Agentic RAG using CrewAI

10 min read · Feb 7, 2025

Ansuman Das Follow

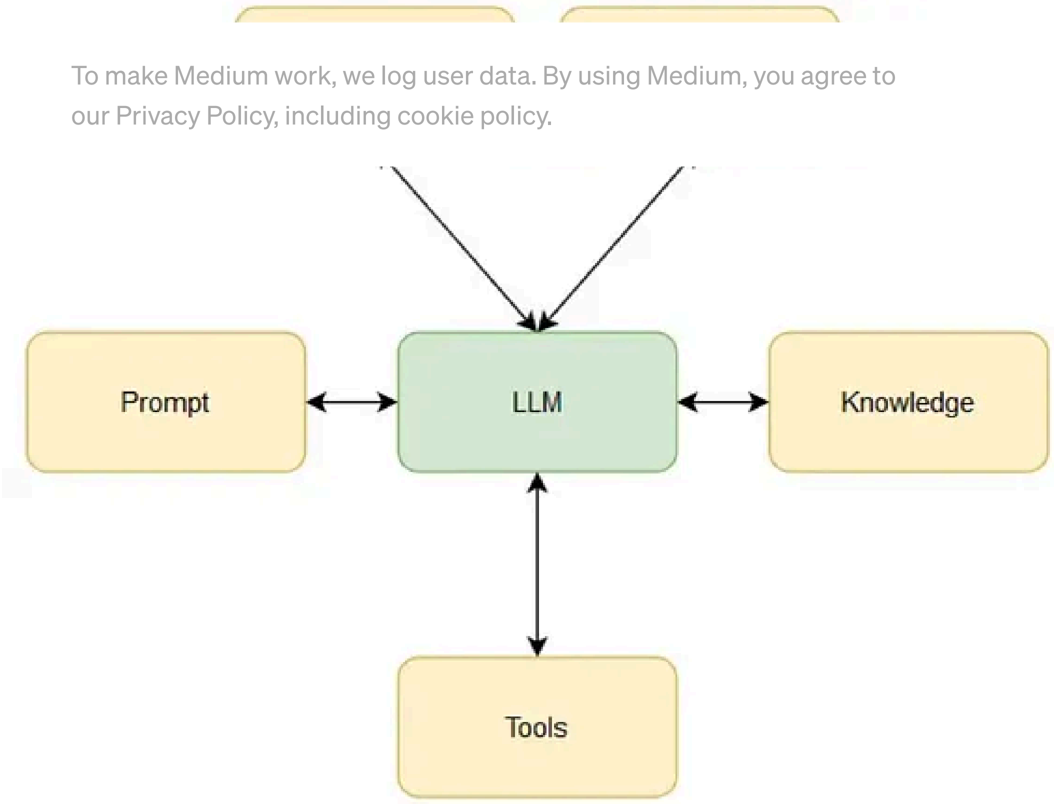
In this post, we will explore the trending topic of **Agentic RAG (Retrieval-Augmented Generation)** and demonstrate how we've implemented it using **CrewAI** with code. We'll cover the key components, including **CrewAI Agents** and the **Agentic RAG** workflow, to showcase how these elements work together to handle complex queries efficiently. Whether you're new to the concept or looking to understand the implementation, this blog will guide you through the process and provide practical insights.

What is an Agent?

LLM agents (Large Language Model agents) are advanced AI systems that help solve problems, answer questions, and perform tasks. They can understand and process language, plan actions, and learn from past experiences.

These agents are great for handling complex problems that require multiple steps. They analyze data, retrieve information, and adapt based on what they've already tried. LLM agents can use external tools like web searches and calculations to help solve problems.

They also have memory, allowing them to remember past actions and improve over time. In short, LLM agents are like smart assistants that can think through problems and adjust their approach.



Basic Components of an Agent

What is Agentic RAG?

Agentic RAG is an enhanced version of the traditional RAG (Retrieval-Augmented Generation) system. In the vanilla RAG pipeline, the process mainly focuses on retrieving relevant information from a predefined data source, like a vector index, and then generating a response using that data. However, it has limitations, such as the inability to access external tools or resources beyond the initial dataset. Vanilla RAG works well for simple retrieval tasks but struggles with more complex problems or situations that require additional data sources, like web searches or calculations.

