An introduction to Model Context Protocol

Connecting AI Models to Data Sources

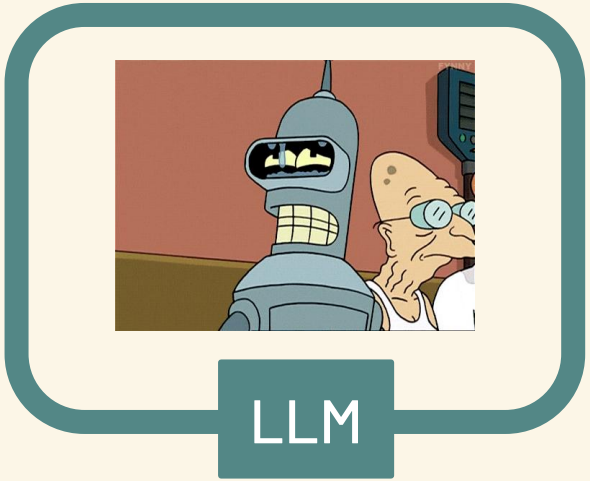
November 6th, 2025

Laura Galera

Table of contents

The AI Sandbox Problem

Your LLM is trapped in a box



📈 No Real-Time Data

"What's the current stock price of TSLA?"

📄 No Private Data Access

"Summarize my Q3_Report.docx file"

✈️ No Ability to Take Action

"Book a flight to SF for next Tuesday"

The Old Ways of Breaking Out

Fine-Tuning

Injects knowledge into **the model itself**.

Pros

- Deeply integrated knowledge

Cons

- Expensive
- Slow

RAG

Injects knowledge into **the prompt**.

Pros

- Good for static knowledge bases

Cons

- Read-only
- Not for actions

Custom Function

Gives the model a list of **functions** it can call.

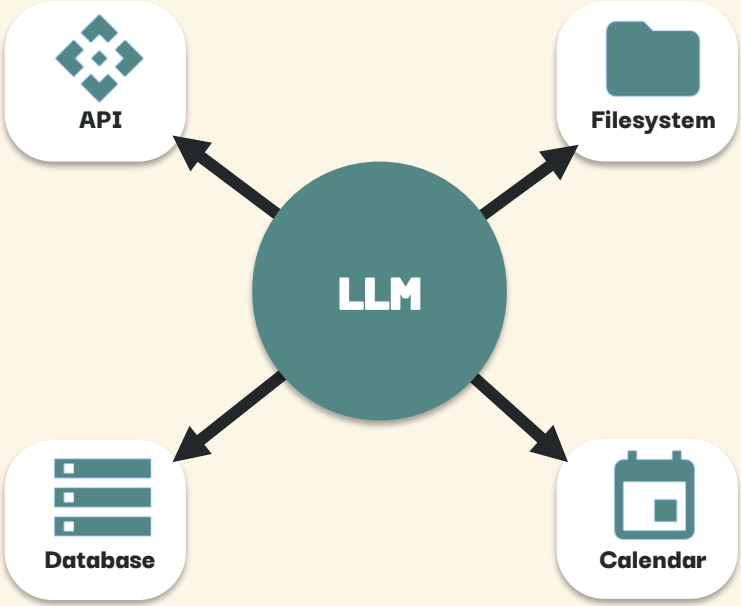
Pros

- Enable actions!

Cons

- Tightly coupled
- Non-standard

Introducing MCP: The “USB-C for AI”



