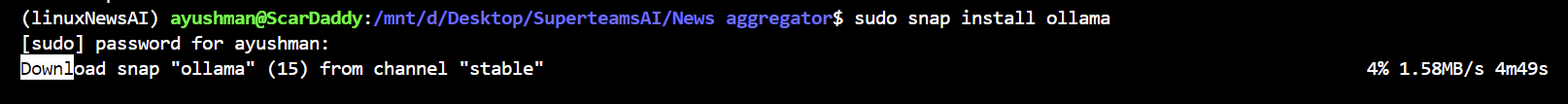
# 3.CrewAi Content planner writer and editor

## Ollama installation

# for ollama

pip install ollama

sudo snap install ollama

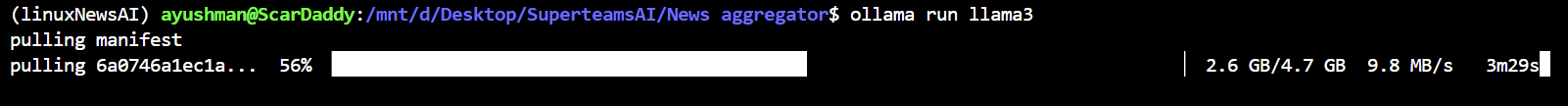


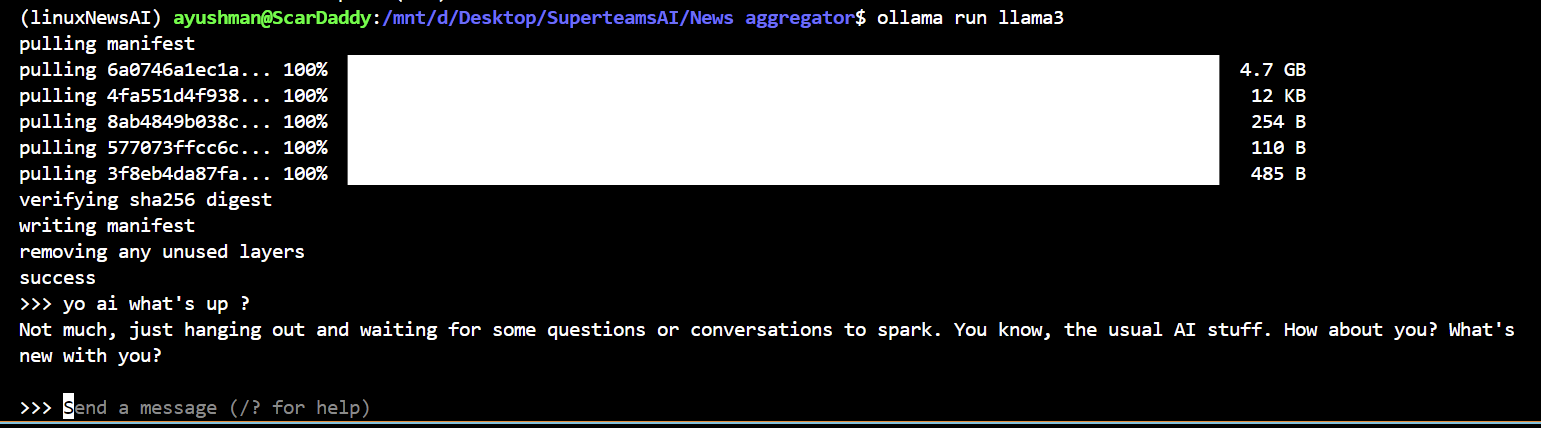
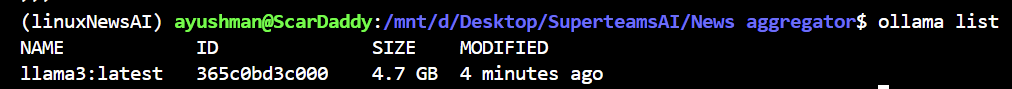


