ReAct vs. Function Calling: Choosing the Right AI Agent Approach in LlamaIndex

Ken Huang, CISSP
AI Book Author |Speaker |DistributedApps.AI |OWASP Top 10 for LLM Co-Author | NIST GenAI Contributor| EC-Council…

October 31, 2024

LlamaIndex framework has both ReAct agents and Function Calling agents. This article aims to provide a comparison of these two methods, complete with code examples to help you make an informed decision for your AI projects. For more reading on AI enginnering, please consult my book at Amazon: https://www.amazon.com/Practical-Guide-Engineers-Ken-Huang/dp/B0D4KP1B2P

Understanding ReAct Agents

ReAct (Reasoning and Acting) agents represent a sophisticated approach to AI implementation that emphasizes a structured reasoning process.

Key Features of ReAct Agents:

1. **Reasoning Process**: ReAct agents utilize a carefully crafted prompt to guide the Language Model (LLM) through a loop of Thought, Action, and Observation steps.

2. **Tool Usage**: ReAct agents can use multiple tools within a single interaction.

3. **Customization**: Developers can fine-tune the reasoning process through prompt engineering.

4. **LLM Compatibility**: These agents can work with a wide range of LLMs.

Code Example for ReAct Agent:

```
from llama_index.agent import ReActAgent

from llama_index.tools import FunctionTool

# Define tools

def search_web(query: str) -> str:

 # Implement web search logic

 return f"Search results for: {query}"

def calculate(expression: str) -> str:

 # Implement calculation logic

 return f"Result of {expression}"

tools = [

 FunctionTool.from_defaults(fn=search_web),

 FunctionTool.from_defaults(fn=calculate),

]

# Create ReAct agent

react_agent = ReActAgent.from_tools(tools)

# Use the agent

response = react_agent.chat("What's the population of France plus the population of Germany?")

print(response)
```

In this example, the ReAct agent can use both a web search tool and a calculation tool to answer complex queries.

Exploring Function Calling Agents

Function Calling agents leverage the built-in capabilities of certain advanced LLMs to streamline the process of tool selection and usage.

Key Features of Function Calling Agents:

1. **Functionality**: These agents rely on the LLM's inherent ability to select and use predefined functions.

2. **Tool Selection**: Typically use a single tool or function per user prompt.

3. **Implementation**: Require LLMs with specific function calling capabilities.

4. **Efficiency**: Often more straightforward to implement for supported tasks.

Code Example for Function Calling Agent:

```
from llama_index.agent import OpenAIAgent

from llama_index.tools import FunctionTool

# Define a tool

def get_weather(location: str) -> str:

 # Implement weather fetching logic

 return f"The weather in {location} is sunny."

weather_tool = FunctionTool.from_defaults(

 name="get_weather",

 description="Get the current weather for a location",

 fn=get_weather

)

# Create Function Calling agent

function_agent = OpenAIAgent.from_tools([weather_tool])

# Use the agent

response = function_agent.chat("What's the weather like in Paris?")

print(response)
```

This example shows a Function Calling agent using a single weather tool to respond to queries.

Making the Right Choice

Choosing between ReAct and Function Calling agents depends on several factors:

1. **Complexity of Tasks**: For complex reasoning scenarios, ReAct agents are often better. For simpler tasks, Function Calling agents may be more efficient.

2. **Customization Needs**: ReAct agents offer more flexibility through prompt engineering. Function Calling agents provide a more standardized

approach.

3. **LLM Compatibility**: Consider the LLMs you're working with. Function Calling agents require specific LLM capabilities.

4. **Development Resources**: ReAct agents may require more initial setup, while Function Calling agents can be quicker to implement for supported LLMs.

Conclusion

Both ReAct and Function Calling agents offer unique advantages in AI implementation. ReAct agents shine in scenarios requiring complex reasoning and flexibility, making them ideal for sophisticated applications. Function Calling agents, with their streamlined approach, excel in efficiency for more straightforward tasks.

Consider this code comparison:

```
# ReAct agent handling a complex query

react_response = react_agent.chat("Find the population of New York and calculate 5% of it.")

# Function Calling agent handling a simple query

function_response = function_agent.chat("What's the weather in Tokyo?")
```

The ReAct agent can handle multi-step tasks involving different tools, while the Function Calling agent excels at direct, single-function queries.

As AI continues to advance, understanding these different approaches becomes crucial. By carefully considering your project's specific needs and the examples provided, you can choose the agent type that best aligns with your goals, ultimately leading to more effective and efficient AI solutions.

For more detail, please see colab examples at

https://colab.research.google.com/drive/1GyPRMiwxS7rKxKpRt4r-ckYfmAw2GxdQ?usp=sharing

And

https://colab.research.google.com/drive/1XYNaGvEdyKVbs4g_Maffyq08DUArcW8H?usp=sharing#scrollTo=h3kycd7CbRDe

Ken Huang, CISSP

AI Book Author |Speaker |DistributedApps.AI |OWASP Top 10 for LLM Co-Author | NIST GenAI Contributor| EC-Council GenAI Security Instructor | CSA Fellow | CSA AI Safety WGs Co-Chair

✓ Following

About
Accessibility
Talent Solutions
Professional Community Policies
Careers
Marketing Solutions
Privacy & Terms
Ad Choices
Advertising
Sales Solutions
Mobile
Small Business
Safety Center

Questions?
Visit our Help Center.

Manage your account and privacy
Go to your Settings.

Recommendation transparency
Learn more about Recommended Content.

Select Language
English (English)

LinkedIn Corporation © 2025