Using function calling with Azure OpenAI Service

- Function calling is useful when you are building an application that bridges the models and functionality of your application. For example, you can give the model access to functions that query a database in order to build an AI assistant that can help users with their orders, or functions that can interact with the UI.
- The latest versions of gpt-35-turbo and gpt-4 are fine-tuned to work with functions and are able to both determine when and how a function should be called.
- If one or more functions are included in your request, the model determines if any of the functions should be called based on the context of the prompt.
- When the model determines that a function should be called, it responds with a JSON object including the arguments for the function.

At a high level you can break down working with functions into three steps:

1. Call the chat completions API with your functions and the user query
2. Use the model's response to call your API or function.
3. Call the chat completions API again, including the response from your function to get a final response.

Note: If the model decides that no function should be called, then the response will contain a direct reply to the user as a regular chat completion response.

Single tool/function calling example

```
import requests
from util import getOpenAIClient
import json

# Get OpenAI Client from Util.
client = getOpenAIClient()

# Replace with your API keys
news_api_key = "c86245eca72141828e351e1ea1e92fae"

# Define the `get_news` function to retrieve news articles based on a given topic
def get_news(topic):
    print(f"In get news about {topic}")
```

```
url = (
    f"https://newsapi.org/v2/everything?q={topic}&apiKey={news_api_key}&pageSize=5"
)
try:
    response = requests.get(url)
    if response.status_code  == 200:
        news = json.dumps(response.json(),  indent=4)
        news_json = json.loads(news)

        # Access all the fields == loop through
        status = news_json["status"]
        total_results = news_json["totalResults"]
        articles = news_json["articles"]
        final_news = []

        # Loop through articles
        for article in articles:
            source_name = article["source"]["name"]
            author = article["author"]
            title = article["title"]
            description = article["description"]
            url = article["url"]
            content = article["content"]
            title_description = f"""
              Title: {title},
              Author: {author},
              Source: {source_name},
              Description: {description},
              URL: {url},
              Content: {content}
            """
            final_news.append(title_description)
        return final_news
    else:
        return []
```

```
    except requests.exceptions.RequestException as e:
        return f"Error occurred during API Request: {e}"

# Define the function schema for OpenAI API
get_news_function_schema = {
    "name": "get_news",
    "description": "Retrieve the latest news articles on a given topic.",
    "parameters": {
        "type": "object",
        "properties": {
            "topic": {
                "type": "string",
                "description": "The topic to search news articles for.",
            }
        },
        "required": ["topic"],
    },
}

# Initial user message
#messages = [{"role": "user", "content": "Based on latest news of AI, generate a brief summary and suggest potential areas for further exploration or discussion"}] # Single function call
messages = [{"role": "user", "content": "Show me the latest news about AI and Stocks."}] # Two Parallel function calls with a single tool/function defined

# First API call: Ask the model to use the function
response = client.chat.completions.create(
    model="gpt-4o-2024-08-06",
    messages=messages,
    tools = [
        {
            "type": "function",
            "function": get_news_function_schema
        }
```

```
    ],
    tool_choice="auto",  # Let the model decide if the function should be called
)
# Process the model's response
response_message = response.choices[0].message
messages.append(response_message)

print("Model's response:")
print(response_message)

# If the model suggests calling the function
if response_message.tool_calls:
    for tool_call in response_message.tool_calls:
        function_name = tool_call.function.name
        function_args =json.loads(tool_call.function.arguments)
        if function_name == "get_news":
            # Call the function with arguments
            topic = function_args.get("topic")
            news_result = get_news(topic)

            # Format the results into a single string
            formatted_news = "\n".join(news_result)
            messages.append({
                "tool_call_id": tool_call.id,
                "role": "tool",
                "name": "get_news",
                "content": formatted_news,
            })
else:
    print("No tool calls were made by the model.")

# Second API call: Get the final response from the model
response = client.chat.completions.create(
    model="gpt-4o-2024-08-06",
    messages = messages,
```

```
)


# Output the response
print("Response:", response.choices[0].message.content)
```

