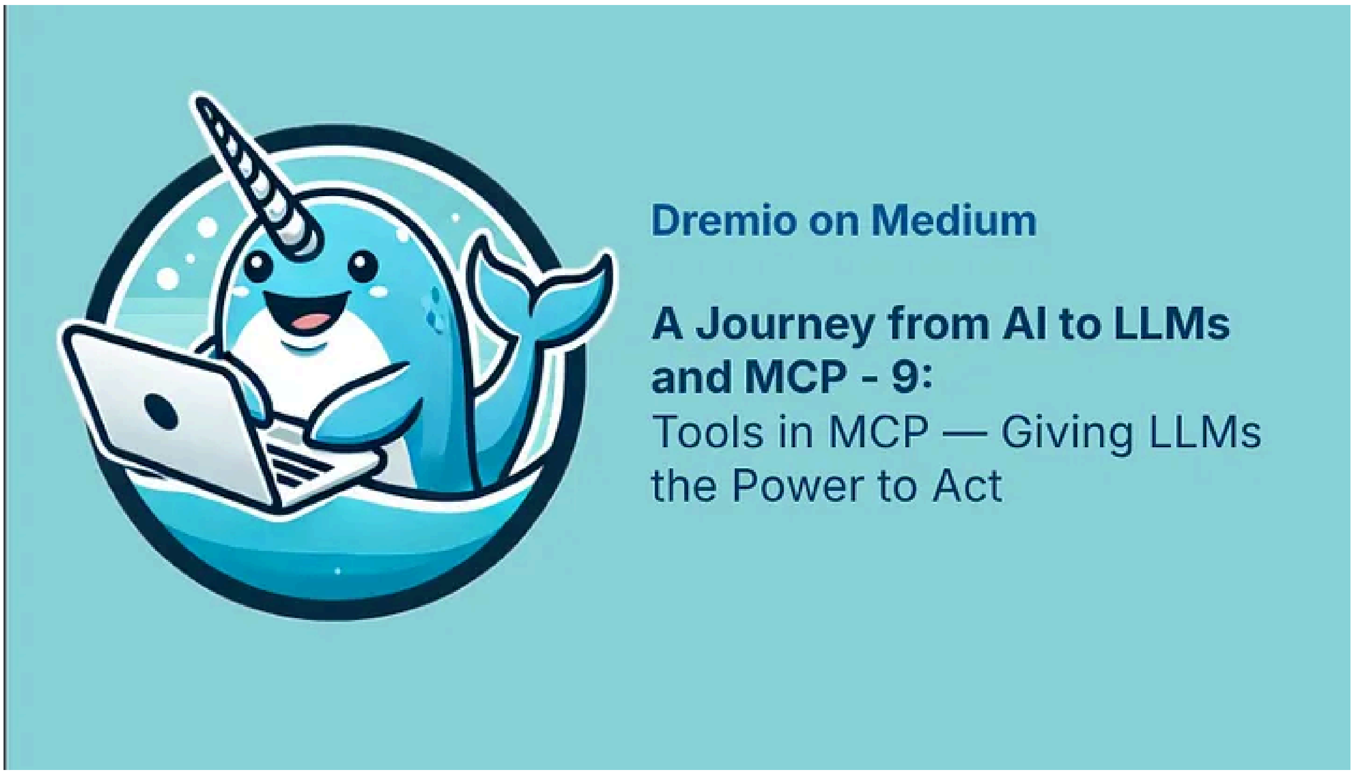
A Journey from AI to LLMs and MCP — 9 — Tools in MCP — Giving LLMs the Power to Act

4 min read · May 15, 2025

Alex Merced Follow


Dremio on Medium
A Journey from AI to LLMs and MCP - 9:
Tools in MCP — Giving LLMs the Power to Act

Free Resources

- Free Apache Iceberg Course
- Free Copy of "Apache Iceberg: The Definitive Guide"
- 2025 Apache Iceberg Architecture Guide

- How to Join the Iceberg Community
- Iceberg Lakehouse Engineering Video Playlist
- Ultimate Apache Iceberg Resource Guide

In the previous post, we looked at Resources in the Model Context Protocol (MCP): how LLMs can securely access real-world data to ground their understanding. But sometimes, *reading* isn't enough.

Sometimes, you want the model to do something.

That's where Tools in MCP come in.

In this post, we'll explore:

- What tools are in MCP
- How tools are discovered and invoked
- How LLMs can use tools (with user control)
- Common tool patterns and security practices
- Real-world examples: from file system commands to API wrappers

Let's dive in.

What Are Tools in MCP?

Tools are executable functions that an LLM (or the user) can call via the MCP client. Unlike resources — which are passive data — tools are active operations.

Examples include:

- Running a shell command
- Calling a REST API
- Summarizing a document
- Posting a GitHub issue
- Triggering a build process

Each tool includes:

- A name (unique identifier)
- A description (for UI/model understanding)
- An input schema (JSON schema describing expected parameters)

> *Tools allow models to interact with the world beyond natural language — under user oversight.*

Discovering Tools

Clients can list available tools via: `tools/list`

Example response:

```
{
  "tools": [
    {
      "name": "calculate_sum",
      "description": "Add two numbers together",
      "inputSchema": {
        "type": "object",
        "properties": {
          "a": { "type": "number" },
          "b": { "type": "number" }
        },
        "required": ["a", "b"]
      }
    }
  ]
}
```

This allows clients (and LLMs) to decide which tools are available and how to call them properly.

⚙️ Calling a Tool

To execute a tool, the client sends:

```
tools/call
```

With this payload:

```
{
  "name": "calculate_sum",
  "arguments": {
    "a": 3,
    "b": 5
  }
}
```

The server responds with:

```
{
  "content": [
    {
      "type": "text",
      "text": "8"
    }
  ]
}
```

That’s it! The LLM can now use this output in a multi-step reasoning chain.

