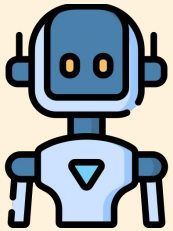
Model Context Protocol (MCP): The API for LLM

May 21,2025

Lingyi Li
UCLA Trustworthy AI Lab
(Directed by Amazon Scholar Prof. Guang Cheng)

Agenda

- Motivation & Intro of MCP
- Deep Dive into MCP
- Next step



How do LLMs help with data analysis?

Example Task: Perform exploratory data analysis and generate plots for the designated dataset.

Solution 1: Provide SQL templates for users

Many templates are available on the data analysis platform.

Developers manually design and maintain hundreds of SQL templates.

Cons:

- Time-consuming and labor-intensive

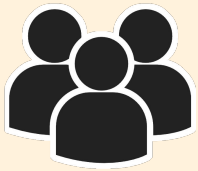
LLM

Solution 2: Text2SQL

Employ LLMs to convert natural language queries into SQL code.

Cons:

- Accuracy



We're not familiar with coding.

How do LLMs help with data analysis?

Example Task: Perform exploratory data analysis and generate plots for the designated dataset.

Tools

Solution 3: LLM with Function Calls

When a user asks a question, the LLM uses the right function to solve it.

Eg. read local data, compute the sample mean, and generate plots

Cons:

- Non-Scalable

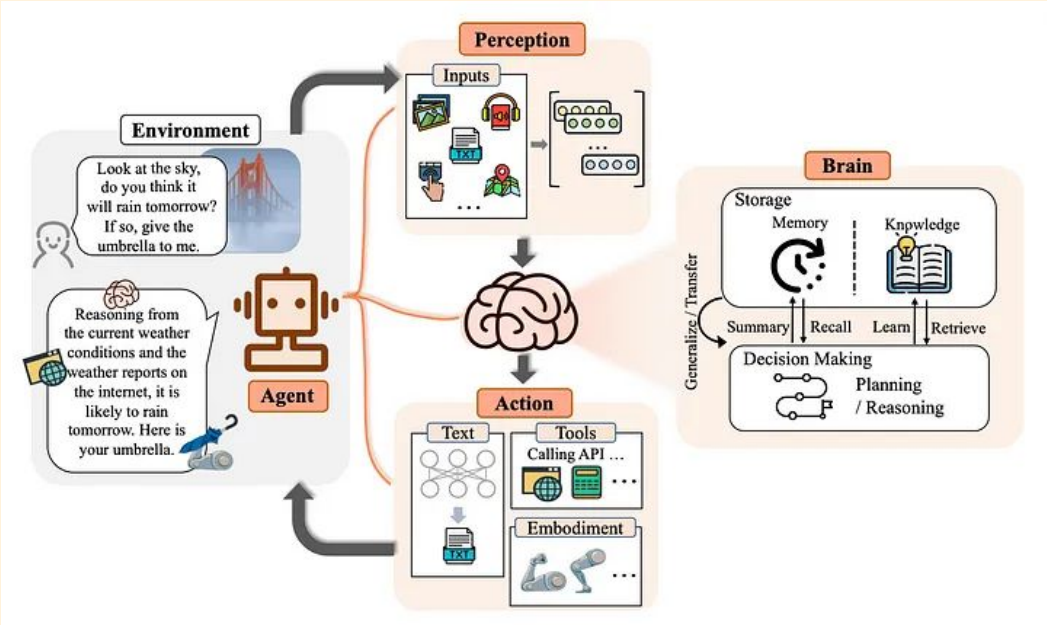
Standardize

Solution 4: LLM with MCP

Similar to function calls —— **Standardize Tools!**

Github Link:
https://github.com/UCLA-Trustworthy-AI-Lab/MCP-Server

Motivation: Enhance Interoperability



Data Isolation: Models frequently function in isolation, without access to real-time data.

Integration Complexity: Each new tool or data source typically requires a custom integration, making it difficult to scale AI solutions effectively

Motivation: Enhance Interoperability

MCP addresses this challenge. It provides a universal, open standard for connecting AI systems with data sources, replacing fragmented integrations with a single protocol. The result is a simpler, more reliable way to give AI systems access to the data they need.

