Function Calling and Tool Calling in Agentic AI

Humaiza Naz

May 30, 2025

1 Introduction

This document provides a detailed explanation of **Function Calling** and **Tool Calling** in the context of Agentic AI, focusing on their implementation in the OpenAI Assistants API. These mechanisms enable AI to interact with external systems, making it a powerful tool for real-world applications like e-commerce, travel planning, and more.

2 What is Function Calling?

Function Calling is a mechanism where an AI model identifies a user's intent and triggers a specific backend function to fulfill it. It is designed for simple, single-purpose tasks where the AI calls one or two predefined functions.

- **Definition**: The AI analyzes user input, selects a registered function, and provides the necessary arguments in a structured format (e.g., JSON).
- **Key Features**:
  - Limited to a small set of functions.
  - Minimal metadata (function name and parameters).
  - Suitable for straightforward tasks like fetching weather or performing calculations.
- **Real-Life Example**: In an e-commerce app, a user asks, "What's the price of this item in PKR?" The AI calls a `convert_currency` function to convert USD to PKR and responds with the result.

2.1 Code Example: Function Calling

Below is a Python snippet using the OpenAI SDK to implement Function Calling for a weather query.

```
import openai

client = openai.Client(api_key="your-api-key")

# Define the function schema
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_weather",
            "description": "Get current weather for a city",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {"type": "string", "description": "City
                       name"}
                },
                "required": ["city"]
            }
        }
    }
]

# Create a thread and message
thread = client.beta.threads.create()
message = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="What's the weather in Lahore today?"
)

# Run the assistant
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id="your-assistant-id",
    tools=tools
)

# Check if function call is required
if run.status == "requires_action":
    tool_calls = run.required_action.submit_tool_outputs.tool_calls
    for tool_call in tool_calls:
        if tool_call.function.name == "get_weather":
            # Simulate function execution
            output = {"weather": "Sunny, 32°C"}
            client.beta.threads.runs.submit_tool_outputs(
                thread_id=thread.id,
                run_id=run.id,
                tool_outputs=[{"tool_call_id": tool_call.id, "output
                   ": str(output)}]
            )
```

3 What is Tool Calling?

Tool Calling is an advanced form of Function Calling, where the AI acts as an intelligent agent, selecting and chaining multiple tools (functions, APIs, or services) based on context to complete complex tasks.

- **Definition**: The AI uses a set of registered tools, each with detailed metadata (name, description, parameters), to perform multi-step workflows.
- **Key Features**:
  - Supports multiple tools (e.g., search, calendar, APIs).
  - Detailed metadata for better decision-making.
  - Enables tool chaining for complex tasks.
- **Real-Life Example**: In a travel app, a user says, “Plan my trip to Islamabad.” The AI:
  1. Calls `search_hotels("Islamabad")` to find hotels.
  2. Calls `check_weather("Islamabad")` to check the weather.
  3. Calls `create_itinerary(...)` to generate a plan.

3.1 Code Example: Tool Calling

Below is a Python snippet demonstrating Tool Calling for a travel planning scenario.

```
import openai

client = openai.Client(api_key="your-api-key")

# Define multiple tool schemas
tools = [
    {
        "type": "function",
        "function": {
            "name": "search_hotels",
            "description": "Search for hotels in a city",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {"type": "string", "description": "City
                       name"},
                    "check_in": {"type": "string", "description": "
                       Check-in date"}
                },
                "required": ["city", "check_in"]
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "check_weather",
            "description": "Get weather forecast for a city",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {"type": "string", "description": "City
                        name"}
                },
                "required": ["city"]
            }
        }
    }
]

# Create a thread and message
thread = client.beta.threads.create()
message = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="Plan a trip to Islamabad for next Friday."
)

# Run the assistant
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id="your-assistant-id",
    tools=tools
)

# Handle multiple tool calls
if run.status == "requires_action":
    tool_outputs = []
    for tool_call in run.required_action.submit_tool_outputs.
        tool_calls:
        if tool_call.function.name == "search_hotels":
            output = {"hotels": ["Hotel A", "Hotel B"]}
            tool_outputs.append({"tool_call_id": tool_call.id, "
                output": str(output)})
        elif tool_call.function.name == "check_weather":
            output = {"weather": "Sunny, 30°C"}
            tool_outputs.append({"tool_call_id": tool_call.id, "
                output": str(output)})
    client.beta.threads.runs.submit_tool_outputs(
        thread_id=thread.id,
        run_id=run.id,
        tool_outputs=tool_outputs
    )
```

