

# What is Model Context Protocol (MCP)?

The Model Context Protocol (MCP) serves as a standardization layer for **AI** applications to communicate effectively with external services such as **tools**, **databases** and predefined templates.

Have you ever tried to build a **multiagent system** but struggled to produce an effective dissemination of information between each specialized agent? Is the variety of prebuilt and custom tools provided to your **AI agent** causing either tool execution or output parsing errors? Or perhaps, have these complications discouraged you from attempting to develop your own agents entirely?

These impediments can be remedied with the Model Context Protocol (MCP). MCP allows AI agents to be context-aware while complying with a standardized protocol for tool integration.

An **AI agent** is a system or program that is capable of autonomously performing tasks on behalf of a user or another system. It performs them by designing its workflow and by using available tools.

**Multiagent systems** consist of multiple AI agents working collectively to perform tasks on behalf of a user or another system.

You can think of MCP for AI applications to serve the same purpose as a USB-C port serves for hardware.<sup>1</sup> This analogy highlights the adaptability USB-C ports provide for connecting hardware comparing it to the standardized way in which various tools and data sources provide context to **AI models** through MCP.

## Tools provide meaning

**Large language models (LLMs)** like **Granite**, Gemini and Llama are limited in their capabilities when deployed on their own. Without any AI tools, LLMs are skilled in several areas including:

- Subsequent text prediction: Prompting an LLM to complete a sentence such as “Jack and Jill went up the...” results in a correct prediction of “Jack and Jill went up the hill.” This prompt and responses is an example of subsequent text prediction and works best on text the model was trained on.
- Basic Q&A: Since an LLM on its own cannot access external databases or web searches, it can answer natural language questions that pertain to information in the data used to train the model. An example can be “Tell me about the treaty of Versailles” because this information about a major world war is most likely included in the training data of general-purpose models. LLMs often perform this text generation in the form of a **chatbot**.
- **Sentiment analysis**: LLMs can process text and determine whether it expresses a positive, negative or neutral sentiment.
- Language translation: LLMs can translate text across languages and geographies. However, not every LLM is trained on data from more than one language.

Aside from basic functions, an LLM without access to external tools cannot successfully run any user query that requires access to real-time information. To provide LLMs with the opportunity to produce more meaningful results, tool integration can be introduced. Providing external tools such as web searches, datasets and **APIs**, allows the LLM to expand its capabilities beyond its training data.

To take this one step further, we can build AI agents by using an LLM and its available tools. Summarizing, agentic systems provide an LLM with a set of tools, allowing the model to determine

appropriate tool use, adapt to a changing environment and form synthesized conclusions based on tool output. However, at scale, these AI systems tend to fail. Hence, MCP, introduced by Anthropic in 2024, establishes an open standard for AI-tool interactions.<sup>2</sup>

Industry newsletter

## The latest AI trends, brought to you by experts

Get curated insights on the most important—and intriguing—AI news. Subscribe to our weekly Think newsletter. See the [IBM Privacy Statement](#).

We use your email to validate you are who you say you are, to create your IBMid, and to contact you for account related matters.

Business email

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe [here](#). Refer to our [IBM Privacy Statement](#) for more information.

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe [here](#). Refer to our [IBM Privacy Statement](#) for more information.

## MCP establishes a standard

It is cumbersome to connect external services to an LLM. Imagine an electrical circuit connecting a motor to various power sources. MCP is like the wiring and switchboard of this circuit; it decides what electrical current (information) flows to the motor (AI model). Tool output or model context can be compared to the input current—it is the voltage flowing from a source of power and can include memory, tools and past findings.

As the switchboard, MCP decides which sources of power (tool output or context) to connect and when to do so, regulates the current (stream of information), filters and prioritizes inputs. It does that to ensure that only relevant wires are energized (the relevant context is loaded) and manages the circuit's timing and routing to not overload the system.

Just as a well-designed circuit prevents overload and ensures efficient power usage, MCP serves as a connector to facilitate efficient, relevant and structured use of context for optimal AI model performance.

MCP establishes a new [open source](#) standard for AI engineers to agree upon. However, standards are not a new concept in the software industry. For example, [REST APIs](#) are industry-standard, offering consistent data exchange between applications through HTTP requests aligned with REST design principles.

Similarly, MCP unifies the LLM and external services to communicate efficiently by setting a standard. This standard allows for “plug-and-play” tool usage rather than writing code for custom integration of each tool.

MCP is not an agent framework, but a standardized integration layer for agents accessing tools. It complements agent orchestration frameworks. MCP can complement agent orchestration frameworks

like [LangChain](#), [LangGraph](#), [BeeAI](#), LlamaIndex and [crewAI](#), but it does not replace them; MCP does not decide when a tool is called and for what purpose.

MCP simply provides a standardized connection to streamline tool integration.<sup>3</sup> Ultimately, the LLM determines which tools to call based on the context of the user's request.