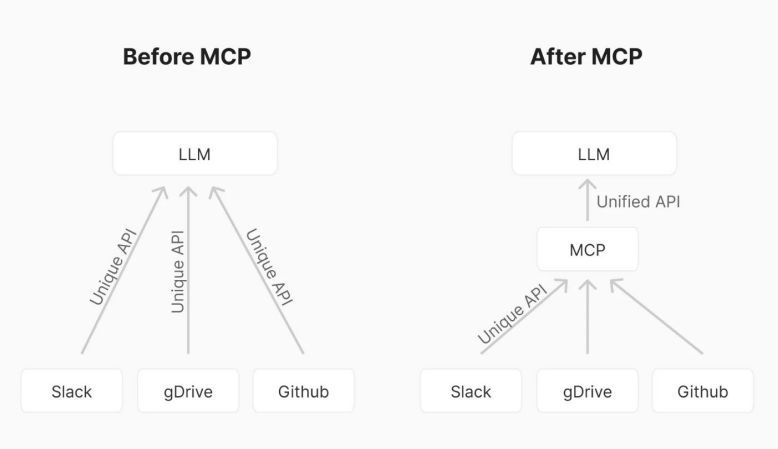
ANTHROP\C

What is MCP?

MCP: API designed for LLM
It lets you give data and functions to the model in a structured and safe way.

Category	REST API	MCP (Model Context Protocol)
Purpose	Communication in web applications	Interaction between LLMs and tools
Resource Unit	URL endpoints	MCP components (Resources, Tools, Prompts)
Invocation Method	HTTP requests (GET/POST)	Internal context calls within the LLM
Target	Expose data/services to human clients	Expose data/services to LLM applications

What is MCP?

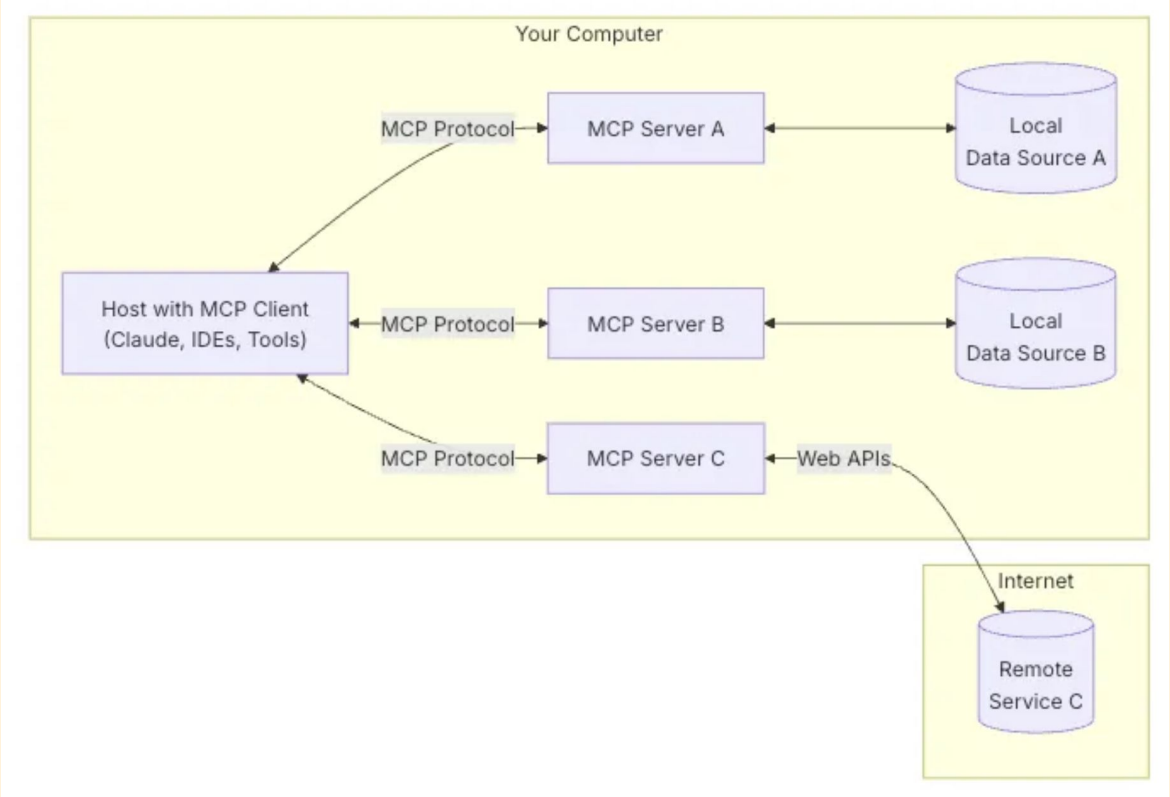


"M×N problem" ⟹ "M+N problem"

MCP implements a client/server architecture with bi-directional communication.

- Share contextual information with language models
- Expose tools and capabilities to AI systems
- Build composable integrations and workflows

What is MCP?



