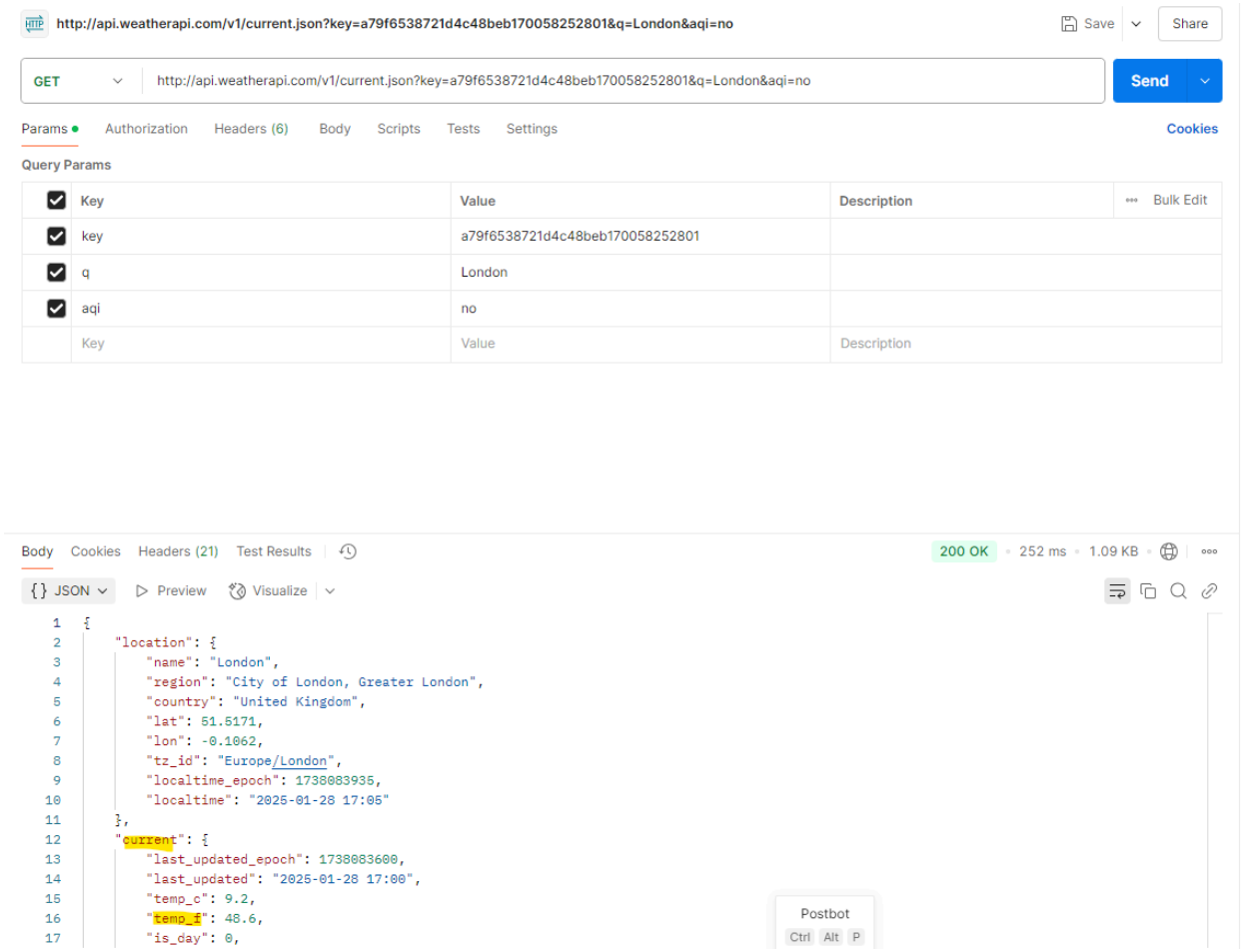
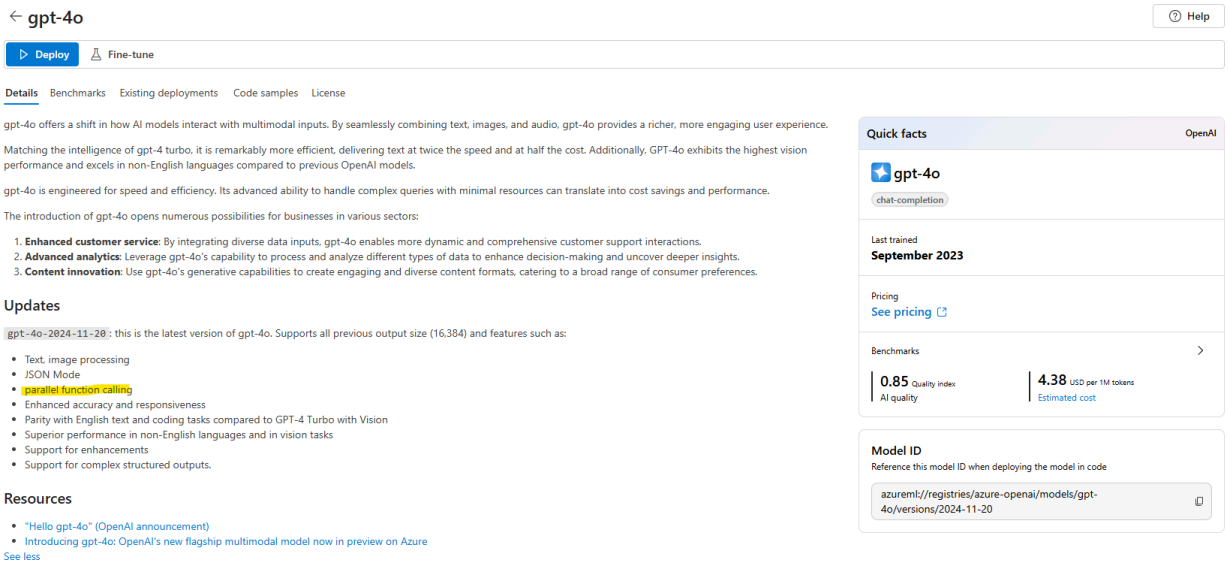
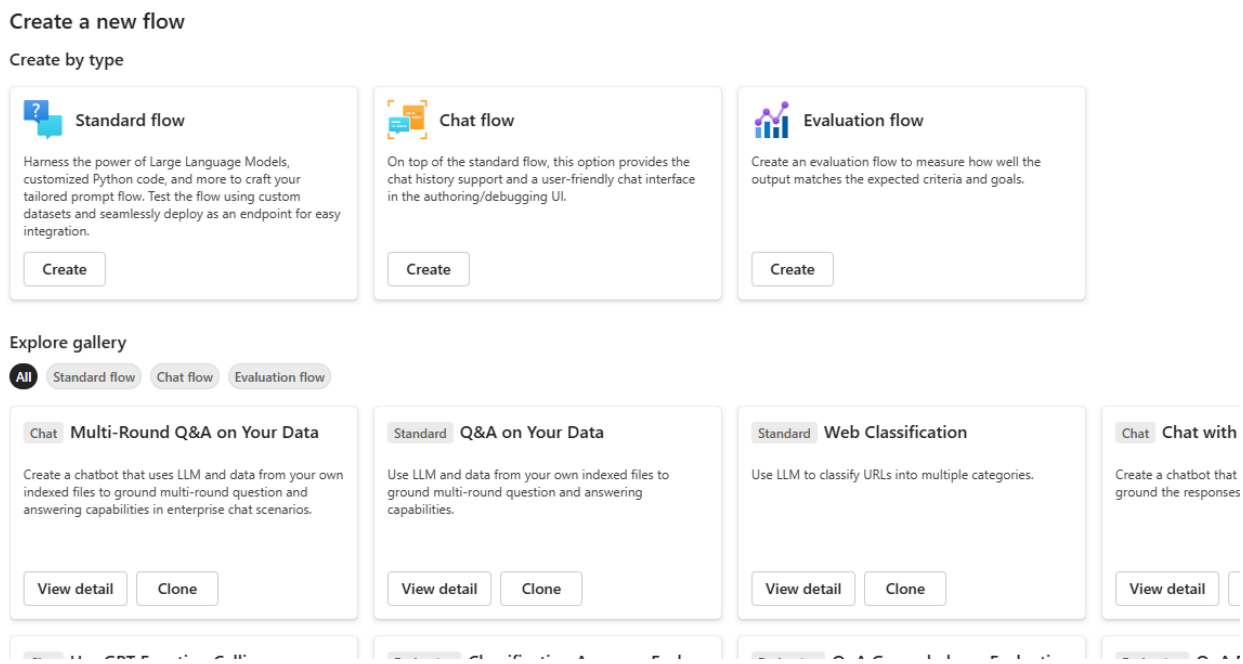
Test the api you plan on using. Note the path of the response you wish to capture. In this case it is current.temp_f



