OpenAI functions

Certain OpenAI models (like gpt-3.5-turbo-0613 and gpt-4-0613) have been fine-tuned to detect when a function should be called and respond with the inputs that should be passed to the function. In an API call, you can describe functions and have the model intelligently choose to output a JSON object containing arguments to call those functions. The goal of the OpenAI Function APIs is to more reliably return valid and useful function calls than a generic text completion or chat API.

A number of open source models have adopted the same format for function calls and have also fine-tuned the model to detect when a function should be called.

The OpenAI Functions Agent is designed to work with these models.

Install `openai`, `tavily-python` packages which are required as the LangChain packages call them internally.

OpenAI API has deprecated `functions` in favor of `tools`. The difference between the two is that the `tools` API allows

the model to request that multiple functions be invoked at once, which can reduce response times in some architectures. It's recommended to use the tools agent for OpenAI models.

See the following links for more information:

OpenAI chat create

OpenAI function calling

The `functions` format remains relevant for open source models and providers that have adopted it, and this agent is expected to work for such models.

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

Initialize Tools

We will first create some tools we can use

```
from langchain import hub
from langchain.agents import AgentExecutor,
create_openai_functions_agent
from langchain_community.tools.tavily_search
```

```
import TavilySearchResults
from langchain_openai import ChatOpenAI
```

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

Create Agent

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

```
prompt.messages
```

```
[SystemMessagePromptTemplate(prompt=PromptTempl
[], template='You are a helpful assistant')),
 MessagesPlaceholder(variable_name='chat_histor
 HumanMessagePromptTemplate(prompt=PromptTempla
['input'], template='{input}')),
 MessagesPlaceholder(variable_name='agent_scrat
```

```
# Choose the LLM that will drive the agent
llm = ChatOpenAI(model="gpt-3.5-turbo-1106")
```

```
# Construct the OpenAI Functions agent
agent = create_openai_functions_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 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. LangChain provides integrations for over 25 different embedding methods and for over 50 different vector stores. It is essentially a library of abstractions for Python and JavaScript, representing common steps and concepts. LangChain supports Python and JavaScript languages and various LLM providers, including OpenAI, Google, and IBM. You can

```
find more information about LangChain [here]
(https://www.ibm.com/topics/langchain).

> Finished chain.
```

```
{'input': 'what is LangChain?',
 'output': 'LangChain 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. LangChain provides
integrations for over 25 different embedding
methods and for over 50 different vector
stores. It is essentially a library of
abstractions for Python and JavaScript,
representing common steps and concepts.
LangChain supports Python and JavaScript
languages and various LLM providers,
including OpenAI, Google, and IBM. You can
find more information about LangChain [here]
(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?",
        "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?",
 'chat_history': [HumanMessage(content='hi!
my name is bob'),
  AIMessage(content='Hello Bob! How can I
assist you today?')],
 'output': 'Your name is Bob.'}
```