Model-Controlled Tool Use

Tools are designed to be invoked by models automatically. The host mediates this interaction with:

- Approval flows (user-in-the-loop)
- Permission gating
- Logging and auditing

This is what enables “agentic behavior.” For example:

Claude sees a CSV file and decides to call analyze_csv to compute averages — without a user explicitly requesting it.

Tool Design Patterns

Let’s look at some common and powerful tool types:

System Tools

```
{
  "name": "run_command",
  "description": "Execute a shell command",
  "inputSchema": {
    "type": "object",
    "properties": {
      "command": { "type": "string" },
      "args": {
        "type": "array",
        "items": { "type": "string" }
      }
    }
  }
}
```

Use case: Let the LLM grep a log file, or check system uptime.

API Integrations

```
{
  "name": "create_github_issue",
  "description": "Open a new issue on GitHub",
  "inputSchema": {
    "type": "object",
    "properties": {
      "repo": { "type": "string" },
      "title": { "type": "string" },
      "body": { "type": "string" }
    }
  }
}
```

Use case: Let an AI dev assistant file bugs or suggest changes.

Data Analysis

```
{
  "name": "summarize_csv",
  "description": "Summarize a CSV file",
  "inputSchema": {
    "type": "object",
    "properties": {
      "filepath": { "type": "string" }
    }
  }
}
```

Use case: Let the LLM analyze performance metrics or user data.

Security Best Practices

Giving LLMs the ability to take action means security is critical. Here’s how to stay safe:

Validate all input

- Use detailed JSON schemas
- Sanitize input (e.g., file paths, commands)

Use access controls

- Gate sensitive tools behind roles
- Allow user opt-in or approval

Log and monitor usage

- Track which tools are used, with what arguments
- Log errors and output for audit trails

Handle errors gracefully Return structured errors inside the result, not just raw exceptions. This helps the LLM adapt.

```
    {
      "isError": true,
      "content": [
        {
          "type": "text",
          "text": "Error: File not found."
        }
    ]
  }
```

Example: Implementing a Tool Server in Python

```python
@mcp.tool()
async def get_weather(city: str) -> str:
    """Return current weather for a city."""
    data = await fetch_weather(city)
    return f"The temperature in {city} is {data['temp']}°C."
```

This tool will automatically appear in the tools/list response and can be invoked by the LLM or user.

Why Tools Matter for Agents

Agents aren't just chatbots — they're interactive systems. Tools give them the ability to:

- Take real-world actions
- Build dynamic workflows
- Chain reasoning across multiple steps
- Drive automation in safe, auditable ways

Combined with resources, prompts, and sampling, tools make LLMs feel like collaborative assistants, not just text predictors.

Tool Concepts Overview

Coming Up Next: Sampling and Prompts — Letting the Server Ask the Model for Help

In the final two posts of this series, we'll explore:

- Sampling — How servers can request completions from the LLM during workflows
- Prompts — Reusable templates for user-driven or model-driven actions

Tools give LLMs the power to act. With proper controls and schemas, they become safe, composable building blocks for real-world automation.

AI Model Context Protocol Machine Learning

Follow

Written by Alex Merced

633 followers · 14 following

I'm a tech, development and data enthusiast who has a lot to say. You can find all my blogs, videos and podcasts at AlexMerced.com

No responses yet

Bgerby

What are your thoughts?

More from Alex Merced and Data, Analytics & AI with Dremio

In Data, Analytics & AI with Dremio by Alex Merced

Building a Basic MCP Server with Python

Free Resources

Apr 4 163 18

In Data, Analytics & AI with Dremio by Alex Merced

The State of Apache Iceberg v4—October 2025 Edition

Get Data Lakehouse Books:

Oct 14 1

In Data, Analytics & AI with Dremio by Alex Merced

2025 Guide to Architecting an Iceberg Lakehouse

Blog: What is a Data Lakehouse and a Table Format?

Dec 10, 2024 96

Alex Merced

The 2025 & 2026 Ultimate Guide to the Data Lakehouse and the Data Lakehouse Ecosystem

Join the Data Lakehouse Community

Sep 23 5

See all from Alex Merced

See all from Data, Analytics & AI with Dremio

Recommended from Medium

Vandana Maurya

📚 Confusion Matrix — Complete Beginner to Advanced Guide

Here, I am sharing easy-to-read, beginner-friendly, complete note on Confusion Matrix you can go through just before your interview or...

Apr 29

Matteo Ferruccio Andreoni

How to Fine-Tune LLMs Locally: The Complete LoRA Guide

Master LoRA Fine-Tuning on Apple Silicon with minimal memory usage. From setup to deployment in one comprehensive tutorial.

3d ago 87

In Towards AI by Ahmed Boulahia

Best Open-Source Embedding Models for RAG

High-Performance Open-Source Embedding Models for RAG Pipelines, Multilingual NLP, and Arabic Text

Oct 14 201

In Coding Nexus by Civil Learning

MarkItDown: Convert Anything into Markdown — the Smart Way to Feed LLMs

You know that feeling when you're trying to feed a PDF or a Word document into an LLM, and it just doesn't understand what's going on...

6d ago 125 1

In Level Up Coding by Fareed Khan

Building an Agentic Deep-Thinking RAG Pipeline to Solve Complex Queries

Planning, Retrieval, Reflection, Critique, Synthesis and more

2d ago 366 1

In MITB For All by TitusIhy

A gentle introduction to LiteLLM

Unify LLM APIs across OpenAI, Ollama, Groq & more with LiteLLM. Deploy, monitor, and manage models easily via a single proxy server.

Jul 10 108 2

See more recommendations