MCP Simply Explained: Function Calling Rebranded or Genuine Breakthrough?

ZACHARY HUANG
APR 09, 2025

22 2 Sl

Ever wished your AI assistant could actually do more than just chat? Like send real em schedule your meetings, analyze your spreadsheets, or search the web for you? The Mo Context Protocol (MCP) aims to make this possible, but is it really revolutionary? Let's break it down!

1. Introducti Buzz

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

Let's face it - getting iss
Sure, ChatGPT can write you a nice email... but can it actually send it? Claud

describe how to analyze your data... but can it run the analysis for you? Wouldn't i amazing if your AI assistant could seamlessly tap into all your favorite tools and actually get things done?

That's exactly why Anthropic introduced MCP in November 2024, and it's taking AI world by storm! The GitHub repo skyrocketed to 30K stars in less than six mon Even tech giants quickly jumped on board - Google planned to integrate it into th systems, and OpenAI released MCP for their agent.

But here's what everyone's wondering: **Is MCP truly revolutionary, or just a fancy rebrand of what we already had?** Despite all the excitement, there's still plenty of confusion about what MCP actually is. What is it exactly? Is it another AI agent? sophisticated tool routing management system? Or something else entirely? And all the hype surrounding it, is it truly revolutionary or just an incremental improvement?

In this simple tutorial, we'll:

- Start with the basics: how function calling (the predecessor to MCP) actually works
- Build up to MCP by showing what problems it solves that function calling couldn't
- Compare both approaches so you can spot the key differences
- Show real examples of what MCP can (and can't) do for you
- Help you underst

No tech jargon overl les a clear explanation o r A assistant become the

To see these concepts in action, check out our simple working example code that demonsı both function calling and MCP side-by-side. Let's dive in!

