Function Calling

Tool Calling

Tool 사용

- Query: 지금 서울 날씨를 알려줘요
- LLM은 일반적인 서울 날씨 정보를 가지고 있지만 지금 날씨는 알 수 없기 때문에 LLM 외부 Tool을 사용해서 정보를 얻어야 한다.

- 질문 1: 사용자가 직접 외부 정보를 얻으면 되지 굳이 LLM을 사용해야 할까? 일반적으로 외부 정보는 자연어 질의·응답을 지원하지 않는다.
- 질문 2: LLM이 Tool을 직접 사용하는가? LLM은 자연어 질의로부터 Tool 사용에 필요한 정보(function-name, arguments)를 추출한다.
- 질문 3: 누가 function call을 수행하는가? 개발자가 직접 function call을 수행하고 반환된 결과를 이전 질문과 합해서 다시 질의하여야 한다.
- 질문 4: 왜 function call 반환 결과를 처음 질문과 합쳐서 다시 LLM에 질의하는가? (처음 질문 + function call result)로 질의하면 LLM이 추가 정보를 이용하여 자연어 응답을 할 수 있기 때문이다.

Basic Tool Calling

1

기초 Tool 호출 순서

- 최근에는 tool calling으로 통일(예전에는 function calling)

① LLM 선택 – OpneAI로 하는 것이 API 사용 용이

② Tool 정의 – Tool은 function의 리스트

③ Message 작성 – LLM 형식에 맞추어 Query 작성

④ LLM 1차 호출 – 정의된 Message와 Tool을 묶어서 LLM API 호출

⑤ Tool 실행 – LLM에서 반환된 function 이름과 arguments를 이용하여 tool 실행

⑥ Message 작성 – 실행된 Tool 결과로부터 새 메시지 작성

⑦ LLM 2차 호출 – 새롭게 작성된 Message와 정의된 Tool을 묶어서 LLM API 재호출

⑧ 결과 반환 – LLM에서 나온 출력물 반환

① LLM 클라이언트 초기화

- `client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))`

② Tool 정의

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_weather",
            "description": "도시의 현재 날씨",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {"type": "string"}
                },
                "required": ["city"]
            }
        }
    }
]
```

③ 메시지 작성

```
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "서울의 날씨를 알려 주세요"}
]
```

④ LLM 1차 호출

```
tool_response = client.chat.completions.create(
    model="gpt-5-mini",
    messages=messages,
    tools=tools,
)
print(tool_response.choices[0].message.tool_calls)
[
    ChatCompletionMessageFunctionToolCall(
        id='call_j09jyzHLaghmMuysxM7ghBfp', # 2차 메시지 작성시 들어가야 함
        function=Function(arguments='{"city":"서울"}', name='get_weather'),
        type='function')
]
```

OpneAI 응답 메시지 형식

```
print(tool_response.choices[0].message)

ChatCompletionMessage(
    content=None,
    refusal=None,
    role='assistant’,
    annotations=[],
    audio=None,
    function_call=None
    tool_calls=[
        ChatCompletionMessageFunctionToolCall(
            id='call_j09jyzHLaghmMuysxM7ghBfp’,
            function=Function(arguments='{"city":"서울"}', name='get_weather’),
            type='function’
        )
    ]
)
```

⑤ Tool 실행

```
tool_name = tool_calls[0].function.name
tool_args = json.loads(tool_calls[0].function.arguments) # json 형식 str → dict


if tool_name == "get_weather":
    city = tool_args.get("city")
    # 여기서 실제 날씨 정보를 가져오는 로직을 구현해야함.
    weather_info = f"{city}의 현재 날씨는 맑음, 기온은 25도입니다."
```

⑥ 최종 메시지 작성

```
final_messages =
    # 1) 처음 메시지
    messages +
    [
        # 2) LLM 1차 응답
        tool_response.choices[0].message,