Core Concepts:

- **Hosts**
- **Clients**
- **Servers**
- **Local Data Sources**
- **Remote Services**

Function Call vs. MCP

MCP:

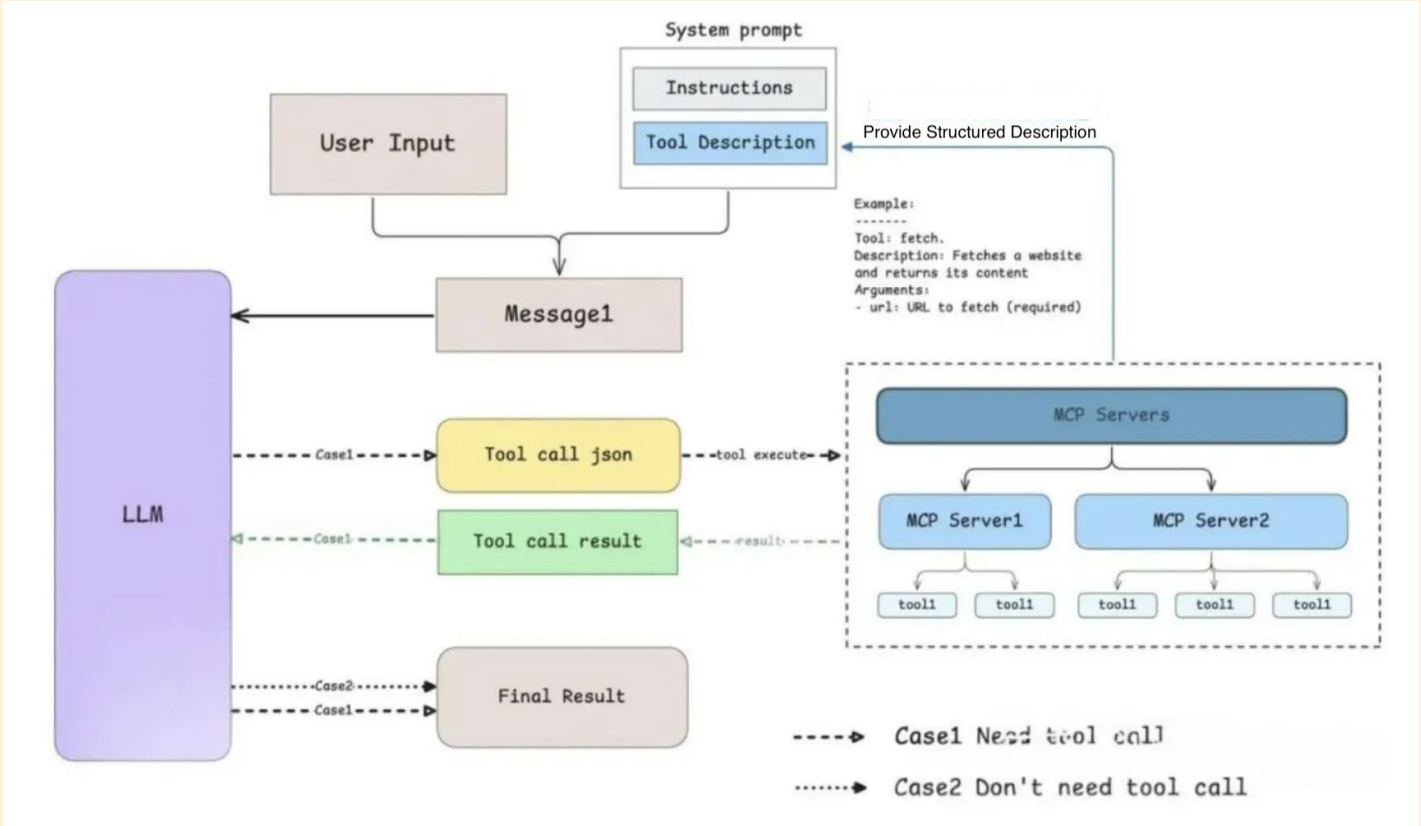
1. A standardized toolbox
2. Promotes modularity and is more efficient for scalable development and integration.

