Function Calling with OpenAI APIs

```
import os
import openai
import json
from dotenv import load_dotenv

load_dotenv()

# set openai api key
openai.api_key = os.environ['OPENAI_API_KEY']
```

Define a Get Completion Function

```
def get_completion(messages, model="gpt-3.5-turbo-1106", temperature=0, max_tokens=300, tools=None, tool_choice=None):
    response = openai.chat.completions.create(
        model=model,
        messages=messages,
        temperature=temperature,
        max_tokens=max_tokens,
        tools=tools,
        tool_choice=tool_choice
    )
    return response.choices[0].message
```

Define Dummy Function

```
# Defines a dummy function to get the current weather
def get_current_weather(location, unit="fahrenheit"):
    """Get the current weather in a given location"""
    weather = {
        "location": location,
        "temperature": "50",
        "unit": unit,
    }

    return json.dumps(weather)
```

Define Functions

As demonstrated in the OpenAI documentation, here is a simple example of how to define the functions that are going to be part of the request.

The descriptions are important because these are passed directly to the LLM and the LLM will use the description to determine whether to use the functions or how to use/call.

```
# define a function as tools
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_current_weather",
            "description": "Get the current weather in a given location",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The city and state, e.g. San Francisco, CA",
                    },
                    "unit": {
                        "type": "string",
                        "enum": ["celsius", "fahrenheit"]},
                },
                "required": ["location"],
            },
        },
    }
]
```

```
# define a list of messages

messages = [
    {
        "role": "user",
        "content": "What is the weather like in London?"
    }
]


response = get_completion(messages, tools=tools)
print(response)
```

```
ChatCompletionMessage(content=None, role='assistant', function_call=None, tool_calls=[ChatCompletionMessageToolCall(id='call_GYg3yhS
```

response.tool_calls[0].function.arguments

We can now capture the arguments:

```
args = json.loads(response.tool_calls[0].function.arguments)


get_current_weather(**args)
```

```
'{"location": "London", "temperature": "50", "unit": "celsius"}'
```

Controlling Function Calling Behavior

Let's say we were interested in designing this `function_calling` functionality in the context of an LLM-powered conversational agent. Your solution should then know what function to call or if it needs to be called at all. Let's try a simple example of a greeting message:

```
messages = [
    {
        "role": "user",
        "content": "Hello! How are you?",
    }
]


get_completion(messages, tools=tools)
```

```
ChatCompletionMessage(content="Hello! I'm here and ready to assist you. How can I help you today?", role='assistant',
function_call=None, tool_calls=None)
```

You can specify the behavior you want from function calling, which is desired to control the behavior of your system. By default, the model decide on its own whether to call a function and which function to call. This is achieved by setting `tool_choice: "auto"` which is the default setting.

```
get_completion(messages, tools=tools, tool_choice="auto")
```

```
ChatCompletionMessage(content="Hello! I'm here and ready to assist you. How can I help you today?", role='assistant',
function_call=None, tool_calls=None)
```

Setting `tool_choice: "none"` forces the model to not use any of the functions provided.

```
get_completion(messages, tools=tools, tool_choice="none")
```

```
ChatCompletionMessage(content="Hello! I'm here and ready to assist you. How can I help you today?", role='assistant',
function_call=None, tool_calls=None)
```

```
messages = [
    {
        "role": "user",
        "content": "What's the weather like in London?",
    }
]
get_completion(messages, tools=tools, tool_choice="none")
```

```
ChatCompletionMessage(content='I will check the current weather in London for you.', role='assistant', function_call=None,
tool_calls=None)
```

You can also force the model to choose a function if that's the behavior you want in your application. Example:

```
messages = [
    {
        "role": "user",
        "content": "What's the weather like in London?",
    }
]
get_completion(messages, tools=tools, tool_choice={"type": "function", "function": {"name": "get_current_weather"}})
```

```
ChatCompletionMessage(content=None, role='assistant', function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_ctUCZtwxZYhfF3sirByNY2qC',
function=Function(arguments='{"location":"London","unit":"celsius"}', name='get_current_weather'), type='function')])
```

The OpenAI APIs also support parallel function calling that can call multiple functions in one turn.

```
messages = [
    {
        "role": "user",
        "content": "What's the weather like in London and Belmopan in the coming days?",
    }
]
get_completion(messages, tools=tools)
```

```
ChatCompletionMessage(content=None, role='assistant', function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_GhyrfrY9QWkrJs9vKqxpGW22', function=Function(arguments='{"location": "London", "unit":
"celsius"}', name='get_current_weather'), type='function'), ChatCompletionMessageToolCall(id='call_Cl9fqGh1AZgi4yD8n5Lrc6NE',
function=Function(arguments='{"location": "Belmopan", "unit": "celsius"}', name='get_current_weather'), type='function')])
```

You can see in the response above that the response contains information from the function calls for the two locations queried.

Function Calling Response for Model Feedback

You might also be interested in developing an agent that passes back the result obtained after calling your APIs with the inputs generated from function calling. Let's look at an example next:

```
messages = []
messages.append({"role": "user", "content": "What's the weather like in Boston!"})
assistant_message = get_completion(messages, tools=tools, tool_choice="auto")
assistant_message = json.loads(assistant_message.model_dump_json())
assistant_message["content"] = str(assistant_message["tool_calls"][0]["function"])

#a temporary patch but this should be handled differently
# remove "function_call" from assistant message
del assistant_message["function_call"]


messages.append(assistant_message)
```

We then append the results of the `get_current_weather` function and pass it back to the model using a `tool` role.

```
# get the weather information to pass back to the model
weather = get_current_weather(messages[1]["tool_calls"][0]["function"]["arguments"])

messages.append({"role": "tool",
                 "tool_call_id": assistant_message["tool_calls"][0]["id"],
                 "name": assistant_message["tool_calls"][0]["function"]["name"],
                 "content": weather})


final_response = get_completion(messages, tools=tools)


final_response
```

```
ChatCompletionMessage(content='The current temperature in Boston, MA is 50°F.', role='assistant', function_call=None,
tool_calls=None)
```