        # 3) Tool 실행 결과
        {
            "role": "tool",
            "name": tool_name,
            "content": weather_info,
            "tool_call_id": tool_response.choices[0].message.tool_calls[0].id
        }
    ]
```

⑦ LLM 최종 호출 및 응답 출력

```
final_response = client.chat.completions.create(
    model="gpt-5-mini",
    messages=final_messages,
)

print(final_response.choices[0].message)
```

ChatCompletionMessage(content='지금 서울은 맑고 기온은 25°C입니다. \n시간대별 예보, 주간 예보나 미세먼지 정보도 원하시면 알려 주세요.', refusal=None, role='assistant', annotations=[], audio=None, function_call=None, tool_calls=None)

2

Multiple Tool 호출

Multiple Tool 호출 순서

- Basic Tool Calling과 기본 흐름 위에

(a) tool calling이 응답이 오지 않을 경우 대비하여 looping 구조로 구현

(b) multiple tool 호출 구조 이해하고 LLM 메시지 작성

Multiple Tools 정의

```
tools = []


function1 = {
        "type": "function",
        "function": {
            "name": "get_weather",
            "description": "도시의 현재 날씨",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {"type": "string"}
                },
                "required": ["city"]
            }
        }
    }
```

```
function2 = {
        "type": "function",
        "function": {
            "name": "get_time",
            "description": "도시의 현재 시간을 문자열로 반환",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {"type": "string",},
                },
                "required": [],
            }
        }
    }


tools.append(function1)
tools.append(function2)
```

Tool 실행기 구현

```
# 3) 두 개의 Tool 구현
def tool_get_weather_sync(city: str, unit: str = "c"):
    temp = 20 if unit == "c" else 68
    return {"city": city, "temperature": f"{temp}°{'C' if unit=='c' else 'F'}"}


def tool_get_time_sync(city: str = "Seoul"):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    return {"city": city, "time": now}


# 간단한 도구 실행기
def run_tool_demo(name: str, arguments: dict):
    if name == "get_weather":
        return tool_get_weather_sync(**arguments)
    if name == "get_time":
        return tool_get_time_sync(**arguments)
    return {"error": f"unknown tool {name}"}
```

Calling Loop

```
def ask_inline_dual(messages: list, max_cycles: int = 3) -> str:

    for _ in range(max_cycles):
        # LLM 호출
        response = client.chat.completions.create(
            model="gpt-5-mini",
            messages=messages,
            tools=tools,
        )

        msg = response.choices[0].message
        tool_calls = msg.tool_calls
        if not tool_calls:
            # 더 이상 툴 호출이 없으면 최종 답변
            return msg.content or ""

        # 메시지에 assistant의 tool_calls 추가
        messages.append(msg)
```

Calling Loop

```
        # 각 tool_call 실행 → tool 실행 결과 메시지에 추가
        for tc in tool_calls:
            args = json.loads(tc.function.arguments or "{}")
            result = run_tool_demo(tc.function.name, args)
            messages.append({
                "role": "tool",
                "tool_call_id": tc.id,  # 필수: 어떤 호출의 결과인지 연결
                "content": json.dumps(result),
            })

    return "[max_cycles reached]"
```

message.tool_calls

```
[
    ChatCompletionMessageFunctionToolCall(id='call_k6VhUFp2elnGlRbdb3yWP6xP’,
    function=Function(arguments='{"city": "Seoul"}', name='get_weather'), type='function’),

    ChatCompletionMessageFunctionToolCall(id='call_t74wmfbWUTIvBWi3ljCnjW68’,
    function=Function(arguments='{"city": "Seoul"}', name='get_time'), type='function’)
]
```

메시지 작성

```
user_prompt = "서울의 현재 날씨와 시간을 알려줘요."
messages = [
        {"role": "system", "content": "You can use tools (get_weather, get_time) as
  needed."},
        {"role": "user", "content": user_prompt},
]

response = ask_inline_dual(messages)
print(response)
```