Connecting to any LLM

Connect CrewAI to LLMs

Default LLM

By default, CrewAI uses OpenAI's GPT-4 model for language processing. You can configure your agents to use a different model or API. This guide shows how to connect your agents to various LLMs through environment variables and direct instantiation.

CrewAI offers flexibility in connecting to various LLMs, including local models via Ollama and different APIs like Azure. It's compatible with all LangChain LLM components, enabling diverse integrations for tailored AI solutions.

CrewAI Agent Overview

The `Agent` class is the cornerstone for implementing AI solutions in CrewAI. Here's an updated overview reflecting the latest codebase changes:

- **Attributes**:
  - `role`: Defines the agent's role within the solution.
  - `goal`: Specifies the agent's objective.
  - `backstory`: Provides a background story to the agent.
  - `llm`: Indicates the Large Language Model the agent uses. By default, it uses the GPT-4 model defined in the environment variable "OPENAI_MODEL_NAME".
  - `function_calling_llm` *Optional*: Will turn the ReAct crewAI agent into a function calling agent.
  - `max_iter`: Maximum number of iterations for an agent to execute a task, default is 15.
  - `memory`: Enables the agent to retain information during and a across executions. Default is `False`.
  - `max_rpm`: Maximum number of requests per minute the agent's execution should respect. Optional.
  - `verbose`: Enables detailed logging of the agent's execution. Default is `False`.
  - `allow_delegation`: Allows the agent to delegate tasks to other agents, default is `True`.
  - `tools`: Specifies the tools available to the agent for task execution. Optional.
  - `step_callback`: Provides a callback function to be executed after each step. Optional.
  - `cache`: Determines whether the agent should use a cache for tool usage. Default is `True`.

```
# Required
os.environ["OPENAI_MODEL_NAME"]="gpt-4-0125-preview"

# Agent will automatically use the model defined in the environment variable
example_agent = Agent(
  role='Local Expert',
  goal='Provide insights about the city',
  backstory="A knowledgeable local guide.",
  verbose=True,
  memory=True
)
```

Ollama Integration

Ollama is preferred for local LLM integration, offering customization and privacy benefits. To integrate Ollama with CrewAI, set the appropriate environment variables as shown below.

Setting Up Ollama

- **Environment Variables Configuration**: To integrate Ollama, set the following environment variables:

```
OPENAI_API_BASE='http://localhost:11434/v1'
OPENAI_MODEL_NAME='openhermes'  # Adjust based on available model
OPENAI_API_KEY=''
```

Ollama Integration (ex. for using Llama 2 locally)

Download Ollama.

After setting up the Ollama, Pull the Llama2 by typing following lines into the terminal `ollama pull llama2`.

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

```
FROM llama2

# 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 """"""
```

Create a script to get the base model, which in our case is llama2, and create a model on top of that with ModelFile above. PS: this will be ".sh" file.

```
#!/bin/zsh

# variables
model_name="llama2"
custom_model_name="crewai-llama2"

#get the base model
ollama pull $model_name

#create the model file
ollama create $custom_model_name -f ./Llama2ModelFile
```

Go into the directory where the script file and ModelFile is located and run the script.

Enjoy your free Llama2 model that powered up by excellent agents from crewai.

```
from crewai import Agent, Task, Crew
from langchain_openai import ChatOpenAI
import os
os.environ["OPENAI_API_KEY"] = "NA"

llm = ChatOpenAI(
    model = "crewai-llama2",
    base_url = "http://localhost:11434/v1")

general_agent = Agent(role = "Math Professor",
                      goal = """Provide the solution to the students that are asking mathematical questions a
                      backstory = """You are an excellent math professor that likes to solve math questions i
                      allow_delegation = False,
                      verbose = True,
                      llm = llm)
task = Task (description="""what is 3 + 5""",
             agent = general_agent,
             expected_output="A numerical answer.")

crew = Crew(
            agents=[general_agent],
            tasks=[task],
            verbose=2
        )

result = crew.kickoff()

print(result)
```

HuggingFace Integration

There are a couple of different ways you can use HuggingFace to host your LLM.