Parallel function calling with multiple functions

Now we will demonstrate another toy function calling example this time with two different tools/functions defined.

```
pip install tzdata
```

```
import json
from dotenv import load_dotenv
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo
from util import GetOpenAIClient

# Get OpenAI Client from Util.
client = GetOpenAIClient()

# Simplified weather data
WEATHER_DATA = {
    "tokyo": {"temperature": "10", "unit": "celsius"},
    "san francisco": {"temperature": "72", "unit": "fahrenheit"},
    "paris": {"temperature": "22", "unit": "celsius"}
}

# Simplified timezone data
TIMEZONE_DATA = {
    "tokyo": "Asia/Tokyo",
    "san francisco": "America/Los_Angeles",
    "paris": "Europe/Paris"
}
```

```
def get_current_time(location):
    """Get the current time for a given location"""
    print(f"get_current_time called with location: {location}")
    location_lower = location.lower()
    for key, timezone in TIMEZONE_DATA.items():
        if key in location_lower:
            print(f"Timezone found for {key}")
            current_time = datetime.now(ZoneInfo(timezone)).strftime("%I:%M %p")
            return json.dumps({
                "location": location,
                "current_time": current_time
            })
    print(f"No timezone data found for {location_lower}")
    return json.dumps({"location": location, "current_time": "unknown"})

def get_current_weather(location, unit=None):
    """Get the current weather for a given location"""
    print(f"get_current_weather called with location: {location}, unit: {unit}")
    location_lower = location.lower()
    for key in WEATHER_DATA:
        if key in location_lower:
            print(f"Weather data found for {key}")
            weather = WEATHER_DATA[key]
            return json.dumps({
                "location": location,
                "temperature": weather["temperature"],
                "unit": unit if unit else weather["unit"]
            })

    print(f"No weather data found for {location_lower}")
    return json.dumps({"location": location, "temperature": "unknown"})

def run_conversation():
    # Initial user message
    #messages = [{"role": "user", "content": "What's the current time in San Francisco"}] # Single function call
```

```
  messages = [{"role": "user", "content": "What's the weather and current time in San Francisco, Tokyo, and Paris?"}]

  # Define the function for the model
# Define the functions for the model
  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 name, e.g. San Francisco",
            },
            "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]},
          },
          "required": ["location"],
        },
      }
    },
    {
      "type": "function",
      "function": {
        "name": "get_current_time",
        "description": "Get the current time in a given location",
        "parameters": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "The city name, e.g. San Francisco",
```

```
                },
            },
            "required": ["location"],
        },
    }
}
]

# First API call: Ask the model to use the function
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    tools=tools,
    tool_choice="auto",
)

# Process the model's response
response_message = response.choices[0].message
messages.append(response_message)

print("Model's response:")
print(response_message)

# Handle function calls
if response_message.tool_calls:
    for tool_call in response_message.tool_calls:
        function_name = tool_call.function.name
        function_args = json.loads(tool_call.function.arguments)
        print(f"Function call: {function_name}")
        print(f"Function arguments: {function_args}")

        if function_name == "get_current_weather":
            function_response = get_current_weather(
                location=function_args.get("location"),
                unit=function_args.get("unit")
```

```
            )
        elif function_name == "get_current_time":
            function_response = get_current_time(
                location=function_args.get("location")
            )
        else:
            function_response = json.dumps({"error": "Unknown function"})

        messages.append({
            "tool_call_id": tool_call.id,
            "role": "tool",
            "name": function_name,
            "content": function_response,
        })
    else:
        print("No tool calls were made by the model.")

    # Second API call: Get the final response from the model
    final_response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=messages,
    )
    return final_response.choices[0].message.content

# Run the conversation and print the result
print(run_conversation())
```