## Llama 3 installation

Then download the llama3 model from the command prompt

ollama run llama3



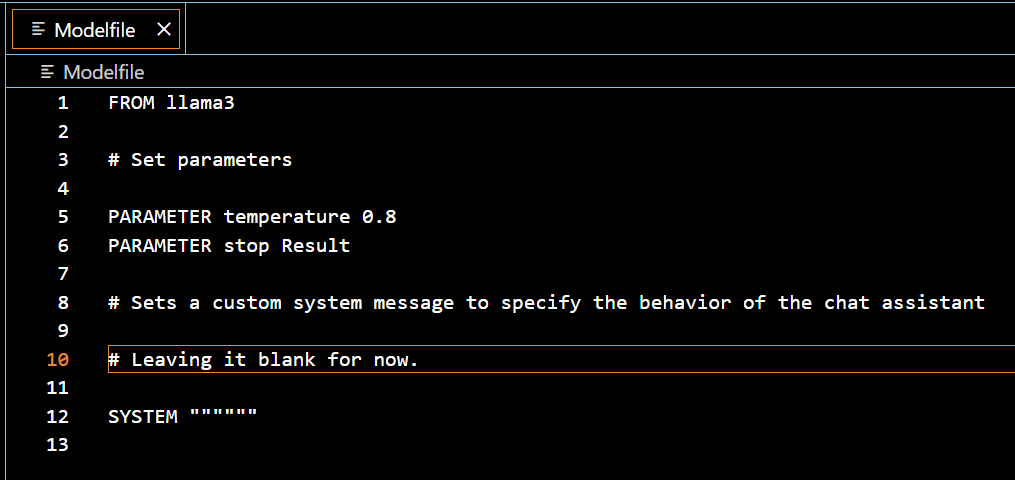
* Crewai

pip install crewai==0.28.8 crewai\_tools==0.1.6 langchain\_community==0.0.29

**Set up the LLM as Llama3**

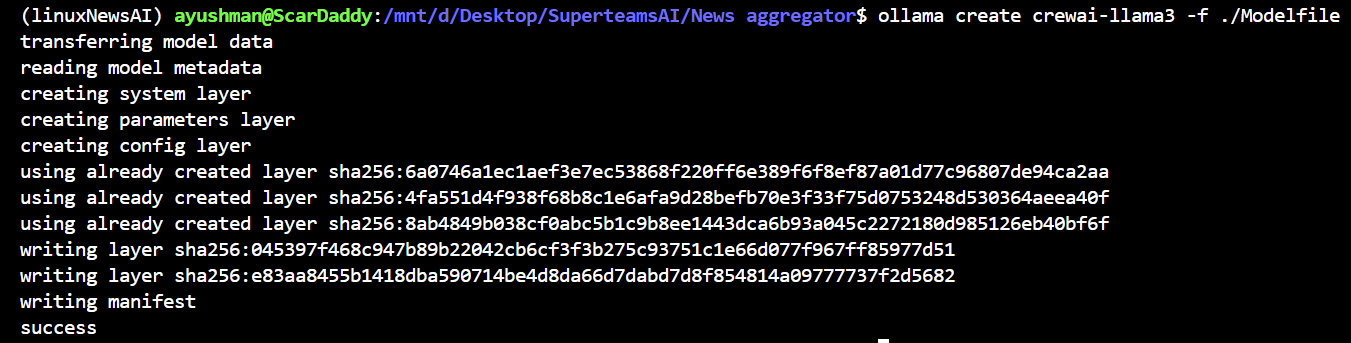
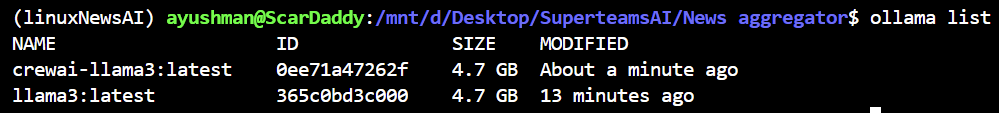
Create a ModelFile similar the one below in your project directory.

FROM llama3  
  
# Set parameters  
  
PARAMETER temperature 0.8  
PARAMETER stop Result  
  
# Sets a custom system message to specify the behavior of the chat assistant  
  
# Leaving it blank for now.  
  
SYSTEM """"""



**Run the following command in the command prompt**

ollama create crewai-llama3 -f ./Modelfile

from crewai import Agent, Task, Crew  
from langchain\_openai import ChatOpenAI  
import os  
os.environ["OPENAI\_API\_KEY"] = "NA"  
  
llm = ChatOpenAI(  
 model = "crewai-llama3",  
 base\_url = "http://localhost:11434/v1")

## Create Agents to plan ,write and edit the content for the blogpost

An agent is an **autonomous unit** programmed to:

* Perform tasks
* Make decisions
* Communicate with other agents

**Agent Attributes**

***Role:****Defines the agent’s function within the crew. It determines the kind of tasks the agent is best suited for.*

***Goal:****The individual objective that the agent aims to achieve. It guides the agent’s decision-making process.*

***Backstory:****Provides context to the agent’s role and goal, enriching the interaction and collaboration dynamics.*

***LLM****:(optional)Represents the language model that will run the agent. It dynamically fetches the model name from the OPENAI\_MODEL\_NAME environment variable, defaulting to "gpt-4" if not specified.*

***Tools****:(optional)Set of capabilities or functions that the agent can use to perform tasks. Expected to be instances of custom classes compatible with the agent's execution environment. Tools are initialized with a default value of an empty list.*

***Function Calling LLM****:(optional)Specifies the language model that will handle the tool calling for this agent, overriding the crew function calling LLM if passed. Default is None.*