Function calling,

1. Tightly integrated within the LLM
2. Enable the model to directly generate and trigger specific function calls during inference

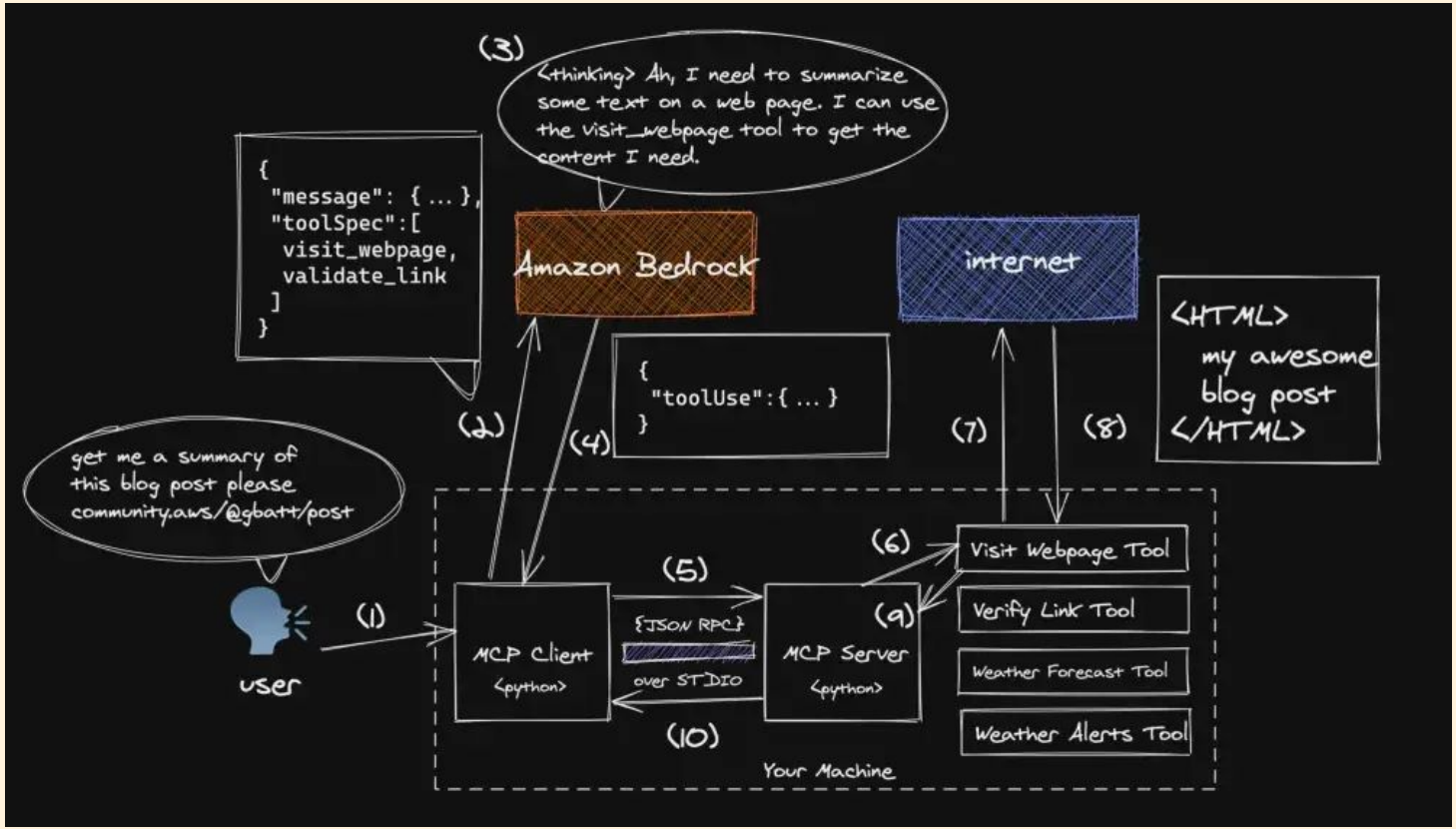
Scenario	Recommended Approach
Complex, secure enterprise systems	MCP
Lightweight, dynamic task execution	Function Calling
Asynchronous, long-running operations	Function Calling
Need for sandboxed, structured environments	MCP
Modular API-based workflows	Function Calling