Agentic RAG, on the other hand, addresses these limitations by **integrating AI agents into the pipeline. These agents can access a variety of external tools, such as web search engines, calculators, APIs, and more**, to enhance the retrieval process. This allows Agentic RAG to retrieve information from a wider range of sources and adapt more effectively to dynamic tasks. The agents in Agentic RAG can also break down complicated tasks into smaller, manageable subtasks, making it easier to handle complex problems.

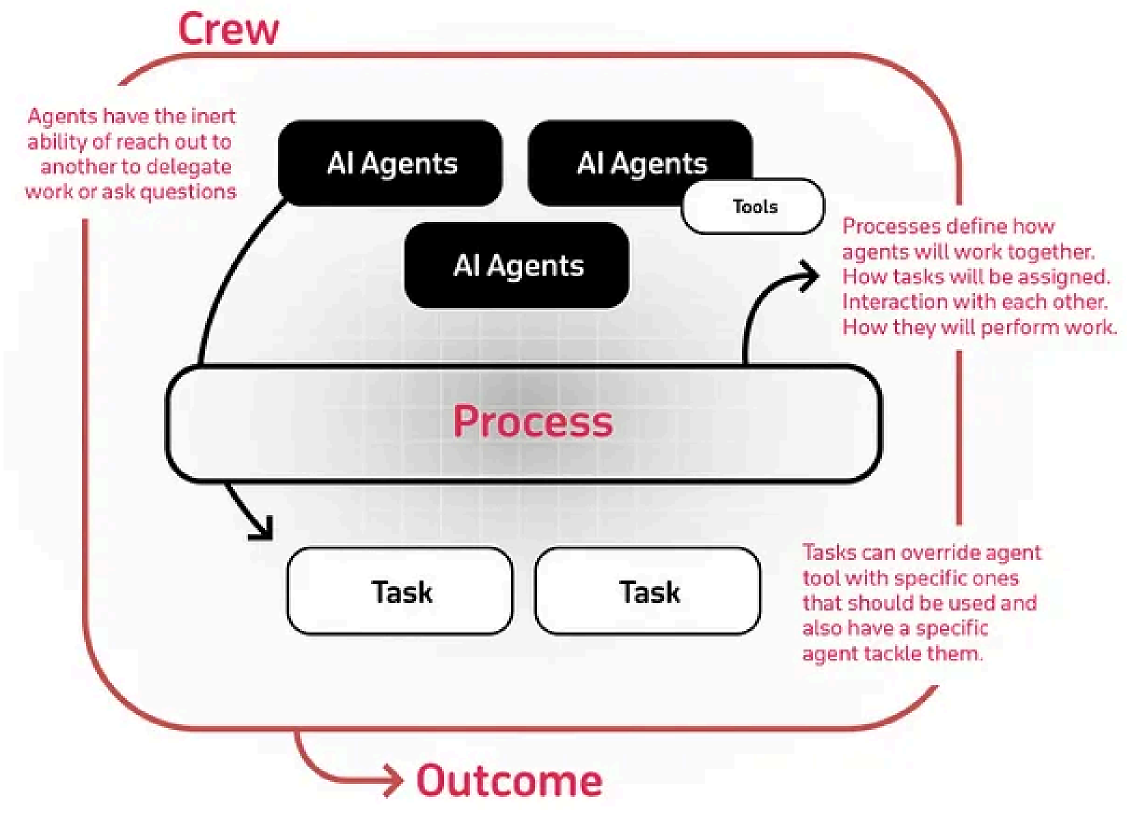
Additionally, Agentic RAG introduces a more intelligent, modular framework where specialized agents collaborate to address different parts of a task. Each agent plays a specific role, such as retrieving data, conducting web searches, or generating responses using large language models (LLMs). These agents have memory, so they can remem

for future steps. This ols, makes Agentic RAG far more efficient, accurate, and adaptable compared to the vanilla RAG approach.

What is crewAI?

CrewAI is an advanced framework designed to coordinate autonomous AI agents, allowing you to create AI teams that work together to tackle complex tasks. Each agent within the team has its own specific role, tools, and goals, making the entire system more efficient and capable.

Imagine assembling an expert group, where each member (agent) brings specialized skills and knowledge. These agents collaborate seamlessly, with their combined efforts driving toward a shared objective. CrewAI helps you harness the power of multiple AI agents, enabling them to work in harmony to solve problems that would be challenging for a single agent alone.


