing 2 tools ( `duckduckgo`

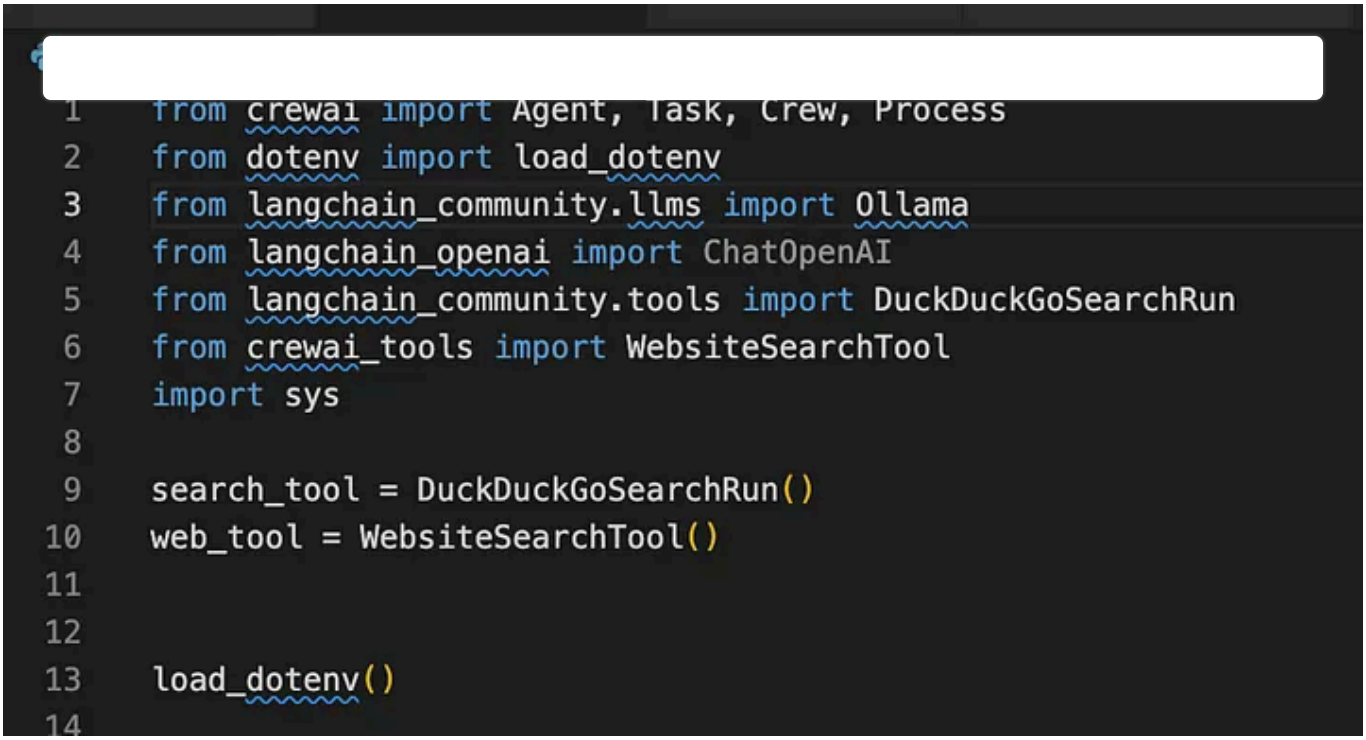
search tool to search using `duckduckgo` and web search tool that uses `serper`), to support the `task_search` agent to do complete research

- `task_post` : This task is about writing the blog, with the provided information in markdown format.

We will be running our models on Ollama on our laptops. To understand more about Ollama Please read my blog list