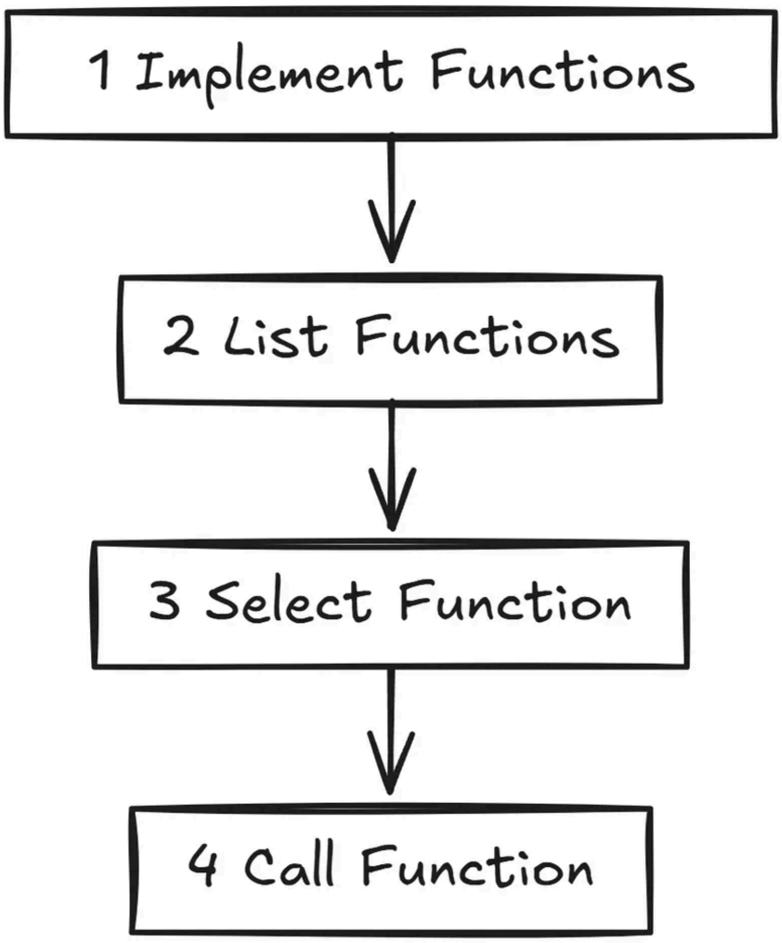
2. Function Calling in Plain English

To understand MCP, we first need to understand function calling - its predecessoı Think of function calling as teaching your AI to "phone a friend" when it needs ex help. The AI handles the conversation, but knows when to call in specialists for ta it's bad at. And AI models can be terrible at certain tasks! Ever asked ChatGPT hc many r's are in "strawberry" and watched it confidently give the wrong answer? O seen Claude botch simple math? Function calling fixes this by connecting AIs to specialized tools that actually work.

Let's use a calculator as our working example. When a user asks "What is 1,567 × 428?", instead of the AI trying to calculate it (and probably failing), function callin lets it use a real calculator and return the correct answer: 670,676.

Here's what a complete function calling workflow looks like:

1. **Implement Functions**: First, we create our calculator functions (add, multiply etc.)
2. **List Functions**: We tell the AI what calculator functions are available
3. **Select Function**: When user asks "What is 1,567 multiplied by 428?", AI selecı "multiply_numbers" with parameters a=1567, b=428
4. **Call Function**: The system executes multiply_numbers(1567, 428) and returns 670,676 to the user



Step 1: Implement Functions

First, we build basic calculator functions:

```
def add_numbers(a, b): return a + b
def subtract_numbers(a, b): return a - b
def multiply_numbers(a, b): return a * b
def divide_numbers(a, b): return a / b
```

Nothing fancy here - just functions that take two numbers and do math. The magi
happens when we cor

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

Step 2: List Fu

Now we need to desc ıt s
"Here's what's availa . to
provide":

```
def get_functions():
    return [
        {
            "type": "function",
            "function": {
                "name": "add_numbers",
                "description": "Add two numbers together",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "a": { "type": "number", "description": "Firs
number" },
                        "b": { "type": "number", "description": "Seco
number" }
                    }
                }
            }
        },
        ... # Other operations follow the same pattern
    ]
```

Step 3: Select Function

When a user asks something, the AI needs to figure out which tool to use. There a two main ways to do this:

The Official Way (APIs)

Most AI providers now have built-in methods for this:

```
def select_function(tools, question):
    from openai import OpenAI

    client = Ope

    response = c
        model="g
        messages=
        tools=to
    )
    # Get the function details
    tool_call = response.choices[0].message.tool_calls[0]
    function_name = tool_call.function.name
    function_args = json.loads(tool_call.function.arguments)

    return function_name, function_args
```

The DIY Way (Clever Prompting)

If your model doesn't have function calling built in, you can hack it with a smart prompt:

~~~
def select_function(tools, question):
    from openai import OpenAI
    import yaml

    prompt = f"""
## Question
{question}

## Tools
{tools}

## Respond in this format:
```yml
thinking: <your reasoning>
tool:
  name: <tool_name>
  parameters:
    <param_name>: <param_value>
    ...
```"""

    client = Oper
    response = c
        model="gp
        messages=
    )

    response_text = response.choices[0].message.content
~~~

```
    yaml_str = response_text.split("```yml")[1].split("```")[0].strip(
    result_data = yaml.safe_load(yaml_str)

    function_name = result_data["tool"]["name"]
    parameters = result_data["tool"]["parameters"]

    return function_name, parameters
```

Either way, if you ask "What's 1,567 multiplied by 428?", you'll get back somethin like:

```
("multiply_numbers", {"a": 1567, "b": 428})
```

Step 4: Call Function

Finally, we run the chosen function with the extracted numbers:

```
def call_function(function_name, arguments):
    # Map function names to actual functions
    functions = {
        "add_numbers": add_numbers,
        "subtract_numbers": subtract_numbers,
        "multiply_numbers": multiply_numbers,
        "divide_numbers": divide_numbers
    }

    # Run it!
    return functions[function_name](**arguments)
```

Put it all together and you get:

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

```
def process_math_
    tools = get_
    function_name
    result = call_
    return result
```

```
# Example
answer = process_math_question("What is 1,567 multiplied by 428?")
print(f"The answer is: {answer}")  # 670,676
```

The real power is that the AI knows when to ask for help, picks the right tool for t job, pulls the necessary information from your question, and formats everything correctly.

