Tools and Function Calling

The tool use / function calling mechanism that lets an LLM request real code execution: the tool-calling loop, defining a tool, and how tool use differs from an "agent."

Intermediate 18 min

Last updated: August 22, 2026
Read online: https://www.learnforgex.com/en/topics/tools-and-function-calling

---

Tools and Function Calling

"LLM Capabilities and Limitations" ended on a single root cause: an LLM predicts plausible text, with no built-in fact-checker, calculator, live data connection, or way to actually *do* anything in the world. On its own, a model can only produce more text. **Tools** (also called **function calling**) are the mechanism that closes that gap -- letting a model trigger real code, running outside the model, and use the result. This lesson covers what tool use actually is before "Introduction to MCP" covers the standard protocol built around it.

What Is Tool Use (Function Calling)?

Tool use is a pattern where an LLM, instead of only generating a final answer, can generate a *structured request* to call a specific function -- naming the function and supplying arguments for it -- and a program outside the model actually executes that function and returns the result. The model itself never runs any code and never directly touches a network or a database -- a more accurate mental model: the model doesn't carry out operations on external systems itself, it *requests* them from the application hosting it, via structured tool-call output. ("It only ever produces text" is a simplified way of putting this -- with multimodal models or vendor-specific tool-call output formats, don't read that as an absolute technical claim; what matters isn't the shape of the model's output, it's that the model itself can never reach the outside world directly.) What makes tool use different from an ordinary response is that the text it produces is deliberately shaped (by the application built around the model) to be recognized as a tool call rather than shown directly to the user. Whether this feature is called "tool use," "function calling," or "tool calling" depends on the vendor, but the underlying idea is the same one described here.

Why Does It Exist?

Recall from "LLM Capabilities and Limitations" that a model's knowledge stops at its **knowledge cutoff**, and that it cannot verify facts, run calculations reliably at scale, or access anything outside its **context** (see "A First Look at Context" in "How Large Language Models Work"). None of that is fixable by making the model bigger or training it differently -- it's a structural consequence of what an LLM is: a next-token predictor. Tool use exists to route around the structural limit instead of trying to patch it: for anything that needs to be current, exact, or backed by a real action -- today's weather, a database lookup, an exact arithmetic result, sending an email -- the model doesn't attempt to produce that information from its trained knowledge at all. It requests that a tool be run, and the *tool's* output (not the model's guess) becomes the answer.

The Tool-Calling Loop

A tool call is not a single step -- it's a short loop between the model and the application hosting it:

1. The application sends the model a prompt, along with a list of available tools (names, descriptions, and the arguments each one accepts).
2. The model decides, based purely on the text of the conversation and the tool descriptions, whether answering requires a tool. If not, it responds normally.
3. If a tool is needed, the model produces a structured tool call instead of a normal answer: which tool, and what arguments.
4. The **application** (not the model) executes the real function -- calling an API, querying a database, running code -- and captures its result.
5. That result is appended back into the model's context as a new piece of input, and the model is called again, now able to use the tool's output to write its actual answer -- or to request another tool call, if one result leads to needing another.

This loop can run more than once before a final answer is produced, and every round trip consumes more of the model's context (recall "Tokens and Context Windows"). Crucially, the model never executes anything itself in this loop -- step 4 always happens in ordinary application code that the model has no direct access to.

Defining a Tool: Name, Description, and Schema

A tool is described to the model with three parts, and the model's ability to use it correctly depends entirely on how well these are written:

- **Name** -- a short, unambiguous identifier, like `get_current_weather` or `search_orders`.
- **Description** -- a plain-language explanation of what the tool does and when to use it. This is one of the most important signals in whether a model picks the right tool at the

right time -- a vague description ("gets data") leads to a model guessing wrong far more often than a specific one ("looks up the current shipping status of an order, given an order ID"). (Tool selection doesn't depend on the description alone -- the tool's name, its parameter schema, the surrounding conversation context, and the model's or application's own design all play a role too; but the description is one of the signals a developer directly controls, and it makes an outsized difference.)