Crew
Agents have the inert ability of reach out to another to delegate work or ask questions
AI Agents
AI Agents
Tools
AI Agents
Processes define how agents will work together. How tasks will be assigned. Interaction with each other. How they will perform work.
Process
Task
Task
Tasks can override agent tool with specific ones that should be used and also have a specific agent tackle them.
Outcome

Agents

In CrewAI, an agent is a core component designed to handle specific tasks within a multi-agent system. Each agent has its own defined role, goal, backstory, and can be equipped v

CrewAI Agent Characteristics:

- **Role:** The function or expertise the agent brings to the team, such as a researcher or reviewer.
- **Goal:** The specific objective the agent is aiming to achieve, guiding its actions and decisions.
- **Backstory:** A context or personality that adds depth to the agent, enriching its interactions with other agents.
- **Tools (Optional):** Tools like web search engines or data analysis utilities can be added to extend the agent's capabilities.
- **LLM (Optional):** A language model that powers the agent, enabling it to understand and generate language.
- **Memory (Optional):** Agents can store past interactions to help guide future decisions and maintain continuity.

Tasks
A task in CrewAI refers to a specific assignment or objective that an agent is responsible for completing. Tasks provide all the necessary details for execution, such as descriptions, assigned agents, required tools, and expected outputs.

Key Components of a Task:

- **Description**: A text-based description of what the task entails.
- **Agent**: The specific agent responsible for carrying out the task.
- **Expected Output**: The desired outcome of the task, such as a "Yes" or "No" answer.

Crew
In CrewAI, a crew represents a collaborative group of agents working together to complete tasks. A crew defines the strategy for task execution, agent collaboration, and overall workflow.

Key Components of a Crew:

- **Agents** expertise.
- **Tasks:** A list of tasks assigned to the crew, outlining what needs to be done and who is responsible for each part.

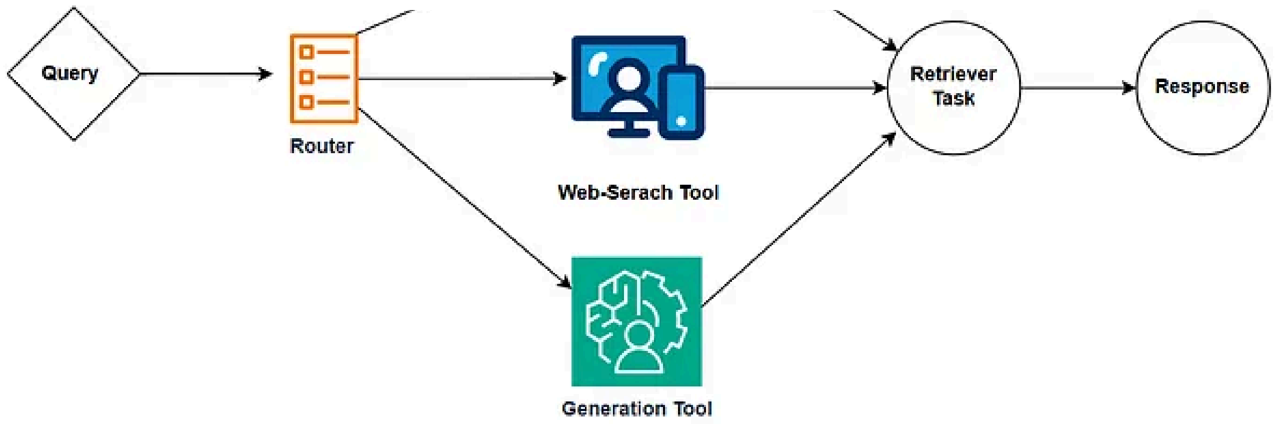
Together, these components make CrewAI a powerful framework for coordinating multiple agents, enhancing collaboration, and improving task efficiency.

How It All Works Together

1. The Crew organizes the overall operation
2. AI Agents work on their specialized tasks
3. The Process ensures smooth collaboration
4. Tasks get completed to achieve the goal

Code with explanation

In this project, we aim to answer questions by using different sources of information. If the question is related to a research paper, we store the embedding in chromadb vector database. which helps us quickly find relevant answers using semantic search . For questions about current news or events, we search for up-to-date information on the web to provide accurate answers. For more general questions, we use a language model (LLM) to generate responses. This approach ensures that the system can handle different types of questions and provide the most appropriate answers based on the source of information.