Architecture & Flows



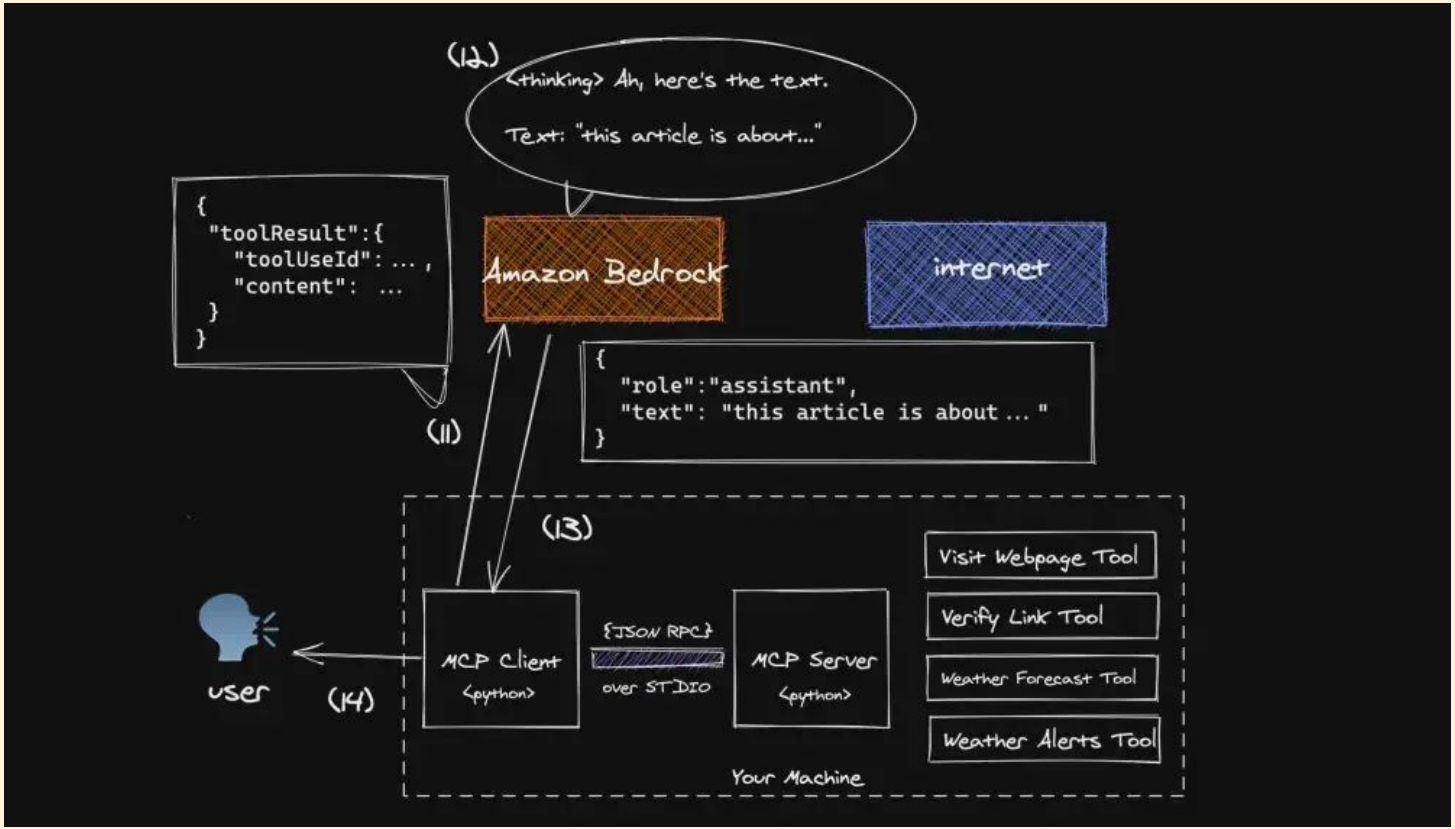
A More Detailed Example

Reference:
https://community.aws/content/2uFvyCPQt7KcMxD9ldsJyjZM1Wp/model-context-protocol-mcp-and-amazon-bedrock



A More Detailed Example

Reference:
https://community.aws/content/2uFvyCPQt7KcMxD9ldsJyjZM1Wp/model-context-protocol-mcp-and-amazon-bedrock



Why MCP is great?

1. A growing list of pre-built integrations
2. Better interoperability,
3. **For companies: efficient for scalable development and integration across systems.**

Implementation and Core Concepts

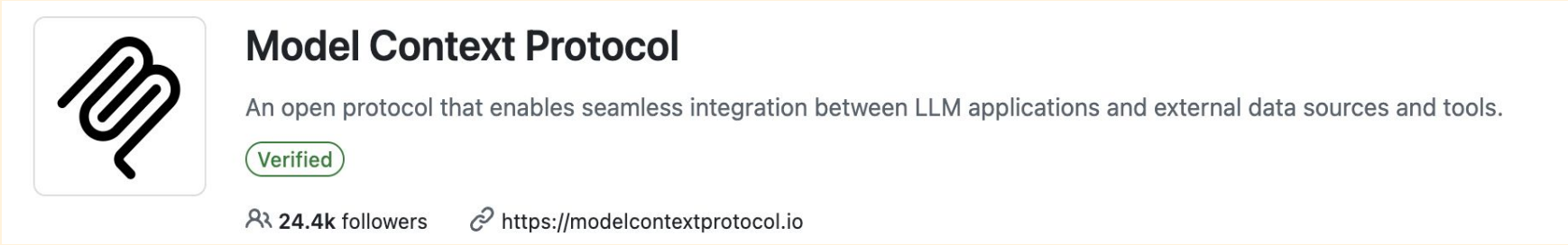
1. Server
2. Tools
   Image
   Context
3. Resources
4. Prompts
5. Authentication
6. Sampling
7. Roots

