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

import json

from util import generateToken

from openai import OpenAI

import os

generateToken()

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

# 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 datetime import datetime, timedelta

from zoneinfo import ZoneInfo

from util import generateToken

from openai import OpenAI

import os

generateToken()

header\_name = os.getenv('GATEWAY\_HEADER\_NAME')

header\_value = os.getenv('GATEWAY\_HEADER\_VALUE')

headers = {

header\_name: header\_value

}

client = OpenAI(default\_headers=headers)

# 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())