4 Function Calling vs Tool Calling

The key differences between Function Calling and Tool Calling are outlined below:

Feature	Function Calling	Tool Calling
Scope	Limited to 1-2 backend functions	Multiple modular tools (f
Flexibility	Low; designed for simple tasks	High; supports complex, r
Tool Metadata	Minimal (name, basic parameters)	Detailed (schema, descrip
Multi-tool Workflow	Difficult; no chaining	Supported; tools can be c
AI Decision-Making	Simple; direct function mapping	Advanced; context-aware
Example Task	"Get weather in Lahore"	"Plan a trip: find hotels,
Use Case	Single-purpose tasks (e.g., currency conversion)	Multi-step tasks (e.g., tra

Table 1: Comparison of Function Calling and Tool Calling

4.1 Detailed Differences

- **Complexity**: Function Calling is for straightforward tasks, like calling a single API. Tool Calling handles complex workflows, such as combining multiple APIs or services.
- **AI Intelligence**: In Function Calling, the AI makes basic decisions about which function to call. In Tool Calling, the AI acts as an agent, deciding which tools to use and in what order based on context.
- **Scalability**: Function Calling is less scalable for large systems, while Tool Calling supports modular, reusable tools for enterprise-level applications.
- **Error Handling**: Tool Calling often includes stricter parameter validation (e.g., `strict: true`) to ensure robust execution.

5 Agentic AI Context

Agentic AI refers to AI systems that can autonomously make decisions, interact with external systems, and perform tasks using tools. Function Calling and Tool Calling are critical components of Agentic AI, enabling:

- **External Interaction**: Connect to APIs, databases, or services (e.g., weather APIs, calendar apps).
- **Task Automation**: Automate multi-step processes like booking flights or generating reports.
- **Context Retention**: Use memory to maintain conversation context for better decision-making.

5.1 Real-Life Applications

- **E-commerce**: A user asks, "Find me a laptop under $500 and check its reviews." The AI uses Tool Calling to:
  - Call `search_products("laptop", budget=500)`.
  - Call `get_reviews(product_id)`.
- **Education**: A student asks, "Summarize this lecture and schedule a study session." The AI:

  - Calls `summarize_text(lecture_content)`.
  - Calls `add_to_calendar("Study Session", date)`.
- **Healthcare**: A doctor asks, “Check patient history and suggest a treatment plan.” The AI:
  - Calls `retrieve_patient_history(patient_id)`.
  - Calls `generate_treatment_plan(history)`.

6 How It Works in OpenAI SDK

The OpenAI Assistants API follows these steps for Function/Tool Calling:

1. **Define Tools**: Register tools with their schemas (name, description, parameters).
2. **Create Thread**: Store user messages in a conversation thread.
3. **Initiate Run**: The AI analyzes input and selects appropriate tool(s).
4. **Execute Tools**: The developer runs the tools and submits outputs back to the AI.
5. **Generate Response**: The AI formats the output into a natural language response.

7 Conclusion

Function Calling is ideal for simple, single-purpose tasks, while **Tool Calling** enables complex, multi-step workflows in Agentic AI. By leveraging the OpenAI SDK, developers can build intelligent assistants that interact with external systems, automate tasks, and provide context-aware responses.

For more details, refer to the OpenAI Documentation.