This works for way more than just math - weather checks, appointment schedulin database searches - function calling bridges the gap between AI's language skills a real-world tools.

The Big Problem With Function Calling

But traditional function calling has a huge limitation: **your tools and AI must live the same process**. Think of it like this: it's as if your smartphone and coffee make to be physically attached to each other with a cable to work together. Not very practical, right?

What does this mean in real life? If you want to add a cool spreadsheet analyzer to Claude, you can't - because you don't have access to Claude's internal code. You ca add new capabilities to Cursor unless they specifically build a way for you to plug things in. Every AI system would need your exact code reimplemented specifically their environment.

This limitation is exactly what MCP was designed to solve, as we'll see next.

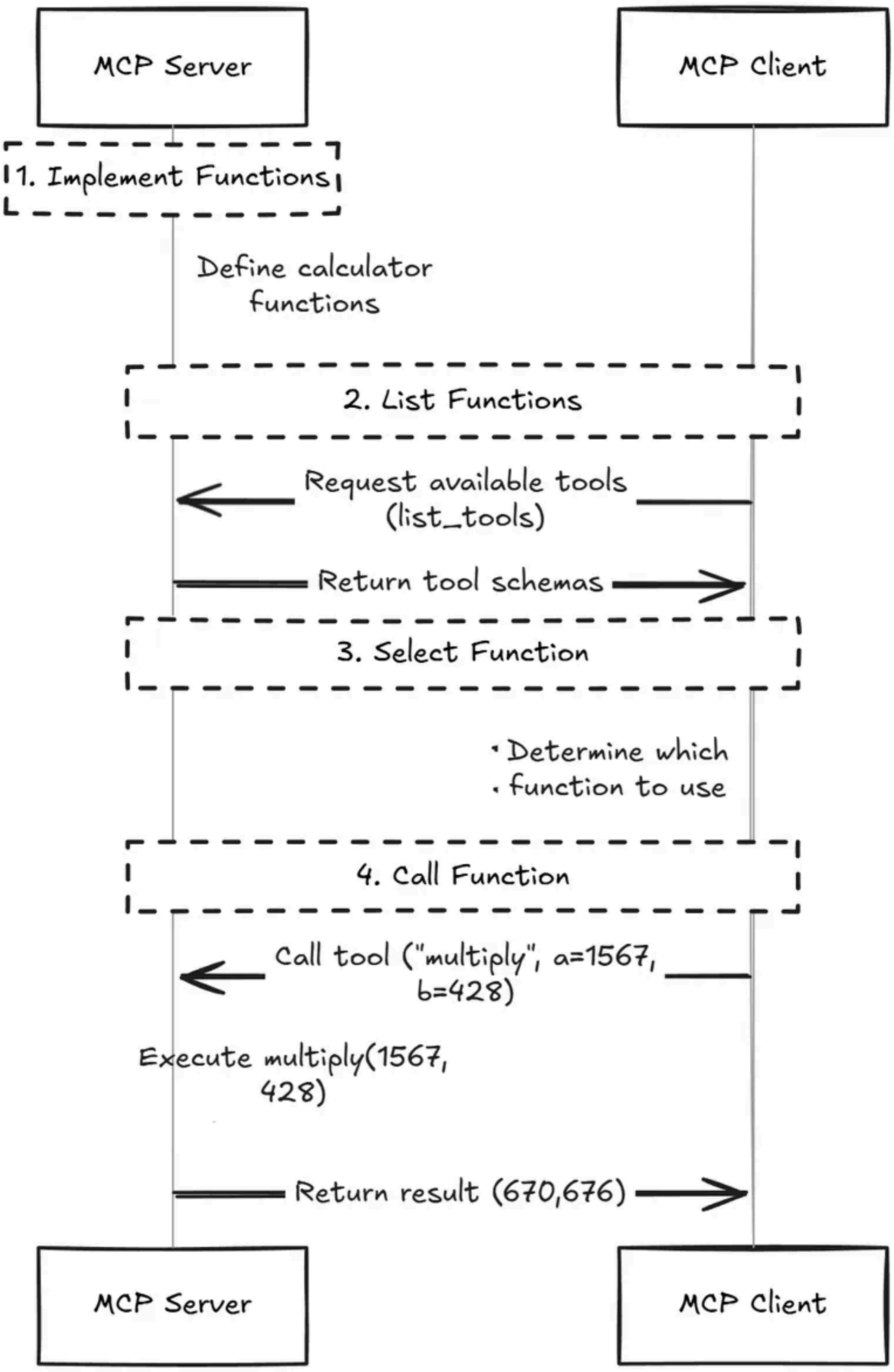
3. MCP: Fun

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

Ever tried connecting sa "Claude, use my calc n y AI and your tools.


MCP Server
MCP Client
1. Implement Functions
Define calculator functions
2. List Functions
Request available tools (list_tools)
Return tool schemas
3. Select Function
Determine which function to use
4. Call Function
Call tool ("multiply", a=1567, b=428)
Execute multiply(1567, 428)
Return result (670,676)
MCP Server
MCP Client

MCP's Big Idea: Split It Up! MCP breaks this problem by separating the tools fro
AI using them:

- **MCP Server** is w er,
  email sender goe
  written in any pr se
  just one specializ

- **MCP Client** serves as the connector (usually part of the AI agent). It finds wha tools are available, helps the AI format requests correctly, and handles communication with servers. Think of it as the food delivery app that knows a the food trucks in town and how to place orders with each one.