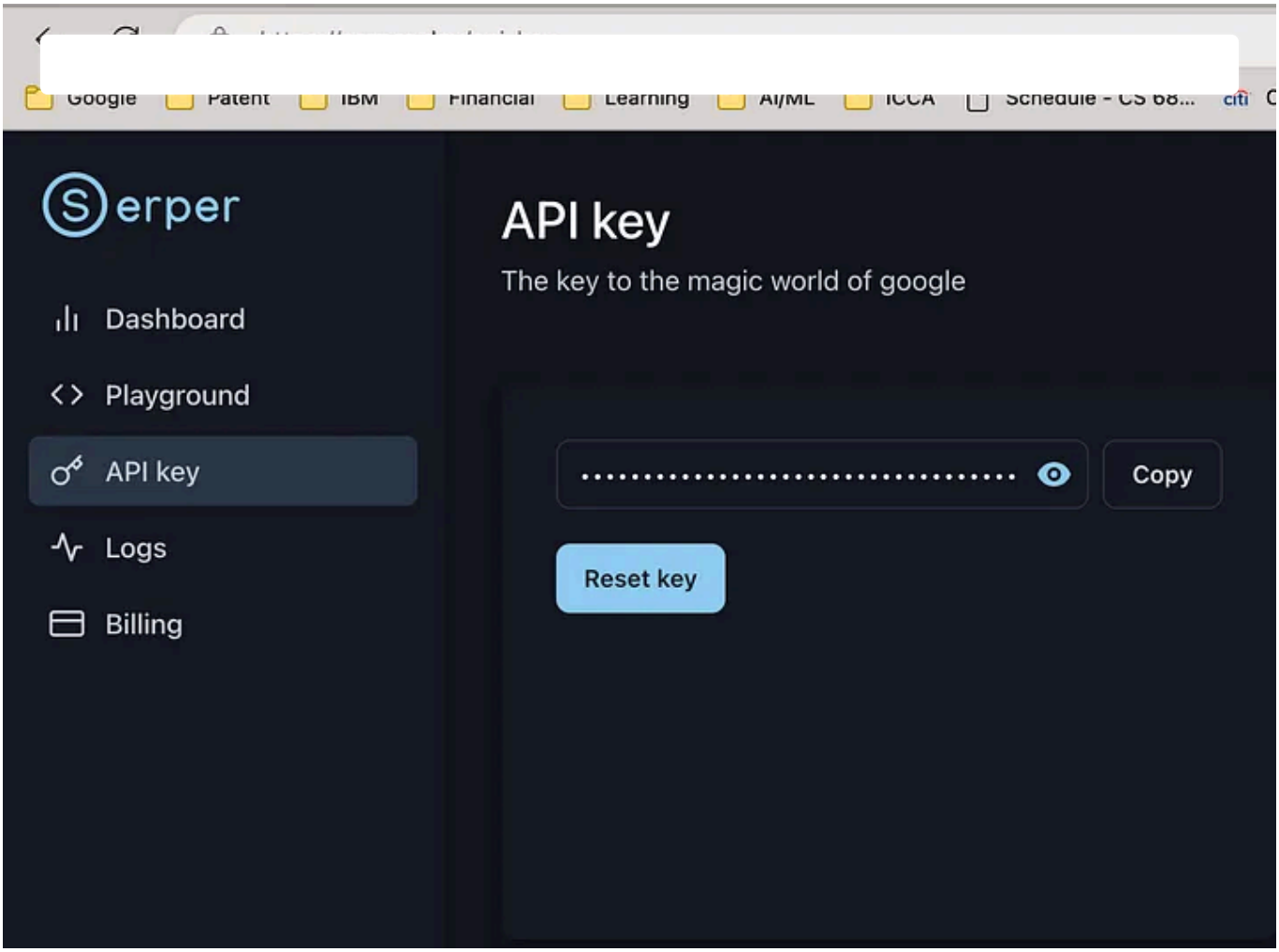



Lets now look at the code. The following code shows all the dependencies imports.

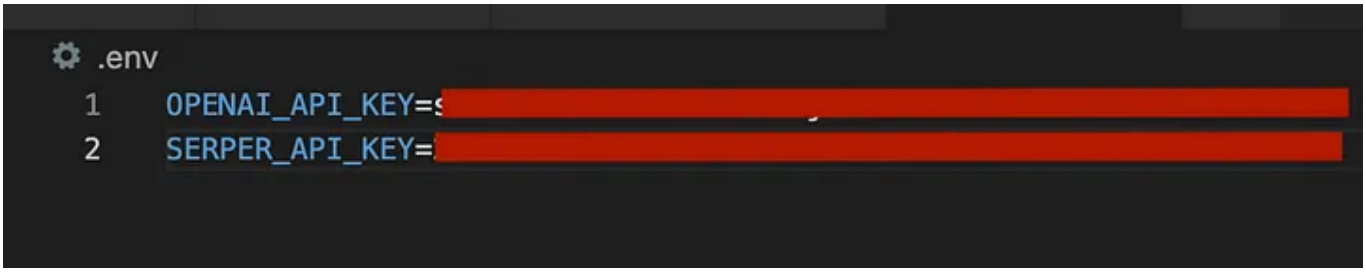


```
1  from crewai import Agent, Task, Crew, Process
2  from dotenv import load_dotenv
3  from langchain_community.llms import Ollama
4  from langchain_openai import ChatOpenAI
5  from langchain_community.tools import DuckDuckGoSearchRun
6  from crewai_tools import WebsiteSearchTool
7  import sys
8
9  search_tool = DuckDuckGoSearchRun()
10 web_tool = WebsiteSearchTool()
11
12
13 load_dotenv()
14
```