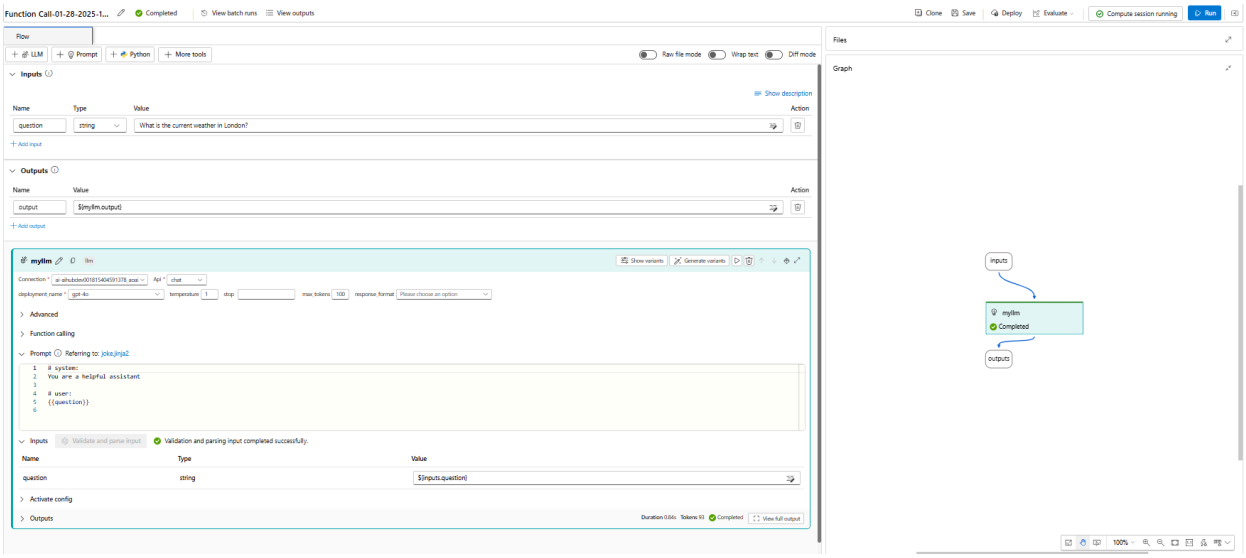
Deploy a model that allow function calling.