Your own HuggingFace endpoint

```
from langchain_community.llms import HuggingFaceEndpoint

llm = HuggingFaceEndpoint(
    endpoint_url="<YOUR_ENDPOINT_URL_HERE>",
    huggingfacehub_api_token="<HF_TOKEN_HERE>",
    task="text-generation",
    max_new_tokens=512
)

agent = Agent(
    role="HuggingFace Agent",
    goal="Generate text using HuggingFace",
    backstory="A diligent explorer of GitHub docs.",
    llm=llm
)
```

From HuggingFaceHub endpoint

```
from langchain_community.llms import HuggingFaceHub

llm = HuggingFaceHub(
    repo_id="HuggingFaceH4/zephyr-7b-beta",
    huggingfacehub_api_token="<HF_TOKEN_HERE>",
    task="text-generation",
)
```

OpenAI Compatible API Endpoints

Switch between APIs and models seamlessly using environment variables, supporting platforms like FastChat, LM Studio, and Mistral AI.

Configuration Examples

FastChat

```
OPENAI_API_BASE="http://localhost:8001/v1"
```

```
OPENAI_MODEL_NAME='oh-2.5m7b-q51'
OPENAI_API_KEY=NA
```

LM Studio

Launch LM Studio and go to the Server tab. Then select a model from the dropdown menu then wait for it to load. Once it's loaded, click the green Start Server button and use the URL, port, and API key that's shown (you can modify them). Below is an example of the default settings as of LM Studio 0.2.19:

```
OPENAI_API_BASE="http://localhost:1234/v1"
OPENAI_API_KEY="lm-studio"
```

Mistral API

```
OPENAI_API_KEY=your-mistral-api-key
OPENAI_API_BASE=https://api.mistral.ai/v1
OPENAI_MODEL_NAME="mistral-small"
```

Solar

```
from langchain_community.chat_models.solar import SolarChat
# Initialize language model
os.environ["SOLAR_API_KEY"] = "your-solar-api-key"
llm = SolarChat(max_tokens=1024)

Free developer API key available here: https://console.upstage.ai/services/solar
Langchain Example: https://github.com/langchain-ai/langchain/pull/18556
```

text-gen-web-ui

```
OPENAI_API_BASE=http://localhost:5000/v1
OPENAI_MODEL_NAME=NA
OPENAI_API_KEY=NA
```

Cohere

```
from langchain_community.chat_models import ChatCohere
# Initialize language model
os.environ["COHERE_API_KEY"] = "your-cohere-api-key"
llm = ChatCohere()

Free developer API key available here: https://cohere.com/
Langchain Documentation: https://python.langchain.com/docs/integrations/chat/cohere
```

Azure Open AI Configuration

For Azure OpenAI API integration, set the following environment variables:

```
AZURE_OPENAI_VERSION="2022-12-01"
AZURE_OPENAI_DEPLOYMENT=""
AZURE_OPENAI_ENDPOINT=""
AZURE_OPENAI_KEY=""
```

Example Agent with Azure LLM

```
from dotenv import load_dotenv
from crewai import Agent
from langchain_openai import AzureChatOpenAI

load_dotenv()

azure_llm = AzureChatOpenAI(
    azure_endpoint=os.environ.get("AZURE_OPENAI_ENDPOINT"),
    api_key=os.environ.get("AZURE_OPENAI_KEY")
)

azure_agent = Agent(
  role='Example Agent',
  goal='Demonstrate custom LLM configuration',
  backstory='A diligent explorer of GitHub docs.',
  llm=azure_llm
)
```

Conclusion

Integrating CrewAI with different LLMs expands the framework's versatility, allowing for customized, efficient AI solutions across various domains and platforms.