in line number 9, 10 we are defining the various tools that we will be using. To use webtool with Serper, you need to signup with server at Serper — The World's Fastest and Cheapest Google Search API, and generate a API key, and configure that in the env. The following screenshot shows how to access the API key




The following is my `.env` file, that we will be loading using `dotenv` library.




Please note that I am not using OpenAI, but I have anyway shown in the following code how we would use OpenAI (if you don't want to run your models on Ollama). In my case, I have installed `misral` model on Ollama, and using it locally. If you want, you can use OpenAI/GPT or any other LLMs.


In the following code we are defining the `kickoffTheCrew(topic)` method. which we will be calling from the main blog. We will be passing the `topic` parameter, to perform the research and generate a blog.

In this method we are instantiating the 2 agents Researcher, Blogger, that we defined earlier in the blog. As you can see, each agent we are defining a specific `role`, `goal` and setting the context using `backstory`. All these are really like prompts with context, and it is very important to provide the right prompts. I had to fine tune this several times to achieve the best results, I am sure we can do better results by fine tuning these 3 parameters. We are also passing the llm to be used, Please note that in our example we are using the same llm, but we could be using multiple different llms for different agents, if required, depending on what the agent is going to achieve. Its infact a good practice to use the best llm for the specific tasks the agent is going to perform, for best results.

```
⌘L to chat, ⌘K to generate

        role = "Internet Research",
        goal = f"Perform research on the {topic}, and find and explore about {topic} ",
        verbose = True,
        llm=llm,
        backstory = """You are an expert Internet Researcher
        Who knows how to search the internet for detailed content on {topic}
        Include any code examples with documentation"""
    )

    blogger = Agent(
        role='Blogger',
        goal="""Write engaging and interesting blog in maximum 2000 words. Add relevant emojis""",
        verbose=True,
        allow_delegation=True,
        llm=llm,
        backstory="""You are an Expert Blogger on Internet.
                    Include code examples, and provide tutorial type instructions for the readers."""
    )
```

In the following code we are defining the 2 tasks that I mentioned earlier in the blog to search and write the blog. Please note that for the `task_search`, we are providing the tools to perform the web searches. Also note that we are configuring which agent is going to perform this task.

```
    task_search = Task(
        description="""Search for all the details about the  {topic}
                    Your final answer MUST be a consolidated content that can be used for blogging
                    This content should be well organized, and should be very easy to read""",
        expected_output='A comprehensive 10000 words information about {topic}',
        max_inter=3,
        tools=[search_tool, web_tool],
        agent=researcher)

    task_post = Task(
        description="""Write a well structured blog and at max 10000 words.
                    The Blog should also include sample programs and codes, tutorials, and all the content that is useful for the readers
                    Also explain the concepts, architecture in detail
                    Once the blog is created, create a new file called blog.md, and save the blog in that file
                    """,
        expected_output='A comprehensive 20 paragraph blog on {topic} in markdown format',
        agent=blogger)
```

To bring the tasks and agents together, we deifying the crew object, and we are passing the agents and tasks, that will be working in the crew, we are also defining out process strategy as sequential. We are calling the kickoff method of the crew to get the agents work on the provided task.