Implementation and Core Concepts

Languages: TypeScript, Python, Java, Kotlin and C#

Link: https://github.com/modelcontextprotocol

- Build MCP clients and MCP servers
- Use standard transports like stdio, SSE, and Streamable HTTP
- Handle MCP protocol messages and lifecycle events



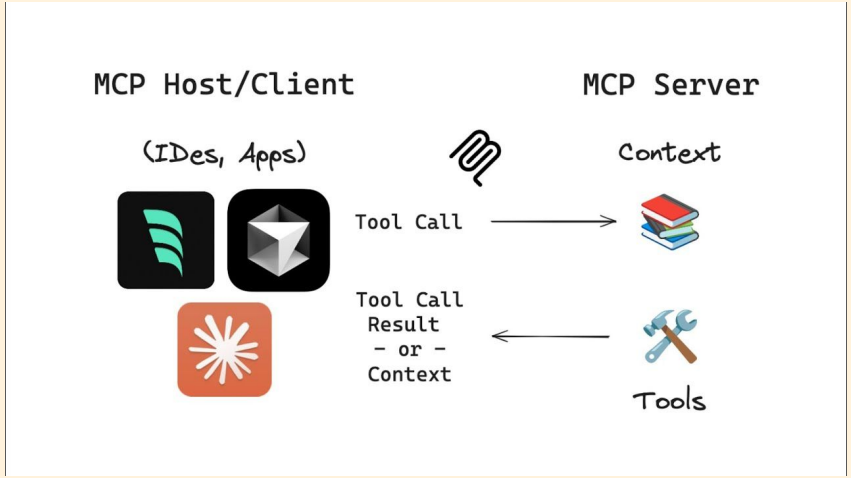
Implementation and Core Concepts

Server:

The FastMCP server is your core interface to the MCP protocol. It handles connection management, protocol compliance, and message routing

```python
# server.py
from mcp.server.fastmcp import FastMCP

# Create an MCP server
mcp = FastMCP("Demo")
```



Implementation and Core Concepts

Tools:

Tools let LLMs take actions through your server.

(sort of like POST endpoints; they are used to execute code or otherwise produce a side effect)

MCP provides a `@mcp.tool()` decorator, which we can use to wrap our function.

The function name will be used as the tool name, its parameters will be treated as tool arguments, and comments (docstrings) can be used to describe the tool, its parameters, and its return value.

```python
@mcp.tool()
def analyze_excel_data(filename: str, sheet_name: Optional[str] = None) -> Dict[str, Any]:
    """
    Perform descriptive analysis on an Excel file

    Args:
        filename: Name of the Excel file (must include .xlsx or .xls extension)
        sheet_name: Optional sheet name to analyze (if not provided, analyzes first sheet)

    Returns:
        Dictionary containing descriptive statistics and column analysis
    """
    desktop_path = get_desktop_path()
    file_path = os.path.join(desktop_path, filename)

    if not os.path.exists(file_path):
        return {"error": f"File {filename} not found on desktop"}

    try:
        # Read the Excel file
        if sheet_name:
            df = pd.read_excel(file_path, sheet_name=sheet_name)
        else:
            df = pd.read_excel(file_path)

        # Basic statistics for numeric columns
        numeric_stats = {}
        for col in df.select_dtypes(include=['number']).columns:
            numeric_stats[col] = {
                "mean": float(df[col].mean()) if not pd.isna(df[col].mean()) else None,
                "median": float(df[col].median()) if not pd.isna(df[col].median()) else None,
                "std": float(df[col].std()) if not pd.isna(df[col].std()) else None,
                "min": float(df[col].min()) if not pd.isna(df[col].min()) else None,
                "max": float(df[col].max()) if not pd.isna(df[col].max()) else None,
                "count": int(df[col].count()),
                "null_count": int(df[col].isna().sum()),
                "unique_values": int(df[col].nunique())
            }
```

Implementation and Core Concepts

Images:

FastMCP provides an `Image` class that automatically handles image data

```python
from pathlib import Path
import os
from PIL import Image as PILImage, ImageFilter
import io
import base64

from mcp.server.fastmcp import FastMCP, Image, Context

# Create an MCP server for image handling
mcp = FastMCP("Image Viewer")

@mcp.tool()
def blur_image(path: str, blur_radius: float = 2.0) -> Image:
    """
    Apply a Gaussian blur effect to an image.

    Args:
        path: Path to the image file. Can use ~/Desktop/ as shorthand.
        blur_radius: The radius of the Gaussian blur. Higher values create more blur.

    Returns:
        Blurred image
    """
```