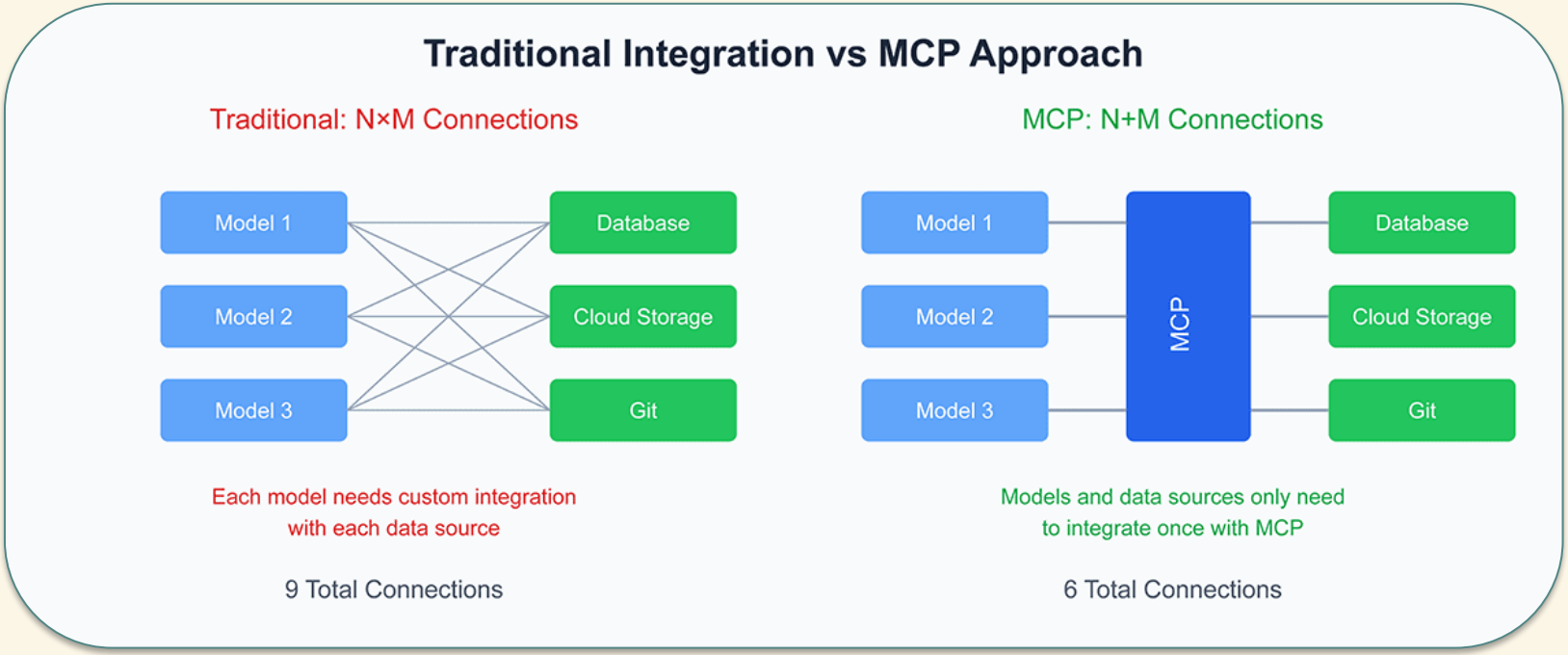
Definition

Model Context Protocol (MCP) is an **open protocol** that standardizes how AI applications provide context to LLMs.

Key Idea

Creates a **clean, secure, and standardized bridge** between AI models and your data sources and tools.

Single protocol replacing **N×M problem** of custom integrations. Enables **fungibility** between AI clients and servers.