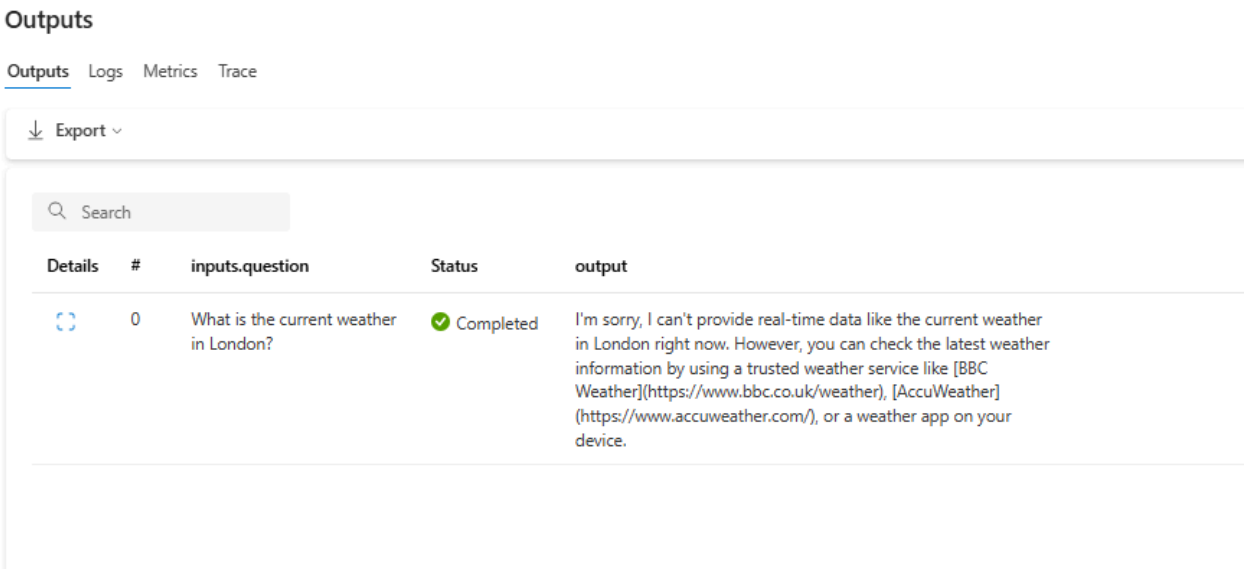
Within Prompt flow create a chat flow



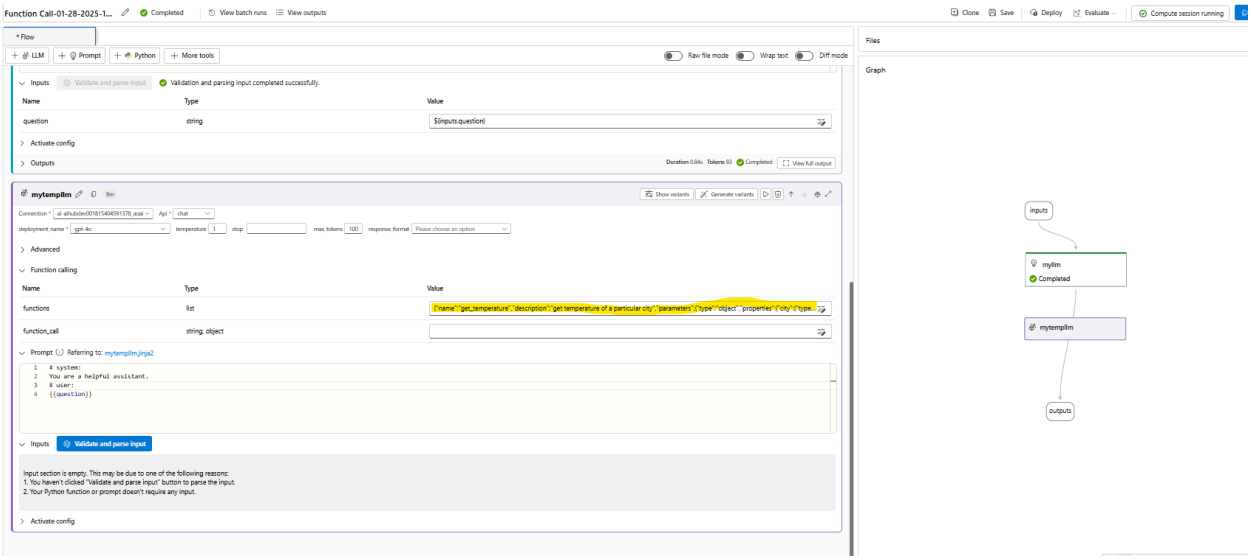
Erase every node but input and output and start the computer sessions. Once started add a prompt. Define the connection and model. Then validate. Configuration should look like below.



Once completed, run the flow without function calling and this should be the output. Note the llm couldn't help with the question.



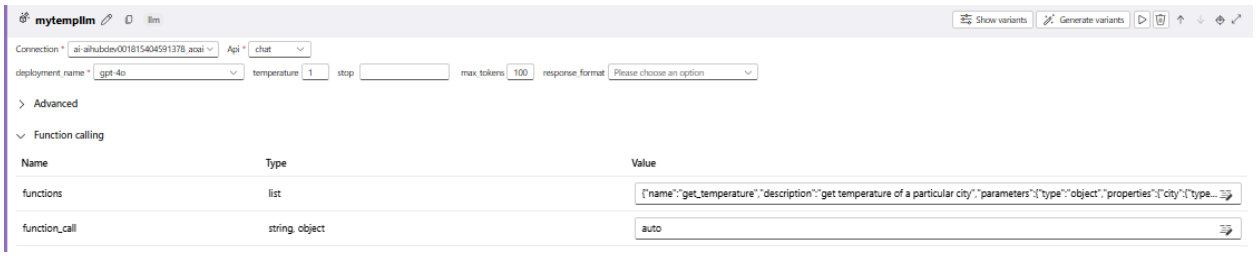
Add another llm and add the following json below to function block. This json defines the schema for the function. The description is of critical importance.