> *Zach's Note: The original MCP terminology with "Server, Client, Host" can be confusi Common architecture only has client and server, but don't further separate "hosts" anc "clients" this way. This separation likely derives from extension/plugin system architec (like VS Code's extension model or browser extension systems). The traditional client-s model is sufficient for understanding.*

Let's remake our calculator with MCP:

Step 1: Implement Functions

- **CLIENT SIDE:** Nothing to do yet! The client doesn't need to know how these work - that's the point!
- **SERVER SIDE:** This is where your tool actually lives.

```
from fastmcp import FastMCP

# Create a server with a name
mcp = FastMCP("Math Operations Server")

# Add some math tools
@mcp.tool()
def add(a: int, b: int) -> int:
    """Add two numbers together"""
    return a + b

@mcp.tool()
def subtract(a: i
    """Subtract b
    return a - b

# ... more funct:

# Fire it up
```

```
if __name__ == "__main__":
    mcp.run()
```

Step 2: List Functions

- **CLIENT SIDE:** The client connects to the server and asks what it can do.

```
import os
MCP_SERVER_PATH = os.environ.get("MCP_SERVER_PATH", "simple_server.py'

def list_functions():
    """Find out what tools are available."""
    async def _get_tools():
        server_params = StdioServerParameters(
            command="python",
            args=[MCP_SERVER_PATH]
        )

        async with stdio_client(server_params) as (read, write):
            async with ClientSession(read, write) as session:
                await session.initialize()
                tools_response = await session.list_tools()
                return tools_response.tools

    return asyncio.run(_get_tools())
```

- **SERVER SIDE:** The server responds with details about each tool - what they'ı called, what they do, what inputs they need. It's like a menu board at that fooc truck - listing everything they serve along with ingredients and prices. FastM(handles this automatically by exposing a standard protocol endpoint when `mcp.run()`.