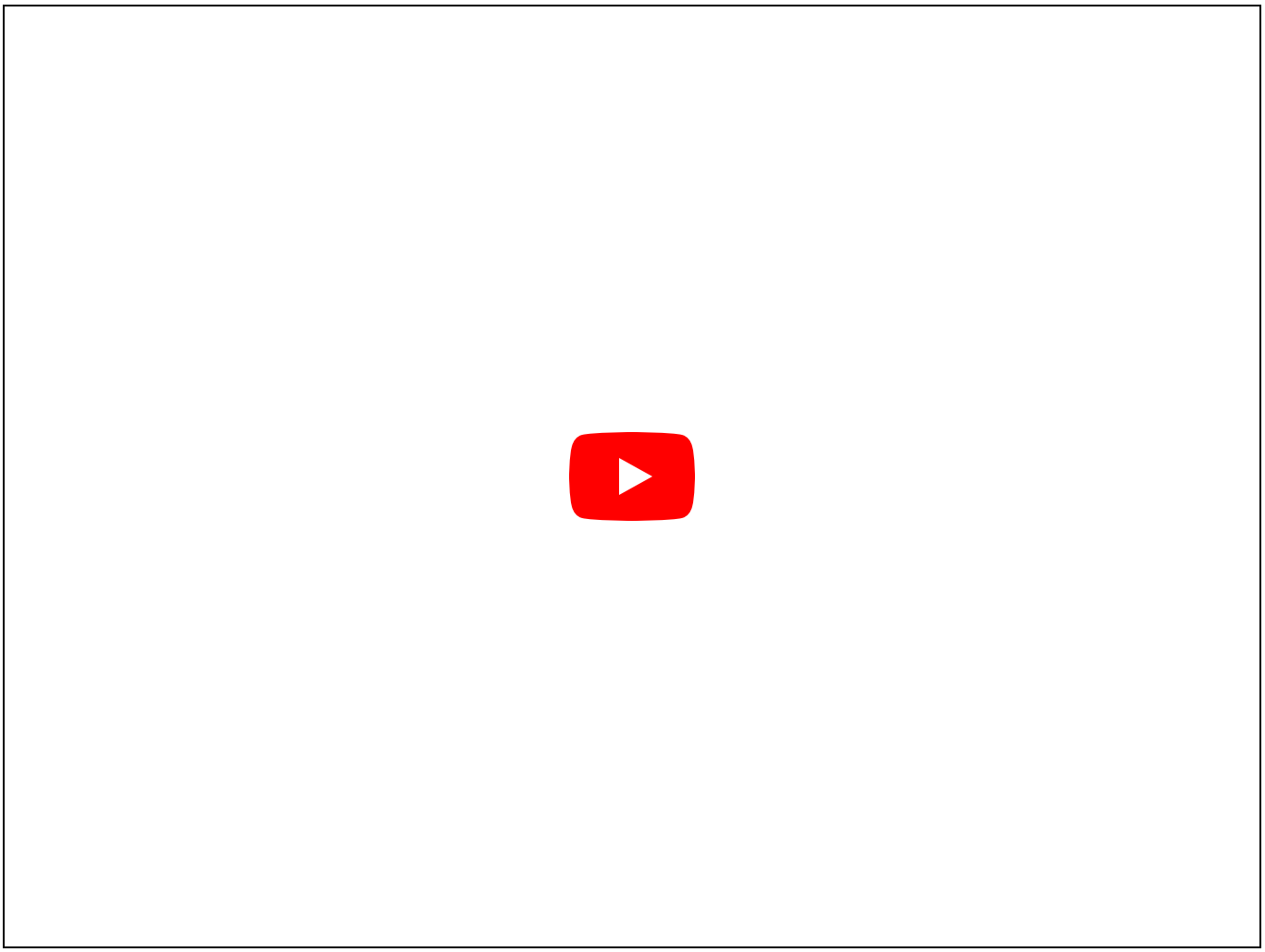
```
58          agents=[researcher, blogger],
59          tasks=[task_search, task_post],
60          verbose=2,
61          process=Process.sequential )
62
63      result = crew.kickoff()
64      return result
```

Now coming to the main block of the code, we are receiving the command line argument as the topic, and pass this topic to the `kickoffTheCrew` method that we defined above. This will execute the system