Agentic RAG workflow for Our Use-case

Installing Required Packages.

```
!pip install crewai crewai_tools langchain_community sentence-transformers lang
```

OpneAI key : To access LLM (you can access LLM via groq also which is opensource)

SERPER_API_KEY: For GoogleSerperAPIWrapper (https://serper.dev/): sign in via email and get your API key for free)

Import Library and all key :

```
from langchain_community.utilities import GoogleSerperAPIWrapper
from crewai.tools import BaseTool
from pydantic import Field
from crewai.tools import PDFSearchTool
from langchain_openai import ChatOpenAI
os.environ['SERPER_API_KEY'] = 'paste your key'
os.environ['OPENAI_API_KEY']='your key'
```

Loading opneAI LLM :(we are using gpt-4 for thsi task)

```
opne_a
llm = (
    ope
    openai_api_key=opne_ai_key,
    model_name="gpt-4",
    temperature=0.1,
    max_tokens=1000,
)
```

In this example, we will use a research paper on hate speech detection, utilizing a language model (LLM) for understanding. We will download the paper from the web, then store it as a PDF in the runtime environment. This will allow us to process the paper for answering questions related to it. The embeddings of the paper's contents will be stored in a vector store for efficient searching later on

```
import requests
#paper url
pdf_url='https://arxiv.org/pdf/2405.01577'
response = requests.get(pdf_url)

with open('hatespeech.pdf', 'wb') as file:
    file.write(response.content)
```

In our example, we have two agents:

1. **Routing Agent:** This agent decides whether a question should be answered using the **vector store** (for research papers) or by searching the **web** (for current events or general knowledge).
2. **Retrieval Agent:** This agent retrieves relevant information from the **vector store** when the question relates to a research paper, by searching the stored embeddings for the most relevant details.

These agents work together to ensure accurate and efficient answers based on the type of question asked.

```
Router_
  role=
  goal=
  backstory=(
    "You are an expert at routing a user question to a vectorstore or web searc
    "Use the vectorstore for questions on hate speech or tiny llm or finetuning
    "use web-search for question on latest news or recent topics."
    "use generation for generic questions otherwise"
  ),
  verbose=True,
  allow_delegation=False,
  llm=llm,
)
Retriever_Agent = Agent(
role="Retriever",
goal="Use the information retrieved from the vectorstore to answer the questior
backstory=(
    "You are an assistant for question-answering tasks."
    "Use the information present in the retrieved context to answer the questic
    "You have to provide a clear concise answer within 200 words."
),
verbose=True,
allow_delegation=False,
llm=llm,
)
```

Define the Tool:

In our system, we use several **tools** to help provide answers based on different sources of information:

1. **Generation Tool:** This tool uses a **language model (LLM)** to generate answers for general questions. When a question doesn't fit with a research paper or current event, the generation tool creates a response by drawing on its trained knowledge base
2. **Web Search Tool:** This tool is responsible for searching the **web** to find up-to-date answers for questions related to recent events, news, or any topic where real-time information is needed. It fetches relevant data from online sources.
3. **PDF Search Tool:** This tool is part of the **Retrieval-Augmented Generation (RAG)** process. It searches for specific information within a **PDF file** (like the research

paper we're using) to extract relevant sections or details needed to answer questic

These tools work together to ensure that questions are answered from the most appropriate source — whether it's a PDF document, the web, or generated by the LLM.

```
search = GoogleSerperAPIWrapper

class SearchTool(BaseTool):
    name: str = "Search"
    description: str = "Useful for search-based queries. Use this to find curre
    search: GoogleSerperAPIWrapper = Field(default_factory=GoogleSerperAPIWrapp

    def _run(self, query: str) -> str:
        """Execute the search query and return results"""
        try:
            return self.search.run(query)
        except Exception as e:
            return f"Error performing search: {str(e)}"
class GenerationTool(BaseTool):
    name: str = "Generation_tool"
    description: str = "Useful for generic-based queries. Use this to find  inf
    #llm: ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

    def _run(self, query: str) -> str:
      llm=ChatOpenAI(model_name="gpt-4o-mini", temperature=0)
      """Execute the search query and return results"""
      return llm.invoke(query)
generation_tool=GenerationTool()
web_search_tool = SearchTool()
pdf_search_tool = PDFSearchTool(
    pdf="hatespeech.pdf",
)
```

In our example, we have two tasks:

1. **Router Task (By Router Agent):** The Router Agent determines whether the question should be answered using the **PDF Search Tool**, **Web Search Tool**, or **Generation Tool** based on the question's type.