***Max Iter****:(optional)The maximum number of iterations the agent can perform before being forced to give its best answer. Default is 25.*

***Max RPM****:(optional)The maximum number of requests per minute the agent can perform to avoid rate limits. It's optional and can be left unspecified, with a default value of None.*

***max\_execution\_time****:(optional)Maximum execution time for an agent to execute a task It's optional and can be left unspecified, with a default value of None, menaning no max execution time*

***Verbose:****(optional)Setting this to True configures the internal logger to provide detailed execution logs, aiding in debugging and monitoring. Default is False.*

***Allow Delegation:****(optional)Agents can delegate tasks or questions to one another, ensuring that each task is handled by the most suitable agent. Default is True.*

***Step Callback:****(optional)A function that is called after each step of the agent. This can be used to log the agent's actions or to perform other operations. It will overwrite the crew step\_callback.*

***Cache:****(optional)Indicates if the agent should use a cache for tool usage. Default is True*

### Content Planner Agent

planner = Agent(  
 role="Content Planner",  
 goal="Plan engaging and factually accurate content on {topic}",  
 backstory="You're working on planning a blog article "  
 "about the topic: {topic} in 'https://medium.com/'."  
 "You collect information that helps the "  
 "audience learn something "  
 "and make informed decisions. "  
 "You have to prepare a detailed "  
 "outline and the relevant topics and sub-topics that has to be a part of the"  
 "blogpost."  
 "Your work is the basis for "  
 "the Content Writer to write an article on this topic.",  
 llm=llm,  
 allow\_delegation=False,  
 verbose=True  
)

### Content Writer Agent

writer = Agent(  
 role="Content Writer",  
 goal="Write insightful and factually accurate "  
 "opinion piece about the topic: {topic}",  
 backstory="You're working on a writing "  
 "a new opinion piece about the topic: {topic} in 'https://medium.com/'. "  
 "You base your writing on the work of "  
 "the Content Planner, who provides an outline "  
 "and relevant context about the topic. "  
 "You follow the main objectives and "  
 "direction of the outline, "  
 "as provide by the Content Planner. "  
 "You also provide objective and impartial insights "  
 "and back them up with information "  
 "provide by the Content Planner. "  
 "You acknowledge in your opinion piece "  
 "when your statements are opinions "  
 "as opposed to objective statements.",  
 allow\_delegation=False,  
 llm=llm,  
 verbose=True  
)

### Content Editor Agent

editor = Agent(  
 role="Editor",  
 goal="Edit a given blog post to align with "  
 "the writing style of the organization 'https://medium.com/'. ",  
 backstory="You are an editor who receives a blog post "  
 "from the Content Writer. "  
 "Your goal is to review the blog post "  
 "to ensure that it follows journalistic best practices,"  
 "provides balanced viewpoints "  
 "when providing opinions or assertions, "  
 "and also avoids major controversial topics "  
 "or opinions when possible.",  
 llm=llm,  
 allow\_delegation=False,  
 verbose=True  
)

**Create Task**

Tasks within crewAI can be collaborative, requiring multiple agents to work together. This is managed through the task properties and orchestrated by the Crew’s process, enhancing teamwork and efficiency.

Task Attributes

***Description:****A clear, concise statement of what the task entails.*

***Agent:****The agent responsible for the task, assigned either directly or by the crew’s process.*

***Expected Output:****A detailed description of what the task’s completion looks like.*

***Tools:****(optional)The functions or capabilities the agent can utilize to perform the task.*

***Async Execution:****(optional)If set, the task executes asynchronously, allowing progression without waiting for completion.*

***Context:****(optional)Specifies tasks whose outputs are used as context for this task.*

***Config:****(optional)Additional configuration details for the agent executing the task, allowing further customization.*

***Output JSON****:(optional)Outputs a JSON object, requiring an OpenAI client. Only one output format can be set.*

***Output Pydantic****:(optional)Outputs a Pydantic model object, requiring an OpenAI client. Only one output format can be set.*

***Output File:****(optional)Saves the task output to a file. If used with Output JSON or Output Pydantic, specifies how the output is saved.*

***Callback:****(optional)A Python callable that is executed with the task's output upon completion.*

***Human Input****:(optional)Indicates if the task requires human feedback at the end, useful for tasks needing human oversight.*

### Create planner task

plan = Task(  
 description=(  
 "1. Prioritize the latest trends, key players, "  
 "and noteworthy news on {topic}.\n"  
 "2. Identify the target audience, considering "  
 "their interests and pain points.\n"  
 "3. Develop a detailed content outline including "  
 "an introduction, key points, and a call to action.\n"  
 "4. Include SEO keywords and relevant data or sources."  
 ),  
 expected\_output="A comprehensive content plan document "  
 "with an outline, audience analysis, "  
 "SEO keywords, and resources.",  
 agent=planner,  
)