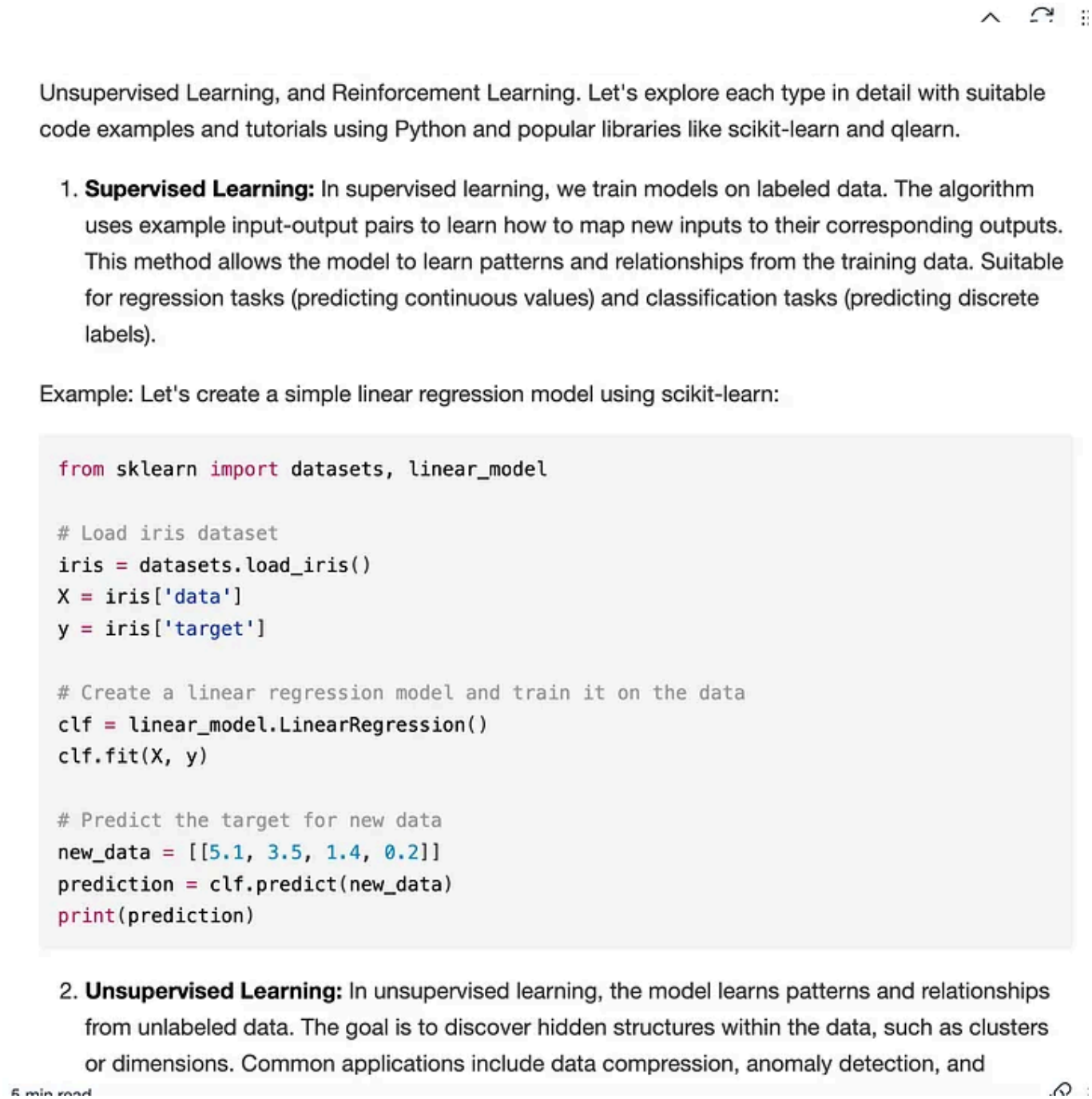
```
67  n = len(sys.argv)
68
69  if n == 2 :
70      topic = sys.argv[1]
71      result = kickoffTheCrew(topic)
72      print (result)
73  else :
74      print ("Please pass topic as parameter. Usage python3 blogger.py topic")
75
```

## Results

Lets now run this code. I recorded a complete screencast video for you to see the results (since we have set the verbose mode, we see the complete execution of the agents. Following the video of the screencast.



Here is a screenshot of the final markdown that our multi-agent blogger application has created.



Unsupervised Learning, and Reinforcement Learning. Let's explore each type in detail with suitable code examples and tutorials using Python and popular libraries like scikit-learn and qlearn.

1. **Supervised Learning:** In supervised learning, we train models on labeled data. The algorithm uses example input-output pairs to learn how to map new inputs to their corresponding outputs. This method allows the model to learn patterns and relationships from the training data. Suitable for regression tasks (predicting continuous values) and classification tasks (predicting discrete labels).