Architecture of MCP

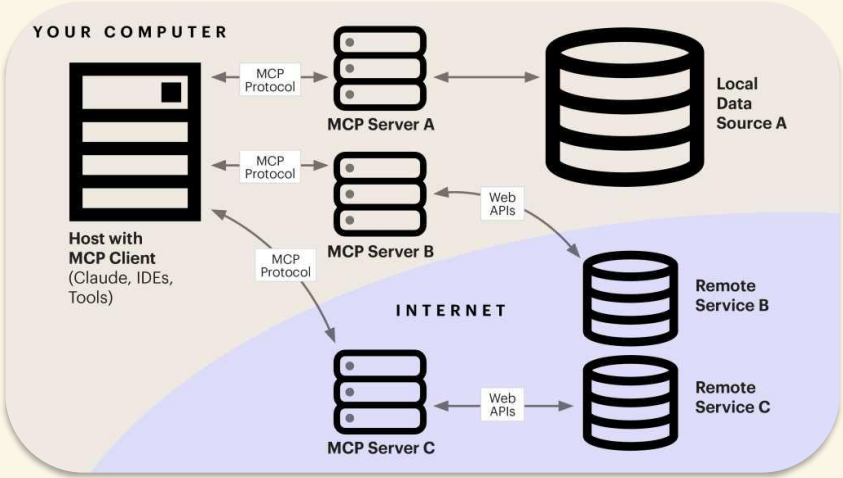
Client-Host-Server Architecture

Client
AI applications that request information and execute tasks

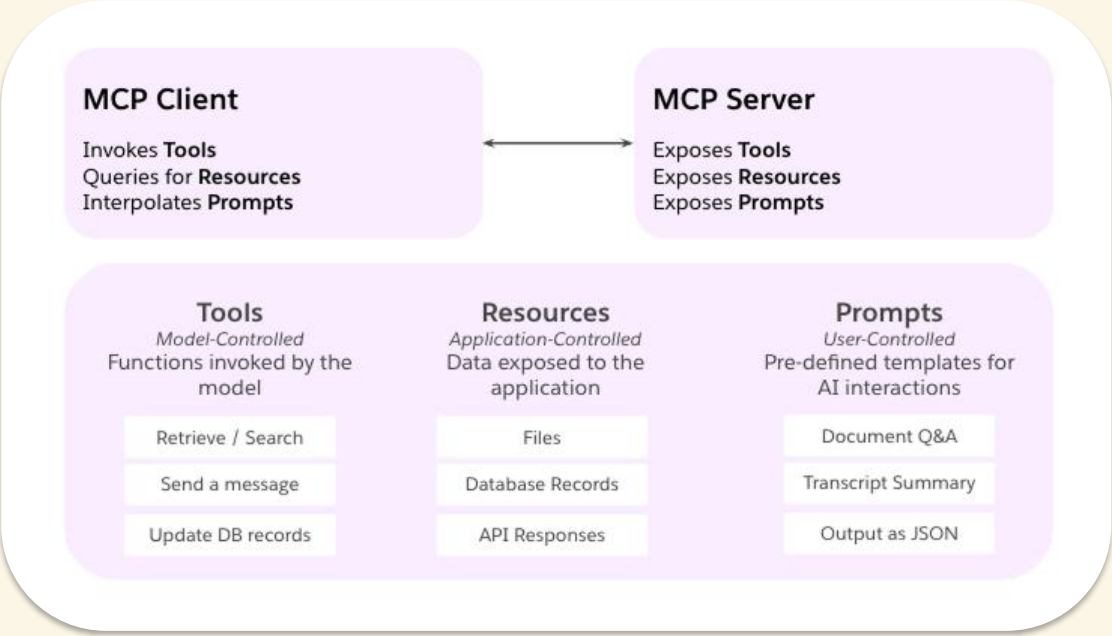
Host
Container and coordinator that creates/manages clients

Server
AI applications that request information and execute tasks

Communication Protocol
JSON-RPC 2.0 messages establish communication between components



Core Features



Core Features - Tools

What are tools?

- **Functions LLMs can call**
  Active capabilities that perform actions
- **Schema-defined interfaces**
  JSON Schema for validation
- **User-controlled**
  May require consent before execution

Key Benefits

- **Model-controlled**
  LLM decides when to use tools
- **Composable**
  Tools can be chained together