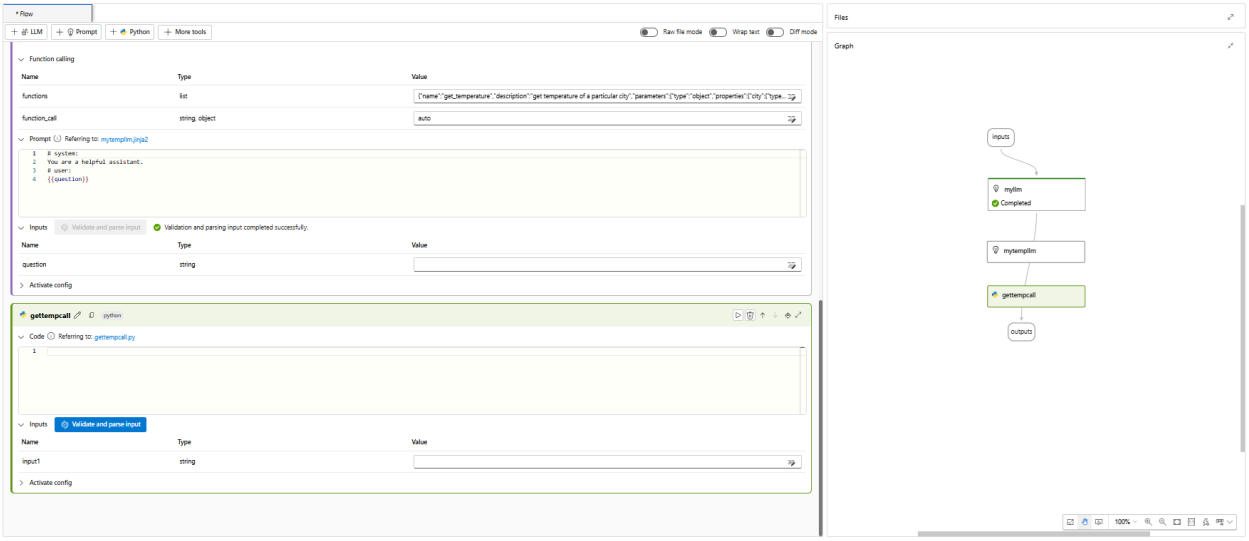
JSON Function Schema. Note the json is inclosed in an list

```
[{
"name":"get_temperature",
"description": "get temperature of a particular city",
"parameters": {
        "type": "object",
        "properties": {
                "city":{
                        "type": "string",
                        "description": "The city e.g. Orlando"
                }
        }
        ,
 "required": ["city"]
}}]
```

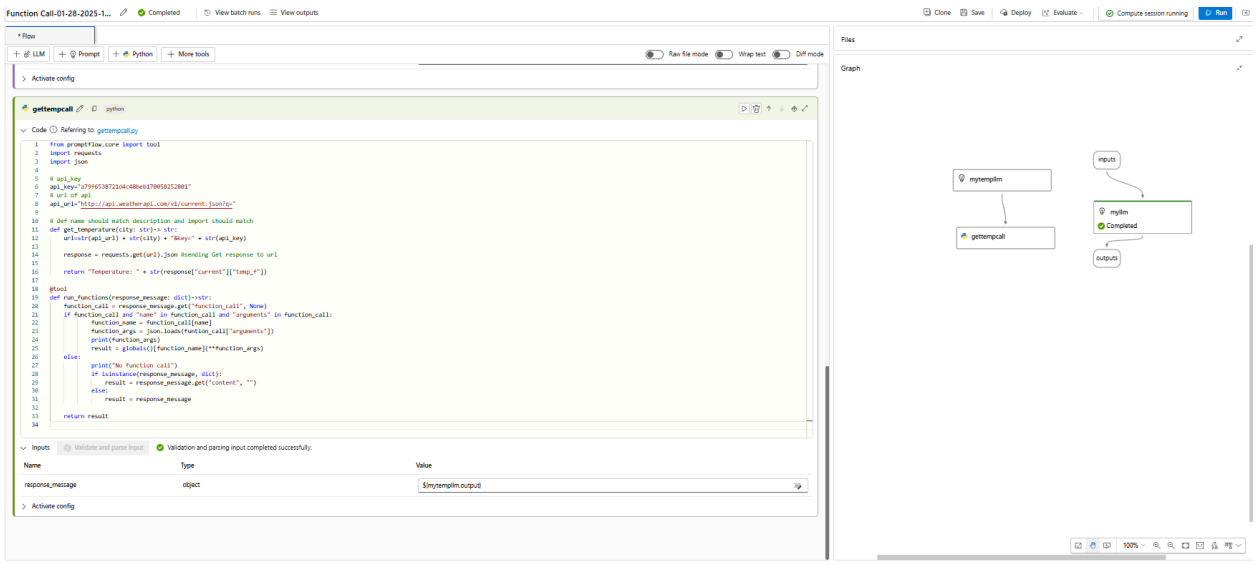
For the function call field add auto. This tells the llm to decide if and when to use the tool.