2. **Retriever Task (By Retriever Agent, Using Context from Router Task, Tools: Web Search, PDF Search, and Generation):** The Retriever Agent uses the context

from the Router Task to fetch or generate the answer using the appropriate tools (web, F

In short, the Router Task directs the question, and the Retriever Task gets the answer using the right tools.

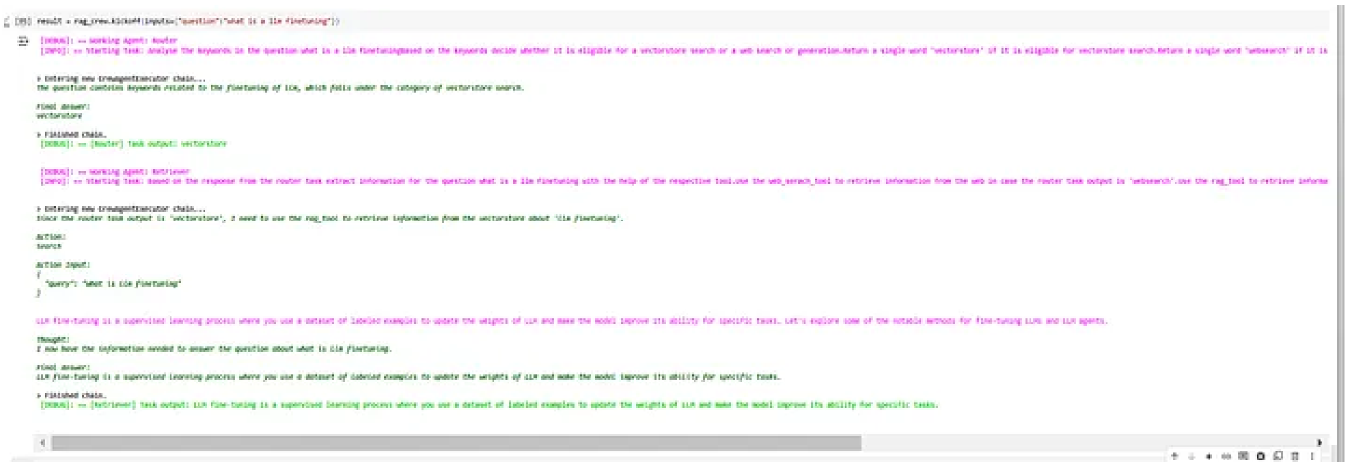
```
router_task = Task(
    description=("Analyse the keywords in the question {question}"
    "Based on the keywords decide whether it is eligible for a vectorstore sear
    "Return a single word 'vectorstore' if it is eligible for vectorstore searc
    "Return a single word 'websearch' if it is eligible for web search."
    "Return a single word 'generate' if it is eligible for generation."
    "Do not provide any other premable or explaination."
    ),
    expected_output=("Give a  choice 'websearch' or 'vectorstore' or 'generate'
    "Do not provide any other premable or explaination."),
    agent=Router_Agent,
   )

retriever_task = Task(
    description=("Based on the response from the router task extract informatio
    "Use the web_serach_tool to retrieve information from the web in case the r
    "Use the rag_tool to retrieve information from the vectorstore in case the 
    "otherwise generate the output basedob your own knowledge in case the route
    ),
    expected_output=("You should analyse the output of the 'router_task'"
    "If the response is 'websearch' then use the web_search_tool to retrieve ir
    "If the response is 'vectorstore' then use the rag_tool to retrieve informa
    "If the response is 'generate' then use then use generation_tool ."
    "otherwise say i dont know if you dont know the answer"

    "Return a claer and consise text as response."),
    agent=Retriever_Agent,
    context=[router_task],
    tools=[pdf_search_tool,web_search_tool,generation_tool],
)
```

In our usecase , we define the **Crew** by assigning specific **tasks** to the corresponding **agents.** The **task list** and **agent list** help structure the workflow, ensuring each agent performs its designated role efficiently.