Example: Let's create a simple linear regression model using scikit-learn:

```
from sklearn import datasets, linear_model

# Load iris dataset
iris = datasets.load_iris()
X = iris['data']
y = iris['target']

# Create a linear regression model and train it on the data
clf = linear_model.LinearRegression()
clf.fit(X, y)

# Predict the target for new data
new_data = [[5.1, 3.5, 1.4, 0.2]]
prediction = clf.predict(new_data)
print(prediction)
```

2. **Unsupervised Learning:** In unsupervised learning, the model learns patterns and relationships from unlabeled data. The goal is to discover hidden structures within the data, such as clusters or dimensions. Common applications include data compression, anomaly detection, and

6 min read

There you go, Isn't it amazing, what we could do with multi-agent systems. I am super excited and I have been exploring various frameworks, I look forward to hearing from you. Hope this was useful.

I will be coming back soon, with more examples and other frameworks that I am playing around with, until then…code and have fun :-D

You can find the code in my GitHub repository abvijaykumar/crewai-blogs (github.com).

LIm | Generative Ai Tools | AI | Crew Ai | Agents

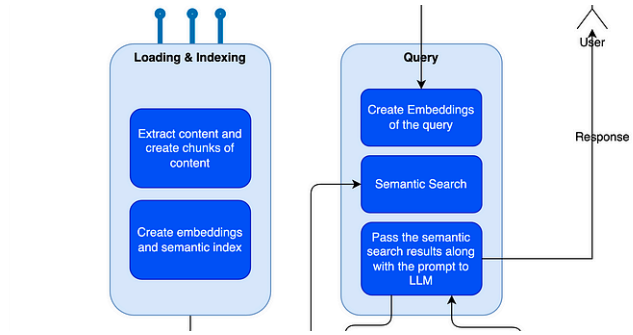
## Written by A B Vijay Kumar

1.7K Followers

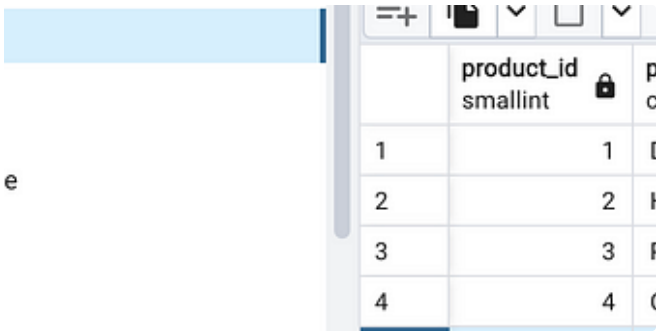
IBM Fellow, Master Inventor, Mobile, RPi & Cloud Architect & Full-Stack Programmer

Follow

### More from A B Vijay Kumar




A B Vijay Kumar




A B Vijay Kumar

## Retrieval Augmented Ge

Implement the RAG technique using Langchain, and LlamaIndex for conversation...

7 min read · Feb 6, 2024

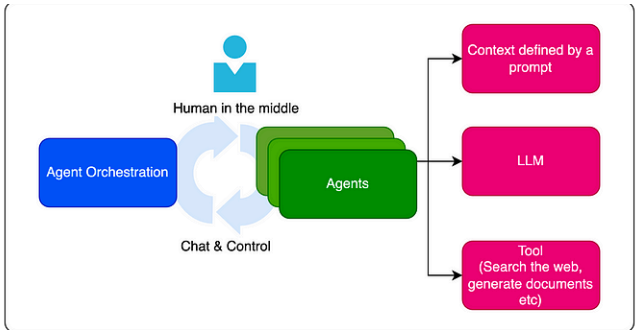
340 7

## Retrieval Augmented

Implement the RAG technique using Langchain, and LlamaIndex for conversation...

4 min read · Feb 7, 2024

148 6



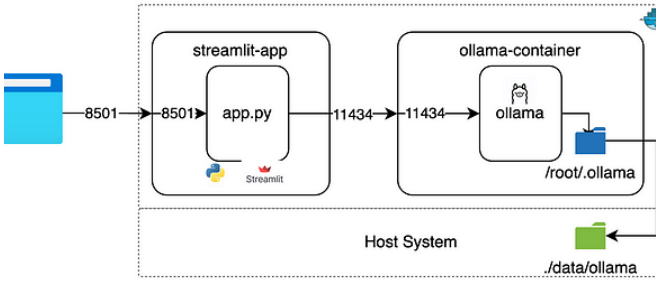

A B Vijay Kumar

## Multi-Agent Architectures

Multi-Agent systems are LLM applications that are changing the automation landscape...