Within the flow add a custom python node and delete the content.



Add the snippet of python below to the code block and validate. Once valid update response_message to mytempllm.output

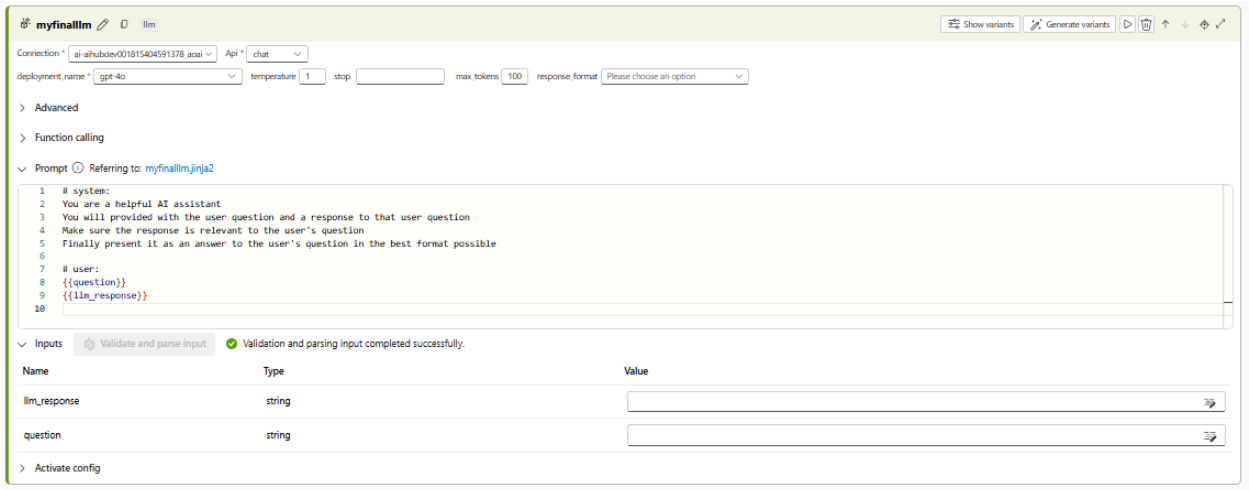


Python snippet. Update with your api key

```
from promptflow.core import tool
import requests
import json

# api_key
```

```
api_key="xxxxxxxxxxxxxxxxx"
# url of api
api_url="http://api.weatherapi.com/v1/current.json?q="

# def name should match description and import should match
def get_temperature(city: str)-> str:
	url=str(api_url) + str(city) + "&key=" + str(api_key)

	response = requests.get(url).json() #sending Get response to url

	return "Temperature: " + str(response["current"]["temp_f"])

@tool
def run_functions(response_message: dict)->str:
	function_call = response_message.get("function_call", None)
	if function_call and "name" in function_call and "arguments" in function_call:
			function_name = function_call["name"]
			function_args = json.loads(function_call["arguments"])
			print(function_args)
			result = globals()[function_name](**function_args)
	else:
			print("No function call")
			if isinstance(response_message, dict):
				result = response_message.get("content", "")
			else:
				result = response_message

	return result
```