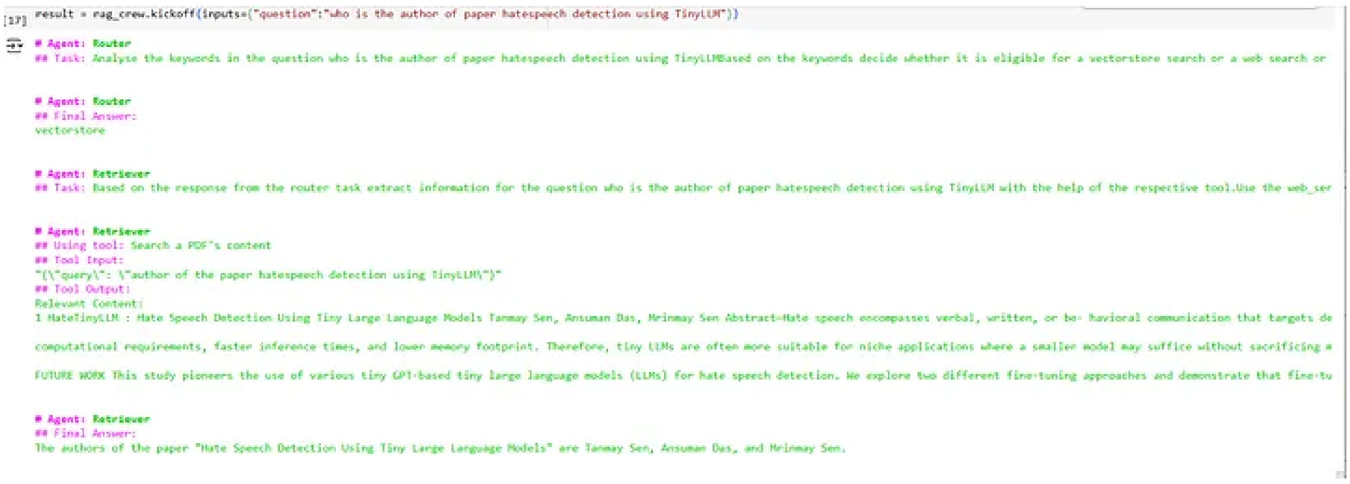
```
rag_cre
    age
    tas
    verbose=True,

)
```

To initiate the execution of the defined **Crew** with a question, we pass a question to the system, which will then trigger the **Router Task** and subsequently the **Retriever Task.**

```
result = rag_crew.kickoff(inputs={"question":"what is a llm finetuning"})
```



