OpenAI tools

Newer OpenAI models have been fine-tuned to detect when **one or more** function(s) should be called and respond with the inputs that should be passed to the function(s). In an API call, you can describe functions and have the model intelligently choose to output a JSON object containing arguments to call these functions. The goal of the OpenAI tools APIs is to more reliably return valid and useful function calls than what can be done using a generic text completion or chat API.

OpenAI termed the capability to invoke a **single** function as **functions**, and the capability to invoke **one or more** functions as **tools**.

In the OpenAI Chat API, **functions** are now considered a legacy options that is deprecated in favor of **tools**.

If you're creating agents using OpenAI models, you should be using this OpenAI Tools agent rather than the OpenAI functions agent.

Using **tools** allows the model to request that more than one function will be called upon when appropriate.

In some situations, this can help signficantly reduce the time that it takes an agent to achieve its goal.

See

- OpenAI chat create
- OpenAI function calling

```
%pip install --upgrade --quiet  langchain-openai tavily-python
```

```
from langchain import hub
from langchain.agents import AgentExecutor, create_openai_tools_agent
from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_openai import ChatOpenAI
```

Initialize Tools

For this agent let's give it the ability to search the web with Tavily.

```
tools = [TavilySearchResults(max_results=1)]
```

Create Agent

```
# Get the prompt to use - you can modify
this!
prompt = hub.pull("hwchase17/openai-tools-
agent")
```

```
# Choose the LLM that will drive the agent
# Only certain models support this
llm = ChatOpenAI(model="gpt-3.5-turbo-1106",
temperature=0)

# Construct the OpenAI Tools agent
agent = create_openai_tools_agent(llm, tools,
prompt)
```

Run Agent

```
# Create an agent executor by passing in the
agent and tools
```

```
agent_executor = AgentExecutor(agent=agent,
tools=tools, verbose=True)
```

```
agent_executor.invoke({"input": "what is
LangChain?"})
```

```



> Entering new AgentExecutor chain...

Invoking: `tavily_search_results_json` with
`{'query': 'LangChain'}`



[{'url':
'https://www.ibm.com/topics/langchain',
'content': 'LangChain is essentially a
library of abstractions for Python and
Javascript, representing common steps and
concepts  LangChain is an open source
orchestration framework for the development
of applications using large language models
other LangChain features, like the eponymous
chains.  LangChain provides integrations for
over 25 different embedding methods, as well
as for over 50 different vector
storesLangChain is a tool for building
applications using large language models
(LLMs) like chatbots and virtual agents. It
simplifies the process of programming and
integration with external data sources and
software workflows. It supports Python and
Javascript languages and supports various LLM
providers, including OpenAI, Google, and
IBM.'}]LangChain is an open source
orchestration framework for the development
of applications using large language models.
It is essentially a library of abstractions
for Python and Javascript, representing
common steps and concepts. LangChain
simplifies the process of programming and
integration with external data sources and
software workflows. It supports various large
language model providers, including OpenAI,
Google, and IBM. You can find more
information about LangChain on the IBM
website: [LangChain - IBM]
(https://www.ibm.com/topics/langchain)

> Finished chain.
```

```
{'input': 'what is LangChain?',
 'output': 'LangChain is an open source
orchestration framework for the development
of applications using large language models.
It is essentially a library of abstractions
for Python and Javascript, representing
common steps and concepts. LangChain
simplifies the process of programming and
integration with external data sources and
software workflows. It supports various large
language model providers, including OpenAI,
Google, and IBM. You can find more
information about LangChain on the IBM
website: [LangChain - IBM]
(https://www.ibm.com/topics/langchain)'}
```

Using with chat history

```
from langchain_core.messages import
AIMessage, HumanMessage

agent_executor.invoke(
    {
        "input": "what's my name? Don't use
tools to look this up unless you NEED to",
        "chat_history": [
            HumanMessage(content="hi! my name
is bob"),
            AIMessage(content="Hello Bob! How
can I assist you today?"),
        ],
    }
)
```

```


> Entering new AgentExecutor chain...
Your name is Bob.

> Finished chain.
```

```
{'input': "what's my name? Don't use tools to look this up unless you NEED to",
 'chat_history': [HumanMessage(content='hi! my name is bob'),
  AIMessage(content='Hello Bob! How can I assist you today?')],
 'output': 'Your name is Bob.'}
```