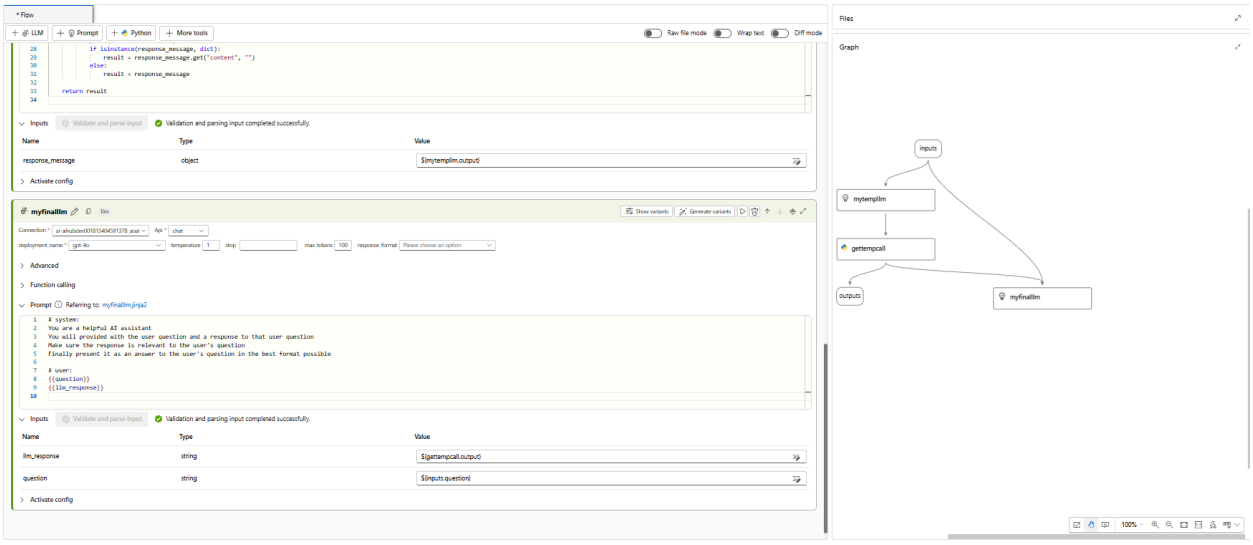
Add another llm to parse the response from the function call and compare to the original question. Use the snippet below for the prompt.



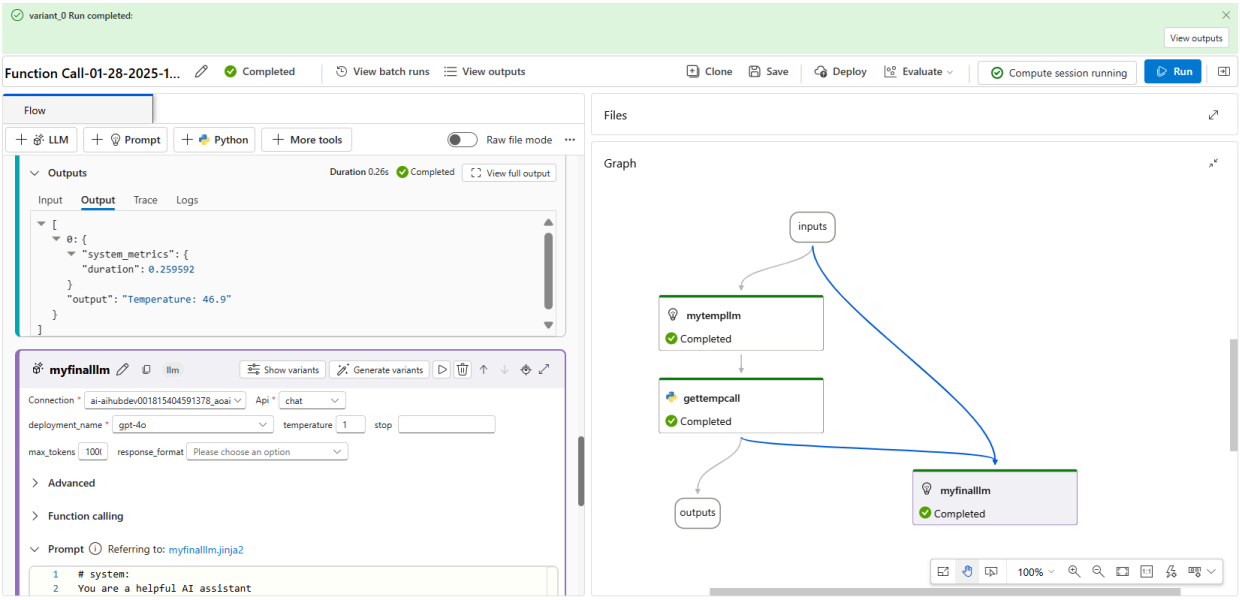
system:
You are a helpful AI assistant
You will provided with the user question and a response to that user question
Make sure the response is relevant to the user's question
Finally present it as an answer to the user's question in the best format possible

user:
{{question}}
{{llm_response}}

Configure the last llm to receive the original question and the output from the function call



Run and see if successful



If successful you can see the question and output