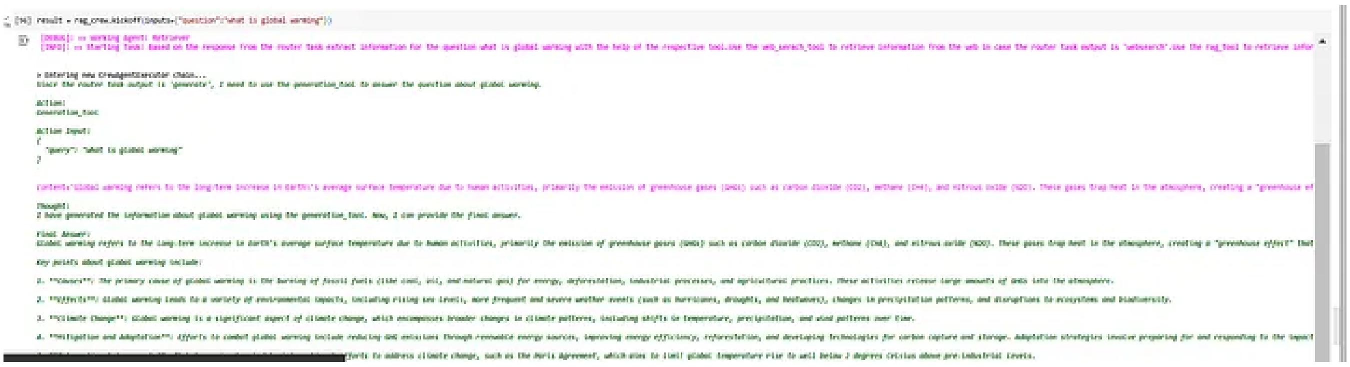
```
[35] result = rag_crew.kickoff(inputs={"question":"what is a llm finetuning"})

 [DEBUG]: == Working Agent: Router
 [INFO]: == Starting Task: Analyse the keywords in the question what is a llm finetuningBased on the keywords decide whether it is eligible for a vectorstore search or a web search or generation.Return a single word 'vectorstore' if it is eligible for vectorstore search.Return a single word 'websearch' if it is


> Entering new CrewAgentExecutor chain...
The question contains keywords related to the finetuning of llm, which falls under the category of vectorstore search.

Final Answer:
vectorstore

> Finished chain.
 [DEBUG]: == [Router] Task output: vectorstore


 [DEBUG]: == Working Agent: Retriever
 [INFO]: == Starting Task: Based on the response from the router task extract information for the question what is a llm finetuning with the help of the respective tool.Use the web_serach_tool to retrieve information from the web in case the router task output is 'websearch'.Use the rag_tool to retrieve informa


> Entering new CrewAgentExecutor chain...
Since the router task output is 'vectorstore', I need to use the rag_tool to retrieve information from the vectorstore about 'llm finetuning'.

Action:
Search

Action Input:
{
  "query": "what is llm finetuning"
}

LLM fine-tuning is a supervised learning process where you use a dataset of labeled examples to update the weights of LLM and make the model improve its ability for specific tasks. Let's explore some of the notable methods for fine-tuning LLMs and LLM agents.

Thought:
I now have the information needed to answer the question about what is llm finetuning.

Final Answer:
LLM fine-tuning is a supervised learning process where you use a dataset of labeled examples to update the weights of LLM and make the model improve its ability for specific tasks.

> Finished chain.
 [DEBUG]: == [Retriever] Task output: LLM fine-tuning is a supervised learning process where you use a dataset of labeled examples to update the weights of LLM and make the model improve its ability for specific tasks.
```



```
[17] result = rag_crew.kickoff(inputs={"question":"who is the author of paper hatespeech detection using TinyLLM"})

# Agent: Router
## Task: Analyse the keywords in the question who is the author of paper hatespeech detection using TinyLLMBased on the keywords decide whether it is eligible for a vectorstore search or a web search or


# Agent: Router
## Final Answer:
vectorstore


# Agent: Retriever
## Task: Based on the response from the router task extract information for the question who is the author of paper hatespeech detection using TinyLLM with the help of the respective tool.Use the web_ser


# Agent: Retriever
## Using tool: Search a PDF's content
## Tool Input:
"{\"query\": \"author of the paper hatespeech detection using TinyLLM\"}"
## Tool Output:
Relevant Content:
1 HateTinyLLM : Hate Speech Detection Using Tiny Large Language Models Tanmay Sen, Ansuman Das, Mrinmay Sen Abstract—Hate speech encompasses verbal, written, or be- havioral communication that targets de

computational requirements, faster inference times, and lower memory footprint. Therefore, tiny LLMs are often more suitable for niche applications where a smaller model may suffice without sacrificing a

FUTURE WORK This study pioneers the use of various tiny GPT-based tiny large language models (LLMs) for hate speech detection. We explore two different fine-tuning approaches and demonstrate that fine-tu


# Agent: Retriever
## Final Answer:
The authors of the paper "Hate Speech Detection Using Tiny Large Language Models" are Tanmay Sen, Ansuman Das, and Mrinmay Sen.
```