- **Parameter schema** -- a structured definition (commonly JSON Schema) of what arguments the tool accepts, their types, and which are required. The model uses this schema to decide what values to fill in, and a well-typed schema (e.g., an order ID as a string, not free text) sharply reduces malformed calls.

None of this is guesswork on the model's side in the way free-text generation is -- the model has been trained specifically to produce tool calls that match a given schema, which is why schema quality matters so much in practice.

Tool Use vs. Agents

A single tool call -- get the weather, return the answer -- is not yet what this course later calls an **agent**. Tool calling is one of the core mechanisms agent systems are built on; an agent is a broader *system* that can plan multiple steps toward a goal, decide for itself which tools to use and in what order, and sustain the loop described above, across several rounds, with some degree of autonomy and without a human deciding each step. Every agent relies on tool use, but the reverse isn't true -- using one tool once, inside an otherwise ordinary conversation, is not by itself an agent -- the "AI Agents" category later in this course covers what additionally has to be true for a system to earn that name.

Best Practices

- Write tool descriptions the way you'd explain the tool to a new teammate, not the way you'd name a variable -- vague descriptions are the most common cause of a model calling the wrong tool (see "Defining a Tool: Name, Description, and Schema").
- Keep a tool's parameter schema as narrow and typed as the real function allows -- an enum of three valid values gives the model far less room to produce an invalid call than an unconstrained free-text field.
- Design for the loop, not a single call (see "The Tool-Calling Loop") -- a tool's result may itself trigger another tool call, so a tool's output should be something the model can reasonably reason about, not just a raw dump of data.

Common Mistakes

- **Assuming the model executes the tool itself.** As "The Tool-Calling Loop" described, the model only ever produces a request -- the surrounding application is always the one that actually runs the code and is responsible for doing so safely.

- **Writing a one-line tool description and expecting reliable selection.** "Defining a Tool: Name, Description, and Schema" showed that description quality is the main signal the model uses to choose between tools -- an underspecified description is the most common real-world cause of the wrong tool being called.
- **Calling any use of a tool an "agent."** As "Tool Use vs. Agents" explained, tool use is the mechanism; an agent is a specific kind of system built with it, covered later in this course.

Summary, Cheat Sheet, and Glossary

Summary

- Tool use (function calling) lets a model request that real code be run outside itself, closing the structural gap left by knowledge cutoff and the lack of built-in verification described in "LLM Capabilities and Limitations".
- The model never executes anything -- it produces a structured request; the hosting application executes the real function and returns the result.
- The tool-calling loop can run multiple rounds: prompt with tools -> model decides -> application executes -> result fed back into context -> model continues or answers.
- A tool is defined by its name, description, and parameter schema -- description quality is one of the most important signals in correct tool selection.
- Tool calling is one of the core mechanisms agent systems are built on; an agent is a broader system that plans multiple steps toward a goal and picks tools -- a single tool call is not by itself an agent.

Cheat Sheet

- Tool use / function calling = model requests a function call; app executes it, not the model.
- Loop = prompt+tools -> model decides -> app executes -> result back into context -> model continues.
- Tool definition = name + description + parameter schema; description quality strongly influences correct selection.
- Tool use != agent. An agent plans toward a goal, picks tools, and sustains the loop with some degree of autonomy.

Glossary

- **Tool use / function calling:** a mechanism letting an LLM request that an external function be executed, using the result in its response.
- **Tool-calling loop:** the repeated cycle of the model requesting a tool call, the application executing it, and the result being fed back into context.
- **Parameter schema:** a structured definition (commonly JSON Schema) of the arguments a tool accepts, used by the model to construct valid calls.

- **Agent:** a broader system, built on tool use, that can plan multiple steps toward a goal, choose and use tools, and sustain its own working loop with some degree of autonomy, covered later in this course.