## MCP architecture

The MCP client/server model can be broken down into three key architectural components:

### MCP host

An AI application receives the user requests and seeks access to context through the MCP. This integration layer can include IDEs such as Cursor or Claude Desktop. It contains the orchestration logic and can connect each client to a server.<sup>4</sup>

### MCP client

Communication in the MCP ecosystem between the host and server must go through a client. This client exists within the host and converts user requests into a structured format that the open protocol can process. Multiple clients can exist with a singular MCP host but each client has a 1:1 relationship with an MCP server.

Examples of MCP clients are IBM® BeeAI, Microsoft Copilot Studio, Claude.ai, Windsurf Editor and Postman. Clients serve as the session manager by handling interruptions, timeouts, reconnections and session closures. Clients also parse responses, conduct error handling and verify that responses are relevant to the context and appropriate.<sup>4</sup>

### MCP server

The external service that provides the context to the LLM by converting user requests to server actions. Examples of MCP server integrations include Slack, GitHub, Git, Docker or web search. These servers are typically GitHub repositories available in various programming languages (C#, Java™, TypeScript, Python and others) and provide access to MCP tools.

Tutorials can typically be found within these GitHub repositories to aid in the technical implementation. MCP servers can also be used to connect [LLM inferencing](#), through AI platform providers such as IBM and OpenAI, to the MCP SDK. In doing so, a reusable MCP service is created for clients to access as a “standardized” chat tool.

MCP servers are versatile as they allow connections to both internal and external resources and tools. According to the docs provided by Anthropic, Model Context Protocol servers expose data through:

- **Resources:** Information retrieval from internal or external databases. Resources return data but do not execute actionable computations.<sup>5</sup>
- **Tools:** Information exchange with tools that can perform a side effect such as a calculation or fetch data through an API request.<sup>6</sup>
- **Prompts:** Reusable templates and workflows for LLM-server communication.<sup>7</sup>

The transport layer between clients and servers is responsible for two-way message conversion. In the client-to-server stream, MCP protocol messages are converted into JSON-RPC format, allowing for the transport of several data structures and their processing rules.<sup>8</sup>

In the reverse server-to-client stream, the received messages in JSON-RPC format are converted back into MCP protocol messages.<sup>9</sup> The three JSON-RPC message types include requests, responses and notifications. Requests require a response from the server, whereas notifications do not.

In the transport layer between clients and servers, there are two main transport methods for MCP protocol, both of which transmit messages in JSON-RPC 2.0 format. The first is standard input/output (stdio) which works best for integrating local resources due to the simple input/output information transmission. This format is used for lightweight, synchronous messaging.<sup>4</sup> Such resources include local file systems, databases and local APIs.

The second is server-sent events (SSE) which works best for integrating remote resources. HTTP POST requests serve as the mechanism for transmitting client-to-server messages and SSE is used for the reverse. This format can be used to handle multiple asynchronous, event-driven server-calls at once.<sup>4</sup>

## Benefits of MCP

Imagine a real-world AI that scans your inbox to schedule client meetings, sends stock updates, and texts summaries of the last hour's Slack activity. Every service provider constructs their APIs differently by requiring different information to pass, returning different output schemas. Thus, the slightest change in these tools can result in the collapse of this entire AI workflow infrastructure.

There is also a significant development load on engineers to manually construct these tool connections, [debug](#) and maintain authentication like API keys and tool permissions. Tools are often dependent on the output of other tools and there exist many edge cases in which these connections fail.

Thus, it is critical to provide MCP integration as the middle layer between the LLM and the development tools. In this layer, the MCP can convert tool output in a way that is understandable by the model. Without the need to alternate between CLIs, the tool integration occurs all in one place.

There are many real-world use cases for MCP. For instance, MCP enhances [multiagent orchestration](#) and communication through a shared workspace with common tools, eliminating the need for direct integrations.<sup>3</sup>

MCP can also be used to supplement [retrieval augmented generation \(RAG\)](#). Rather than providing the retriever to search a vector store or knowledge base, MCP can connect to a [vector database](#) through a server action. Searching the database as a tool rather than passing the retriever in every LLM invocation allows for more strategic use of the tool. This approach also allows for further [tool-calling](#) upon data retrieval.<sup>3</sup>

## The future of MCP

MCP represents an evolving approach to LLM tool integration that continues to mature and reshape the space over time. As technical challenges arise and MCP servers evolve, the standard adapts and MCPs continue to improve.

Regardless, the need for standardized tool integration is critical for AI agents to operate autonomously and adapt dynamically to real-world environments.<sup>10</sup> With MCP, we can streamline the automation of complex [agentic workflows](#) to allow for less human oversight. In turn, this shift enabled by MCP allows our time to be spent on more nuanced tasks requiring human intellect and intuition.