Lets ask something generic Question.For a **generic question**, the process is slightly different. Instead of looking for specific information in a PDF or the web, the system will use the **Generation Tool** to generate an answer

```
result = rag_crew.kickoff(inputs={"question":"what is global warming"})
```



```
[14] result = rag_crew.kickoff(inputs={"question":"what is global warming"})

 [DEBUG]: == Working Agent: Retriever
 [INFO]: == Starting Task: Based on the response from the router task extract information for the question what is global warming with the help of the respective tool.Use the web_serach_tool to retrieve information from the web in case the router task output is 'websearch'.Use the rag_tool to retrieve infor

> Entering new CrewAgentExecutor chain...
Since the router task output is 'generate', I need to use the generation_tool to answer the question about global warming.

Action:
Generation_tool

Action Input:
{
  "query": "what is global warming"
}

Content:'Global warming refers to the long-term increase in Earth\'s average surface temperature due to human activities, primarily the emission of greenhouse gases (GHGs) such as carbon dioxide (CO2), methane (CH4), and nitrous oxide (N2O). These gases trap heat in the atmosphere, creating a "greenhouse ef

Thought:
I have generated the information about global warming using the generation_tool. Now, I can provide the final answer.

Final Answer:
Global warming refers to the long-term increase in Earth's average surface temperature due to human activities, primarily the emission of greenhouse gases (GHGs) such as carbon dioxide (CO2), methane (CH4), and nitrous oxide (N2O). These gases trap heat in the atmosphere, creating a "greenhouse effect" that

Key points about global warming include:

1. **Causes**: The primary cause of global warming is the burning of fossil fuels (like coal, oil, and natural gas) for energy, deforestation, industrial processes, and agricultural practices. These activities release large amounts of GHGs into the atmosphere.

2. **Effects**: Global warming leads to a variety of environmental impacts, including rising sea levels, more frequent and severe weather events (such as hurricanes, droughts, and heatwaves), changes in precipitation patterns, and disruptions to ecosystems and biodiversity.

3. **Climate Change**: Global warming is a significant aspect of climate change, which encompasses broader changes in climate patterns, including shifts in temperature, precipitation, and wind patterns over time.

4. **Mitigation and Adaptation**: Efforts to combat global warming include reducing GHG emissions through renewable energy sources, improving energy efficiency, reforestation, and developing technologies for carbon capture and storage. Adaptation strategies involve preparing for and responding to the impact

[illegible] efforts to address climate change, such as the Paris Agreement, which aims to limit global temperature rise to well below 2 degrees Celsius above pre-industrial levels.
```

As the above question is about global warming which is not related to our pdf or vector store so its routing to internet and giving the results .

Conclusion

In this system, we've demonstrated how **Agentic Retrieval-Augmented Generation (RAG)** works using a well-defined **Crew AI** to handle different types of questions effectively. By combining the strengths of **agents**, **tools**, and **tasks**, we can provide accurate and context-appropriate responses based on the nature of the question.By structuring the system in this way, we create an efficient workflow where each agent performs a specific task, ensuring that the right tools are used for the right question type. This **Agentic RAG** approach is powerful for answering complex queries across multiple domains, from research papers to real-time web data and general knowledge, making it adaptable and scalable for a variety of use cases.

Thanks

Ansuman Das

www.linkedin.com/in/ansuman-das-3778896b

Agents LIm Generative Ai Tools Crew Ai Machine Learning

Follow

Written by Ansuman Das

26 followers · 3 following

Data Scientist || GCP||ML

No responses yet

Write a response

What are your thoughts?

More from Ansuman Das

Ansuman Das

Rethinkin

We've enter aren't just tools running code, but intelligent agents…

Jul 16 3

Ansuman Das

Finetuning Large Language Models

A Large Language Model is an advanced artificial intelligence (AI) system designed to process, understand, and generate human-like text…

Jul 19, 2023 12 1

Ansuman Das

Explainable AI(XAI) Using Shapash Library

What is Explainable AI:

Jun 8, 2022 40 1

Ansuman Das

Understanding Large Concept Models (LCMs): The Next Step in AI's Evolution

Artificial Intelligence (AI) has transformed many aspects of our lives, with advancements like Large Language Models (LLMs) driving much of…

Mar 16 4

See all from Ansuman Das

Recommended from Medium

In Artificial Intelligence in Plain English by Piyush Agnihotri

Building Agentic RAG with LangGraph: Mastering Adaptive RAG for Production

Build intelligent RAG systems that know when to retrieve documents, search the web, or generate responses directly

Jul 20 2.2K 30

In AI Simplified in Plain English by Devashish Datt Mamgain

Building Enterprise-Grade AI Agents with LangChain (LangGraph), Microsoft AutoGen, and Microsoft…

Executive Summary

Jun 21 10

In Towards Dev by Dharmendra Pratap Singh

Beyond Retrieval: Building an Agentic AI RAG System and Evaluating it with RAGAS

Retrieval-Au ystems. By combining t

Oct 31

In DataDrivenInvestor by Yibin Ng

Multi-Agent Workflows using AutoGen for Stock Recommendation

With step-by-step instructions from zero to one

Aug 1 53 1

In ITNEXT by Tsvetan Tsvetkov

Building a Text-to-SQL AI Agent with pure Java, LangChain4j, and Ollama: Ask your

How about a

Jul 17 72

David W

Agentic RAG—Catalytic Agents Integrating Structured and Unstructured Data

This article continues the AI journey I began in my last article. Like before, the focus is on petroleum subsurface data—the domain I...

Oct 21

See more recommendations