Context:

The Context object gives your tools and resources access to MCP capabilities

```python
@mcp.tool()
async def process_images_with_context(image_paths: list[str], operations: list[str], ctx: Context) -> str:
    """
    Process multiple images, demonstrating Context's three main features:
    1. Reading MCP resources data
    2. Providing task progress reports
    3. Recording error log messages

    Args:
        image_paths: List of image paths to process
        operations: List of operations to perform ("blur", "brighten", "invert")
        ctx: MCP Context object

    Returns:
        Summary of processing results
    """
    if not image_paths:
        ctx.error("No images provided")
        return "Error: No images provided"

    if not operations:
        ctx.error("No operations specified")
        return "Error: No operations specified"
```

Implementation and Core Concepts

Resources:

Resources are how you expose data to LLMs.

（sort of like GET endpoints; they are used to load information into the LLM's context)

```python
@mcp.resource("excel://sales/all")
def get_all_sales_data() -> dict:
    """Get all sales data from the Excel file"""
    desktop_path = os.path.join(os.path.expanduser("~"), "Desktop")
    excel_path = os.path.join(desktop_path, "Sell_Data.xlsx")
    df = pd.read_excel(excel_path)
    return df.to_dict(orient="records")
```

Prompts:

Prompts are reusable templates that help LLMs interact with your server effectively.

```python
@app.prompt('draw_plot')
async def translate_expert(
        target_plot: str = 'Bar_Plot',
        target_column: str = 'Sales',
        target_file: str = 'Sell_Data.xlsx',
) -> str:
    return f'You are an expert in ploting data, please draw the  {target_plot} of {target_column} in {target_file}'
```

Implementation and Core Concepts

Authentication:

Used by servers that want to expose tools accessing protected resources.

Roots:

Define the boundaries where servers can operate.

Sampling:

It allows servers to request LLM completions through the client, enabling sophisticated agentic behaviors while maintaining security and privacy.

The sampling flow follows these steps:

1. Server sends a `sampling/createMessage` request to the client
2. Client reviews the request and can modify it
3. Client samples from an LLM
4. Client reviews the completion
5. Client returns the result to the server

This human-in-the-loop design ensures users maintain control over what the LLM sees and generates.

Implementation of MCP Clients

MCP Client is a protocol client that maintains one to one connection with MCP Server. It is basically a piece that runs inside the MCP host app, such as Claude Desktop, Cursor…

LLM + Transport Layer + Protocol Implementation = MCP Client

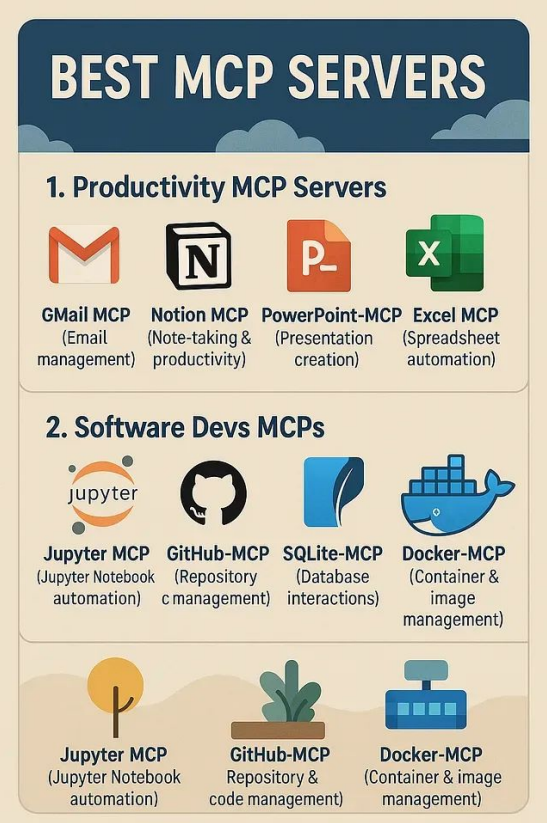
Example MCP Clients

Client	Resources	Prompts	Tools	Discovery	Sampling	Roots
5ire	❌	❌	✅	?	❌	❌
AgentAI	❌	❌	✅	?	❌	❌
Claude Desktop App	✅	✅	✅	❌	❌	❌
Cline	✅	❌	✅	✅	❌	❌
Continue	✅	✅	✅	?	❌	❌
Copilot-MCP	✅	❌	✅	?	❌	❌
Cursor	❌	❌	✅	❌	❌	❌
Daydreams Agents	✅	✅	✅	❌	❌	❌
Emacs Mcp	❌	❌	✅	❌	❌	❌
fast-agent	✅	✅	✅	✅	✅	✅
FLUJO	❌	❌	✅	?	❌	❌