Step 3: Select

- **CLIENT SIDE:** ' e b
  on the user's req
  decision.

~~~
def select_function(tools, question):
    """Figure out which tool to use."""
    from openai import OpenAI
    import yaml

    prompt = f"""
## Question
{question}

## Tools
{tools}

## Respond in this format:
```yml
thinking: <your reasoning>
tool:
  name: <tool_name>
  parameters:
    <param_name>: <param_value>
    ...
```"""

    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[{"role": "user", "content": prompt}]
    )

    response_text = response.choices[0].message.content
    yaml_str = response_text.split("```yml")[1].split("```")[0].strip
    result_data = yaml.safe_load(yaml_str)

    tool_name = result_data["tool"]["name"]
    parameters = result_data["tool"]["parameters"]

    return tool_name, parameters
~~~

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

- **SERVER SIDE:** io calls. Like our fo

Remind me later Hide Forever

Step 4: Call Function

- **CLIENT SIDE:** Sends the request to the server.

```
def call_function(tool_name, parameters):
    """Run a tool on the MCP server."""
    async def _call_tool():
        server_params = StdioServerParameters(
            command="python",
            args=[MCP_SERVER_PATH]
        )

        async with stdio_client(server_params) as (read, write):
            async with ClientSession(read, write) as session:
                await session.initialize()
                result = await session.call_tool(tool_name, parameters
                return result.content[0].text

    return asyncio.run(_call_tool())
```

- **SERVER SIDE:** Does the actual work and returns the result. The server runs t tool and returns the answer - all without the AI needing to know how the tool works! Just like you don't need to know how to cook pad thai to order it from food truck.

The Bottom Line on MCP

MCP completely changes how AI and tools work together:

- **Tools Connect, Don't Embed**: Tools aren't stuffed inside AI apps; they conne through a standard protocol. It's like Bluetooth for AI - connect anything that speaks the same language.
- **Everything's Modular**: Each tool can update independently without changing AI - swap out yo
- **Build Once, Use** ati AI. Write a weat ıst bot.

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

This is incredibly useful for closed-source AI systems where you want to add new capabilities but can't modify the code, and for non-technical folks who don't want mess with integration.

Check out the MCP Example on GitHub for a demo that shows the difference between M and function calls with just ***a simple toggle****.*

---

4. The Myth or the Hype?

With all the excitement around MCP, let's do a reality check on some common cla

"MCP is the first standardized way for tools"

⚖️ **Partially Accurate**

MCP is marketed as "a USB-C port for AI applications," but let's break this down There are two types of standards that people often confuse:

- **At the application level**, OpenAI and Anthropic already had standardized for for function definitions. They have annoying differences (OpenAI uses `parameters`; Anthropic uses `input_schema`) so standardization is helpful.
- **At the transport level**, MCP does shine with multiple transport options (stdic SSE). Unlike OpenAI's ChatGPT Plugins which only use HTTP/REST, which limited for local calls. This flexibility enables both local and remote operation

So yes, MCP brings better standardization, but it's not necessarily "the first."

"As the numb simpler for ag

❌ **Not Accurate**

This claim is like saying having 500 restaurants in your food delivery app makes it easier to decide what to eat for dinner. Having 500 tools via MCP instead of 500 di functions doesn't magically solve the selection problem. The selection is still done the client which still needs custom logic to decide which tool to use, typically with another LLM call. **Both approaches face the same challenge**: LLMs are notorious **BAD** at choosing the right tool from many options. MCP makes the integration w available tools easier but doesn't make the AI any better at picking between them.

"MCP uniquely allows users to add new abilities to agents on-the-fly, that aren't pre-specified"

❌ Not Accurate

This isn't unique to MCP. ChatGPT's plugins already allow new tools from Instac Shopify, Slack, Wolfram, Zapier, and more. However, ChatGPT's plugins have sho that simple integration isn't enough - you need **smart design** around each tool for actually improve results. Having access to tools doesn't guarantee they'll be used effectively. Adding tons of tools often makes performance **worse, not better**. Most agents struggle when choosing between more than ~10 tools. Effective tool use ne thoughtful workflow design, not just plugging in more tools.