Python Example

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

# Initialize server
mcp = FastMCP("weather")

# Define tool with decorator
@mcp.tool()
async def get_alerts(state: str) -> str:
"""Get weather alerts for a US state."""
url = f"https://api.weather.gov/alerts/active/area/{state}"
return await fetch_weather_data(url)
```

tools/list tools/call

Core Features - Resources

What are resources?

- **Structured data access**
  Read-only information for context
- **Unique URI identification**
  Each resource has a unique address
- **MIME type declaration**
  For appropriate content handling

Key Benefits

- **Application-driven**
  Flexible retrieval and processing
- **Parameter completion**
  Helps discover valid values

Python Example

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

# Initialize server
mcp = FastMCP("calendar")

# Define resource with decorator
@mcp.resource("calendar://events/{year}")
async def get_calendar_events(year: str) -> str:
"""Get calendar events for a specific year."""
events = await fetch_events(year)
return format_events(events)
```

resources/list resources/read

Core Features - Prompts

What are prompts?

- **Reusable templates**
  Pre-built instruction templates
- **User-controlled**
  Explicit invocation required
- **Context-aware**
  References tools and resources

Key Benefits

- **Structured workflows**
  Consistent task execution
- **Parameter completion**
  Helps discover valid values

Python Example

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

# Initialize server
mcp = FastMCP("travel")

# Define prompt with decorator
@mcp.prompt()
async def plan_vacation(destination: str, duration: int) -> str:
"""Create a vacation planning prompt."""
return f"""Plan a {duration}-day trip to {destination}.
    Consider weather, activities, and budget."""
```

```
{
"name": "plan-vacation",
"title": "Plan a vacation",
"description": "Guide through vacation planning",
"arguments": [ { "name": "destination", "type": "string", "required": true },
{ "name": "duration", "type": "number", "description": "days" } ]
}
```

Implementation Details

MCP SDKs

- Python
- TypeScript
- Java
- Go
- Rust
- More...

Best Practices

- Use stderr for logging
- Validate inputs
- Async operations
- Clear documentation
- Error handling
- Version compatibility

Setting Up an MCP Server

1. Initialize project with SDK
2. Create server instance
3. Define tools, resources, prompts
4. Implement handlers
5. Configure transport (stdio/HTTP)

Use Cases

Common Use Cases

Code Analysis
PR Reviews, code quality checks

Document Processing
Content analysis, summarization

Workflow Automation
Scheduling, notifications

Integration with AI Applications

Popular AI Clients

- Claude Desktop
- Claude Code
- Cursor
- Windsurf
- Cline (VS Code)
- Custom Applications

Real-World Examples

Travel Planning
Flight search, booking, itinerary

Database Access
Query, analyze, visualize data

Benefits of MCP

Future Directions

Current State

- **Open Protocol**
  Released by Anthropic in November 2024
- **Multiple SDKs**
  Python, TypeScript, Java, Go, Rust, and more
- **Growing Ecosystem**
  Reference servers and community implementations

Future Developments

- **Enhanced Security**
  Advanced authentication and authorization
- **Performance Optimization**
  Reduced latency and improved throughput

Current State

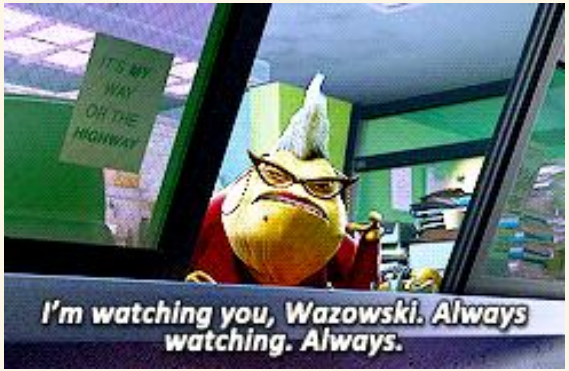
Open Source
Github repositories

Discord Community
Active discussions

Documentation
Tutorials and guides

List of interesting MCP Servers

- Figma MCP Server
- Obsidian MCP Server
- Asana MCP Server
- Notion MCP Server
- AWS MCP Server
- Brave MCP Server
- Slack MCP Server
- And more...



DEMO: DISCORD MCP

Thanks!

Do you have any questions?