Many applications now support MCP integrations. (contain MCP clients already)

Full List:
https://modelcontextprotocol.io/clients

Example MCP Servers



- Aggregators
- Art & Culture
- Browser Automation
- Cloud Platforms
- Code Execution
- Coding Agents
- Command Line
- Communication
- Customer Data Platforms
- Databases
- Data Platforms
- Delivery
- Developer Tools
- Data Science Tools
- Embedded system
- File Systems
- Finance & Fintech
- Gaming
- Knowledge & Memory
- Location Services
- Marketing
- Monitoring
- Multimedia Process
- Search & Data Extraction
- Security
- Social Media
- Sports
- Support & Service Management
- Translation Services
- Text-to-Speech
- Travel & Transportation
- Version Control
- Other Tools and Integrations

Awesome MCP Servers:
https://github.com/punkpeye/awesome-mcp-servers?tab=readme-ov-file

Microsoft in MCP

- Support MCP across the entire ecosystem
- An updated authorization specification
- A new MCP server registry design

Reference:

https://blogs.microsoft.com/blog/2025/05/19/microsoft-build-2025-the-age-of-ai-agents-and-building-the-open-agentic-web/

AWS in MCP

- **Core** Server: manages and coordinates other MCP servers
- **Documentation** Server: provides access to AWS documentation
- **Nova Canvas** MCP Server: generate images
- **Bedrock Knowledge Base Retrieval** Server: retrieve information from Amazon Bedrock Knowledge Bases
- **Cost Analysis** Server: analyze the cost of AWS services
- **AWS Lambda** Server: select and run AWS Lambda functions as MCP tools

For more AWS MCP servers: https://awslabs.github.io/mcp/

Guidance for Deploying Model Context Protocol Servers on AWS:
https://aws.amazon.com/solutions/guidance/deploying-model-context-protocol-servers-on-aws/

How MCP helps AMC?

- Convert existing AMC API to MCP servers
- Develop MCP servers for new AMC functions
- Research: Scalable MCP-Server generation via GenAI
- Research A2A: Discoverability of MCP servers via A2A

If you have any ideas related to MCP, feel free to contact [redacted]

Agent-to-Agent (A2A)

Interoperability and collaboration between AI agents

Our MCP v.s. Gumloop

Github Link: https://github.com/UCLA-Trustworthy-AI-Lab/MCP-Server

- More flexible
- Highly customizable

Appendix: Communication Protocol

Its communication protocol can be served via two transport mechanisms:

1. Stdio transport
    - Uses standard input/output for communication
    - Ideal for local processes
2. HTTP with SSE transport
    - Uses Server-Sent Events for server-to-client messages
    - HTTP POST for client-to-server messages

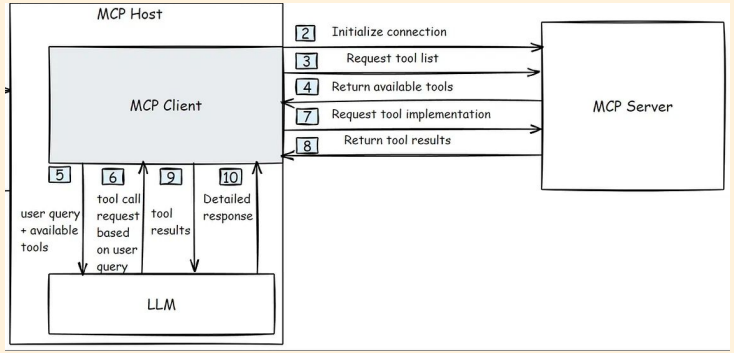
All transports use **JSON-RPC** 2.0 to exchange messages between:

- Hosts: LLM applications that initiate connections
- Clients: Connectors within the host application
- Servers: Services that provide context and capabilities

Appendix: Implementation of MCP Clients

How to build an MCP Client?

1. Capturing and Processing user queries (common)
2. Connecting to MCP servers (common)
3. Discovering available tools through tools/list (common)
4. Translating tool definitions for LLM compatibility (LLM Specific)
5. Forwarding user queries to LLMs with available tools (LLM Specific)
6. Processing LLM responses to identify tool calls (LLM Specific)
7. Executing tool calls through tools/call (common)
8. Sending tool results to the LLM for continued conversation (LLM Specific)
9. Returning results from LLM to users (common)



Reference:https://www.devshorts.in/p/how-to-build-an-mcp-client?utm_source=substack&utm_medium=email&utm_content=share

Ask anything!

Thank you!