"MCP tool discovery is superior"

⚖️ Partially Accurate

There's some truth here, but let's not oversell it. MCP does provide a standardized `/tools/list` endpoint for consistent discovery, but the discovery mechanism is pretty basic - just a flat list of tools. Provide the same list to a traditional function system, and you get similar results. Where MCP truly shines is from an **economic perspective** - once yc rs consumers. It's like t ch attracts users, which ery easier through a grov

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

"MCP is more secure"

⚖️ Partially Accurate

We need to clarify what kind of security we're talking about:

- **Process isolation:** ✅ Yes! Separate processes mean your WhatsApp message won't crash your Claude AI. It's like having your email and browser in separat apps - if your browser crashes, you don't lose your email draft.
- **User data protection:** ⚠️ Maybe not. Your WhatsApp messages could still be leaked if you're using a malicious MCP server. MCP includes human approval tool execution and standardized error handling, which helps. But this is like having a security guard who just asks "Are you sure?" before letting you do something potentially dangerous. Better than nothing, but not foolproof.

"MCP allows function updates without modifying the agent code"

✅ Accurate

This is where MCP genuinely shines. With traditional function calling, adding ne functions means changing your agent code. With MCP, functions live on servers completely separate from the client. New tools can be added without touching cli code at all.

It's like how you can install a new app on your phone without having to update th operating system. You can connect *any language and environment* - Python AI to JavaScript tools, local to cloud, whatever works best. For building AI-facing functi this flexibility is super valuable! You design the right operations, expose it as an M server, and don't worry about the agent.

5. Conclusi

So is MCP right for y

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

- ⚠️ For Technica

Remind me later Hide Forever

f

you're focused on building high performance, reliable systems, integration is the beginning - actually the easy part. There are much harder aspects of syste design and tool orchestration that remain. For critical tasks, you might want t design custom function calls tailored to your specific needs rather than relyin the standardized approach. The added abstraction layer may not provide significant benefits for sophisticated agent development.

> I'll be writing a follow-up tutorial on function call design best practices for bui high-performance agents - stay tuned!

- ✅ **For Non-Technical Users**: MCP offers a "magical" experience, allowing A assistants like Claude to seamlessly integrate with your tools. It's like giving y smart speaker the ability to control not just your lights, but also your custom- robot butler. It's perfect when you need to augment off-the-shelf and potentia closed-source AI agents like Claude AI. If your tasks aren't overly complex an you just want something that "works out of the box" without diving into code MCP delivers exactly that experience.
- ✅ **For Tool Builders**: MCP shines brightest here, letting you build once and connect to any compatible AI system without modifying source code. You can create tools in any language you prefer, expose them either locally or remotely make them standardized and discoverable across the ecosystem. It's like build a Lego piece that fits with any Lego set, no matter who made it. This dramati simplifies distribution - build a great tool and every MCP-compatible system use it immediately.

With what you've learned, you can now evaluate whether MCP makes sense for yo specific needs - whether you're building advanced AI systems, looking to augmen your favorite assistant with custom capabilities, or creating tools that AI can lever The real power of MCP isn't technical superiority but rather its potential to democratize AI tool

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

Want to see the differe th lets you toggle between ges.

22 Likes

← Previous Next

Discussion about this post

Comments Restacks

Write a comment...

Michael Roos May 5

Liked by Zachary Huang

Great article, been loving all your posts.

I've always considered MCP great for 3rd party integrations, in more of a plugin style model
tool calling still has uses for developing more complex AI driven algorithms, like giving it an
function or a DB read function and the model can solve more things all in one prompt, and y
okay with tool calls being tightly coupled

Would be great to see an article on PocketFlow and how you'd approach implementing tool
and LLM driven state transitions to do more complex and non-linear tasks.

LIKE (2) REPLY

Lixx May 20

Liked by Zachary

Thank you for yo

LIKE (1) RE

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever