Developing an AI agent using **Crew AI** involves organizing AI agents into a collaborative structure to solve complex tasks. Crew AI allows you to build and orchestrate **multi-agent systems** (MAS), where each agent can specialize in a specific task and work together toward a shared goal.

□ What Is Crew AI?

Crew AI is a framework built in Python that helps you:

- Create **agents** (autonomous roles powered by LLMs or tools)
- Assign them **roles**, **goals**, and **tools**
- Coordinate them into a **crew** to solve tasks collaboratively

---

□ Prerequisites

1. **Python 3.10+**
2. Basic knowledge of:
    - Python programming
    - LLMs (e.g., OpenAI's GPT)
3. Install packages:

```
4. pip install crewai
```

Optional (for enhanced tools):

- `openai`, `langchain`, `chromadb`, etc.

□ Step-by-Step Guide to Build a Crew AI Agent

1. □ Install and Setup

```
pip install crewai
```

Set up your `.env` file with your OpenAI key:

```
export OPENAI_API_KEY="your-api-key"
```

Or use in code:

```
import os
os.environ["OPENAI_API_KEY"] = "your-api-key"
```

---

2. □♂□ Define Your Agents

Each **agent** has:

- A name
- A role (like "Data Analyst")
- A goal (like "Analyze sales data")
- A language model (e.g., GPT-4)

```
from crewai import Agent
from langchain.chat_models import ChatOpenAI

# Define an agent
analyst = Agent(
    role="Data Analyst",
    goal="Analyze the quarterly sales data and provide insights",
    backstory="You are a meticulous analyst with a knack for finding trends
in complex data.",
    verbose=True,
    allow_delegation=False,
    llm=ChatOpenAI(model="gpt-4")
)
```

---

3. □ Define Tasks

Tasks are assigned to agents. A task includes:

- A description
- Who performs it (the agent)
- Optional dependencies on other tasks

```
from crewai import Task

task1 = Task(
    description="Extract the top 5 sales regions from the dataset.",
    agent=analyst,
    expected_output="List of regions and their sales numbers"
)
```

---

4. □□ Define the Crew

Now wrap it all together into a **Crew**, which handles orchestration.

```
from crewai import Crew

crew = Crew(
    agents=[analyst],
    tasks=[task1],
    verbose=True
)
```

---

5. □ Run the Crew

```
result = crew.kickoff()
print(result)
```

Done! Your AI agent has now completed a task autonomously.

□ Example: Multi-Agent Use Case

Imagine building a **market research crew**:

```
researcher = Agent(
    role="Market Researcher",
    goal="Research latest trends in renewable energy",
    backstory="Expert at scouring the web and compiling data",
    verbose=True,
    llm=ChatOpenAI(model="gpt-4")
)

writer = Agent(
    role="Technical Writer",
    goal="Create a report based on research findings",
    backstory="Experienced in crafting clear and concise reports",
    verbose=True,
    llm=ChatOpenAI(model="gpt-4")
)

task_research = Task(
    description="Find the top 5 trends in renewable energy from 2025",
    agent=researcher,
    expected_output="Bullet points of each trend with source links"
)

task_report = Task(
    description="Write a summary report of the renewable energy trends",
    agent=writer,
    depends_on=[task_research]
)

crew = Crew(
    agents=[researcher, writer],
    tasks=[task_research, task_report],
    verbose=True
)

result = crew.kickoff()
print(result)
```

□ Useful Concepts

Concept	Description
Agent	An LLM-powered role that performs tasks
Task	A unit of work for an agent
Crew	A group of agents + tasks
Dependencies	Order of task execution
LLM	OpenAI GPT-3.5, GPT-4, Claude, etc.

□ Tips for Success

- Keep agent goals **specific and clear**
- Use **task dependencies** to structure execution
- Use **verbose=True** for debugging and tracing output
- Agents can use **tools** (e.g., web search, Python execution) via LangChain

---

□ Resources

- Crew AI GitHub
- LangChain Tools
- OpenAI API

Let's build a **CrewAI multi-agent system** using **Ollama** running the **LLaMA 3** model locally instead of OpenAI's GPT. This is 100% local, private, and customizable.

□ What We'll Do

- Use **Ollama** with **LLaMA 3**
- Create 2 agents:
  - A **Researcher**
  - A **Writer**
- Connect them using **CrewAI**
- Run locally with **no cloud API required**

□ Requirements

Open your command prompt in windows pc and type following commands

1. □ Install Required Packages

```
pip install crewai langchain langchain-community
```

2. □ Install Ollama and LLaMA 3

Install Ollama if you haven't already:

```
# Install LLaMA 3 8B
ollama pull llama3
```

Confirm it's working:

```
ollama run llama3
```

□ Full Updated Code Using Ollama + LLaMA 3(main.py)

```
from crewai import Agent, Task, Crew
from langchain_community.llms import Ollama

# □ Use local LLaMA 3 model via Ollama
llm = Ollama(model="llama3", temperature=0.7)

# □□ Agent 1: Market Researcher
researcher = Agent(
    role="Market Researcher",
    goal="Discover the top 5 renewable energy trends in 2025",
    backstory=(
        "You are a skilled researcher specializing in environmental
technology. "
        "You're excellent at finding recent, relevant, and impactful data."
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)

# ✍□ Agent 2: Technical Writer
writer = Agent(
    role="Technical Writer",
```

```
    goal="Create a well-written summary report on renewable energy trends",
    backstory=(
        "You're a technical writer with a background in green technologies. "
        "You transform complex data into easy-to-understand reports."
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)

# □ Task 1: Research trends
task1 = Task(
    description=(
        "Identify and explain the top 5 renewable energy trends expected in
2025. "
        "Include a short description for each and why it's significant."
    ),
    expected_output="A bullet list of 5 trends with brief explanations",
    agent=researcher,
)

# □ Task 2: Write the report
task2 = Task(
    description=(
        "Write a summary report in professional tone, using the researcher's
findings. "
        "Keep it under 500 words, informative yet engaging."
    ),
    expected_output="A summary report suitable for publication",
    agent=writer,
    depends_on=[task1],
)

# □ Build the Crew
crew = Crew(
    agents=[researcher, writer],
    tasks=[task1, task2],
    verbose=True,  # Shows detailed step-by-step execution
)

# □ Run the crew
final_output = crew.kickoff()

# □ Print the final report
print("\n□ Final Report:\n")
print(final_output)
```

□ Output Example

Make sure Ollama is running (background server):

Ollama run llama3

And Then, in your terminal:

```
python main.py

[Researcher]: Task started: Identifying renewable energy trends...
[Researcher]: Output:
- Solar PV efficiency increases
- Green hydrogen development
- Offshore wind tech...
...

[Writer]: Writing final report using research...

□ Final Report:

In 2025, renewable energy will continue advancing across several key
domains...
```

Note:-

Step-by-Step Guide to Run CrewAI with Ollama (LLaMA 3)

□ 1. Install Ollama and Pull LLaMA 3

If you haven't installed Ollama yet:

- Download and install it from: https://ollama.com/download
- Once installed, run in terminal:

```
ollama pull llama3
```

This downloads the `llama3` model (~4GB+). Wait for it to finish.

Test it works:

```
ollama run llama3
```

You should be able to chat with LLaMA 3 locally.

3. Install Python Dependencies

```
pip install crewai langchain langchain-community
```

These install:

- `crewai`: The multi-agent framework
- `langchain`: Used to connect to LLMs
- `langchain-community`: Needed for Ollama integration

4. Save the Python Script

Create a new file called `main.py` and paste this full code:

```
from crewai import Agent, Task, Crew
from langchain_community.llms import Ollama

# □ Use local LLaMA 3 model via Ollama
llm = Ollama(model="llama3", temperature=0.7)

# □□ Agent 1: Market Researcher
researcher = Agent(
    role="Market Researcher",
    goal="Discover the top 5 renewable energy trends in 2025",
    backstory=(
        "You are a skilled researcher specializing in environmental
technology. "
        "You're excellent at finding recent, relevant, and impactful data."
    ),
    verbose=True,
    allow_delegation=False,
```

```
    llm=llm,
)

# ✍ Agent 2: Technical Writer
writer = Agent(
    role="Technical Writer",
    goal="Create a well-written summary report on renewable energy trends",
    backstory=(
        "You're a technical writer with a background in green technologies. "
        "You transform complex data into easy-to-understand reports."
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)

#  Task 1: Research trends
task1 = Task(
    description=(
        "Identify and explain the top 5 renewable energy trends expected in
2025. "
        "Include a short description for each and why it's significant."
    ),
    expected_output="A bullet list of 5 trends with brief explanations",
    agent=researcher,
)

#  Task 2: Write the report
task2 = Task(
    description=(
        "Write a summary report in professional tone, using the researcher's
findings. "
        "Keep it under 500 words, informative yet engaging."
    ),
    expected_output="A summary report suitable for publication",
    agent=writer,
    depends_on=[task1],
)

#  Build the Crew
crew = Crew(
    agents=[researcher, writer],
    tasks=[task1, task2],
    verbose=True,  # Shows detailed step-by-step execution
)

#  Run the crew
final_output = crew.kickoff()

#  Print the final report
print("\n Final Report:\n")
print(final_output)
```

▶ 5. Run the Script

Make sure Ollama is running (background server):

Then, in your terminal:

```
python main.py
```

You'll see output as each **agent** performs its **task**, and finally prints the **report**.

□ Final Output Example

```
[CrewAI]: Starting Task 1: Research renewable energy trends...
[Researcher]: Here's my list of top 5 renewable energy trends:
 - Trend 1: Green Hydrogen...
 - Trend 2: Solar panel efficiency...
...

[CrewAI]: Starting Task 2: Writing report based on research...
[Writer]: Here's the final report...

□ Final Report:

In 2025, renewable energy is poised to evolve in 5 key directions...
```

□ Optional Improvements

Feature	How to Add
Add more agents	Define more `Agent()` + `Task()`
Export to file	Use Python file I/O
Build GUI	Use Streamlit
Build API	Use FastAPI or Flask
Add tools	Use LangChain tool integrations