### Create Writer Task

write = Task(  
 description=(  
 "1. Use the content plan to craft a compelling "  
 "blog post on {topic}.\n"  
 "2. Incorporate SEO keywords naturally.\n"  
 "3. Sections/Subtitles are properly named "  
 "in an engaging manner.\n"  
 "4. Ensure the post is structured with an "  
 "engaging introduction, insightful body, "  
 "and a summarizing conclusion.\n"  
 "5. Proofread for grammatical errors and "  
 "alignment with the brand's voice.\n"  
 ),  
 expected\_output="A well-written blog post "  
 "in markdown format, ready for publication, "  
 "each section should have 2 or 3 paragraphs.",  
 agent=writer,  
)

### Create Editor Task

edit = Task(  
 description=("Proofread the given blog post for "  
 "grammatical errors and "  
 "alignment with the brand's voice."),  
 expected\_output="A well-written blog post in markdown format, "  
 "ready for publication, "  
 "each section should have 2 or 3 paragraphs.",  
 agent=editor  
)

**Note**: The benefit of using *multiple strings* :

varname = "line 1 of text"  
 "line 2 of text"

versus the *triple quote docstring*:

varname = """line 1 of text  
 line 2 of text  
 """

is that it can avoid adding those whitespaces and newline characters, making it better formatted to be passed to the LLM.

**Creating the Crew**

* Create your crew of Agents
* Pass the tasks to be performed by those agents.
* **Note**: *For this simple example*, the tasks will be performed sequentially (i.e they are dependent on each other), so the *order* of the task in the list *matters*.
* verbose=2 allows you to see all the logs of the execution.

crew = Crew(  
 agents=[planner, writer, editor],  
 tasks=[plan, write, edit],  
 verbose=2  
)

**Run the crew**

inputs = {"topic":"Comparative study of LangGraph, Autogen and Crewai for building multi-agent system."}  
result = crew.kickoff(inputs=inputs)

### Response

from IPython.display import Markdown,display

display(Markdown(result))

**Comparative Study of LangGraph, Autogen, and Crewai for Building Multi-Agent Systems**

===========================================================

As technology continues to advance, the development of artificial intelligence (AI) and multi-agent systems has become increasingly important. The ability to simulate complex real-world scenarios makes them invaluable in fields such as logistics, healthcare, and finance. However, choosing the right framework for building a multi-agent system can be a daunting task.

## **LangGraph: A Scalable Yet Complex Solution**

LangGraph is a popular choice among developers due to its scalability and flexibility. It allows for complex relationships between agents, making it suitable for projects involving dynamic environments. While it excels in these areas, LangGraph's limited support for complex relationships and steep learning curve may deter some users. For instance, the framework might be more challenging for newcomers or those with limited experience in developing multi-agent systems.

## **Autogen: A Simplified Solution with Limitations**

Autogen takes a different approach by simplifying the development process and supporting complex relationships. This makes it an attractive choice for those who value ease of use over customization options. However, Autogen's limited scalability and customization options may limit its applicability in certain projects. For instance, if a project requires advanced AI features or extensive customizations, Autogen might not be the best fit.

## **Crewai: A Highly Customizable Solution with Complexities**

Crewai stands out due to its high customizability and support for complex relationships. This makes it ideal for projects that require a high degree of flexibility. While it excels in these areas, Crewai's steep learning curve and limited community support may pose challenges for some users. For instance, the framework might be more suitable for experienced developers or those willing to invest time in learning its intricacies.

# **Comparison Table**

| **Strength** | **LangGraph** | **Autogen** | **Crewai** |
| --- | --- | --- | --- |
| Scalability | High | Low | N/A |
| Flexibility | High | Medium | High |
| Complex Relationships Support | Limited | High | High |
| Learning Curve | Steep | Medium | Steep |
| Customization Options | Limited | Limited | High |
| Community Support | Large | Medium | Small |

**Conclusion:** Each framework has its unique strengths and weaknesses. LangGraph excels in scalability and flexibility but struggles with complex relationships. Autogen shines in simplifying the development process and supporting complex relationships, while Crewai offers high customizability. When choosing a framework for building a multi-agent system, consider the project's specific requirements and the team's expertise. By doing so, developers can make informed decisions that meet their project goals.

**Call to Action:** Are you ready to learn more about LangGraph, Autogen, and Crewai? Dive deeper into each framework's strengths and weaknesses to find the perfect solution for your next multi-agent systems project.

**SEO Keywords:** comparative study, multi-agent systems, LangGraph, Autogen, Crewai, artificial intelligence

**Resources:**

* "A Survey of Multi-Agent System Development Frameworks" by XYZ
* "Design Patterns for Multi-Agent Systems" by ABC