Tools For Agents

A conversation about built-in vs. custom tools across AI agent platforms

PROMPT

Are there a built-in set of commands provided by the AI platform that agents can use, or do all of an agent's commands come from definitions in the agent code?

RESPONSE

The answer is **both**, depending on the platform and framework.

The raw Anthropic Claude API does not ship a built-in "command library" that agents automatically inherit. When you build an agent using the Anthropic API directly, you define all tools/functions yourself in the `tools` parameter of the messages request. The model doesn't come with pre-loaded tools — you declare them (name, description, input schema), and Claude decides when to call them based on context.

However, there are layers where built-in tools do exist:

1. **Claude's server-side tools** — In certain contexts (like claude.ai or the API with specific flags), Anthropic provides built-in tools like `web_search` and `text_editor`. These are defined by Anthropic, not by your code. When using the API, you can opt into some of these without having to define the schema yourself.
2. **Claude Code** — The CLI agent ships with a curated set of built-in tools (file read/write, bash execution, etc.) that the agent can use out of the box. You don't define these; they're part of the Claude Code runtime.
3. **MCP (Model Context Protocol)** — This is Anthropic's open standard for connecting tools to agents. MCP servers expose tools that agents can discover dynamically at runtime. So the tools aren't hardcoded in agent code or built into the model — they come from external servers the agent connects to. This is the closest thing to a "plug-in" system.
4. **Frameworks like LangChain, CrewAI, etc.** — These third-party frameworks often bundle pre-built tool libraries (web search, file I/O, calculators, etc.) that agents can use without you writing the tool logic yourself.

So the pattern is: the **base API** is BYOT (bring your own tools), but various **runtimes and frameworks** layer on built-in capabilities. MCP is Anthropic's answer to making tool discovery and sharing more standardized across agents.

PROMPT

Is the answer the same for OpenAI and Gemini agents?

RESPONSE

Yes, the pattern is essentially the same across all three major platforms — the base API is BYOT, with built-in tools layered on top in specific contexts.

OpenAI

The base Chat Completions API requires you to define all tools yourself via the `tools` parameter, just like Anthropic. However, the **Assistants API** (their agent-oriented layer) ships with built-in tools: `code_interpreter`, `file_search`, and `web_search`. You just enable them — no schema definition needed. **GPTs** (custom ChatGPT agents) also get built-in capabilities like browsing, DALL-E image generation, and code execution without user-defined tool schemas. OpenAI supports **function calling** for custom tools, same concept as Anthropic's tool use.

Google Gemini

The base Gemini API also follows BYOT for function calling — you declare tools with names and schemas. Gemini offers some **built-in tools** like `google_search` (grounding with Google Search) and `code_execution` that you can enable without defining them yourself. In **Vertex AI Agent Builder**, Google provides a higher-level agent framework with pre-built connectors and tools. Google has also been pushing **extensions** as their plug-in model, somewhat analogous to MCP or OpenAI's GPT actions.

The Common Pattern Across All Three

The raw LLM API gives you function/tool calling as a mechanism, but you define everything. Each vendor then builds higher-level agent runtimes (Assistants API, Claude Code, Vertex Agent Builder) that bundle commonly needed tools. And all three are converging on some form of tool ecosystem — OpenAI has GPT actions, Anthropic has MCP, Google has extensions — to let agents discover tools dynamically rather than hardcoding them.

The main differentiator right now is MCP being an open standard, while OpenAI's and Google's approaches are more proprietary.