5 min read · Mar 24, 2024

496




A B Vijay Kumar

## Ollama—Build a ChatBot with Langchain, Ollama & Deploy on...

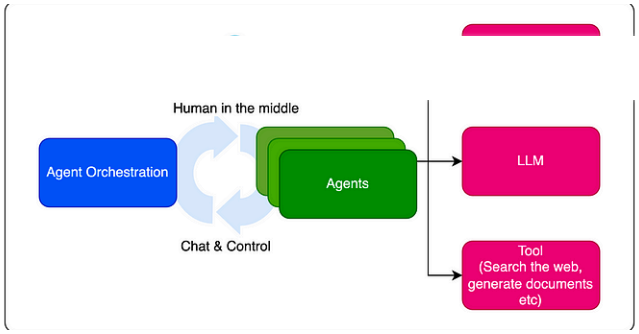
Working with Ollama to run models locally, build LLM applications that can be deployed...

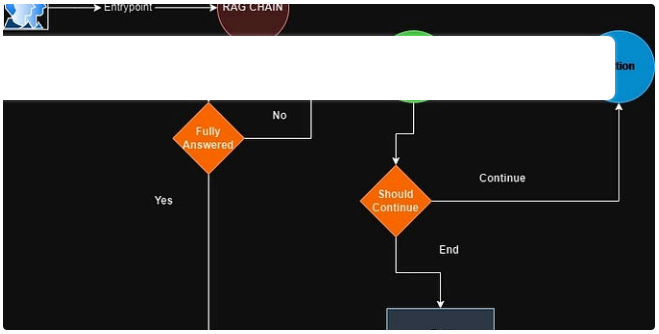
5 min read · Feb 21, 2024

440 2

See all from A B Vijay Kumar

# Recommended from Medium




A B Vijay Kumar

## Multi-Agent Architectures

Multi-Agent systems are LLM applications that are changing the automation landscape...

5 min read · Mar 24, 2024

496

Plaban Nayak in The AI Forum

## Implementing Agentic RAG using Langchain

Native RAG

12 min read · Mar 31, 2024

328

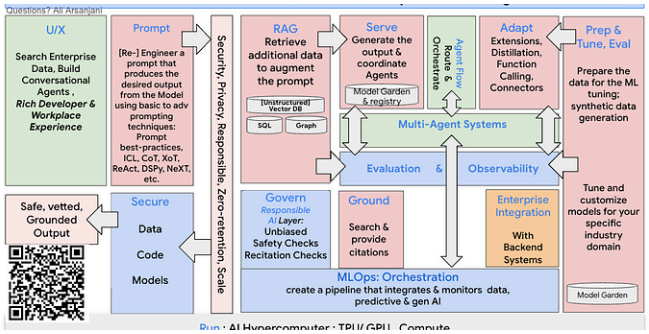
### Lists

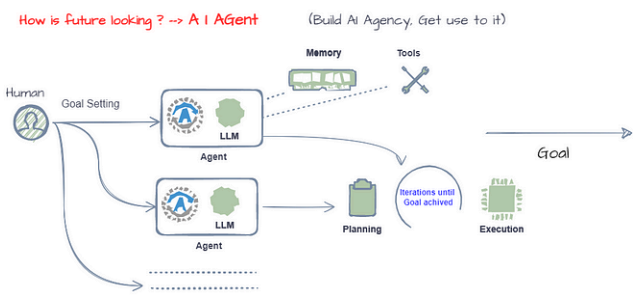
**Generative AI Recommended Reading**
52 stories · 1039 saves

**What is ChatGPT?**
9 stories · 354 saves

**Natural Language Processing**
1449 stories · 958 saves

**The New Chatbots: ChatGPT, Bard, and Beyond**
12 stories · 374 saves



Ali Arsanjani

## The GenAI Reference Architecture

26 min read · Apr 28, 2024

613 6

Aniket Hingane

## Why Entire AI field is headed towards AI Agents

GenAI is just the beginning, what comes next is AI Agent.

· 6 min read · Apr 2, 2024

1.6K 16

JIN in **AI monks.io**

## 7 Free AI Tools Revolutionizing Workflows

Discover the Top AI Tools for Streamlining Processes Across Industries—From Image...

· 7 min read · Apr 28, 2024

321 1

Gabriel Rennó in LatinXinAI

## The future is Agentic—crewAI

Easily creating Agentic Workflows with crewAI, LLama3 and groq.

· 18 min read · Apr 26, 2024

160 2

See more recommendations