

MCP Server

Model Context Protocol: Basics

The standardized interface between AI and external tools, data sources & systems

Robert Klugseder (ACDH)

What is MCP?

The Model Context Protocol (MCP), introduced by Anthropic on November 25, 2024, is an open protocol that connects large language models with external tools, data sources and systems.

Using MCP, AI applications like Claude or ChatGPT can connect to data sources (e.g. local files, databases), tools (e.g. search engines) and workflows (e.g. specialized prompts) - enabling them to access key information and perform tasks.

MCP Host

Application with LLM (e.g. Claude Desktop, Claude Code CLI, Gemini CLI, Cursor IDE, VS Code, n8n-workflows, AI Agents, Chatbots etc.)

MCP Client

Connection component within the host

MCP Server

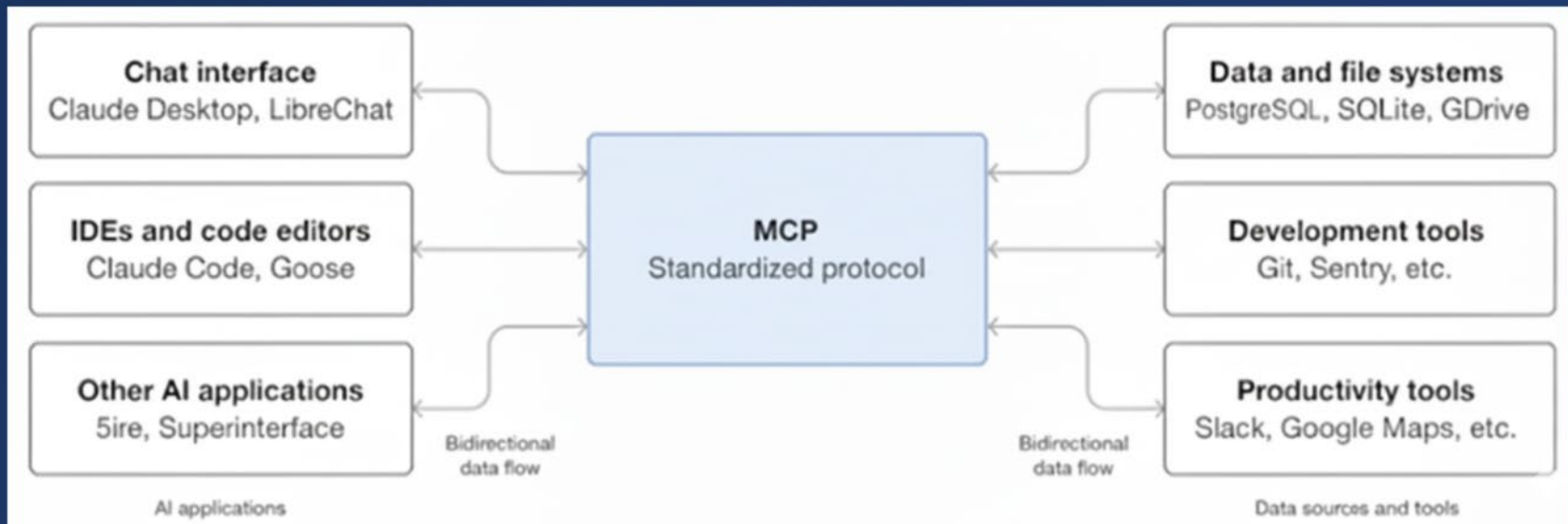
Service that provides tools, resources and functions

Protokoll: JSON-RPC 2.0 | **Framework:** FastMCP

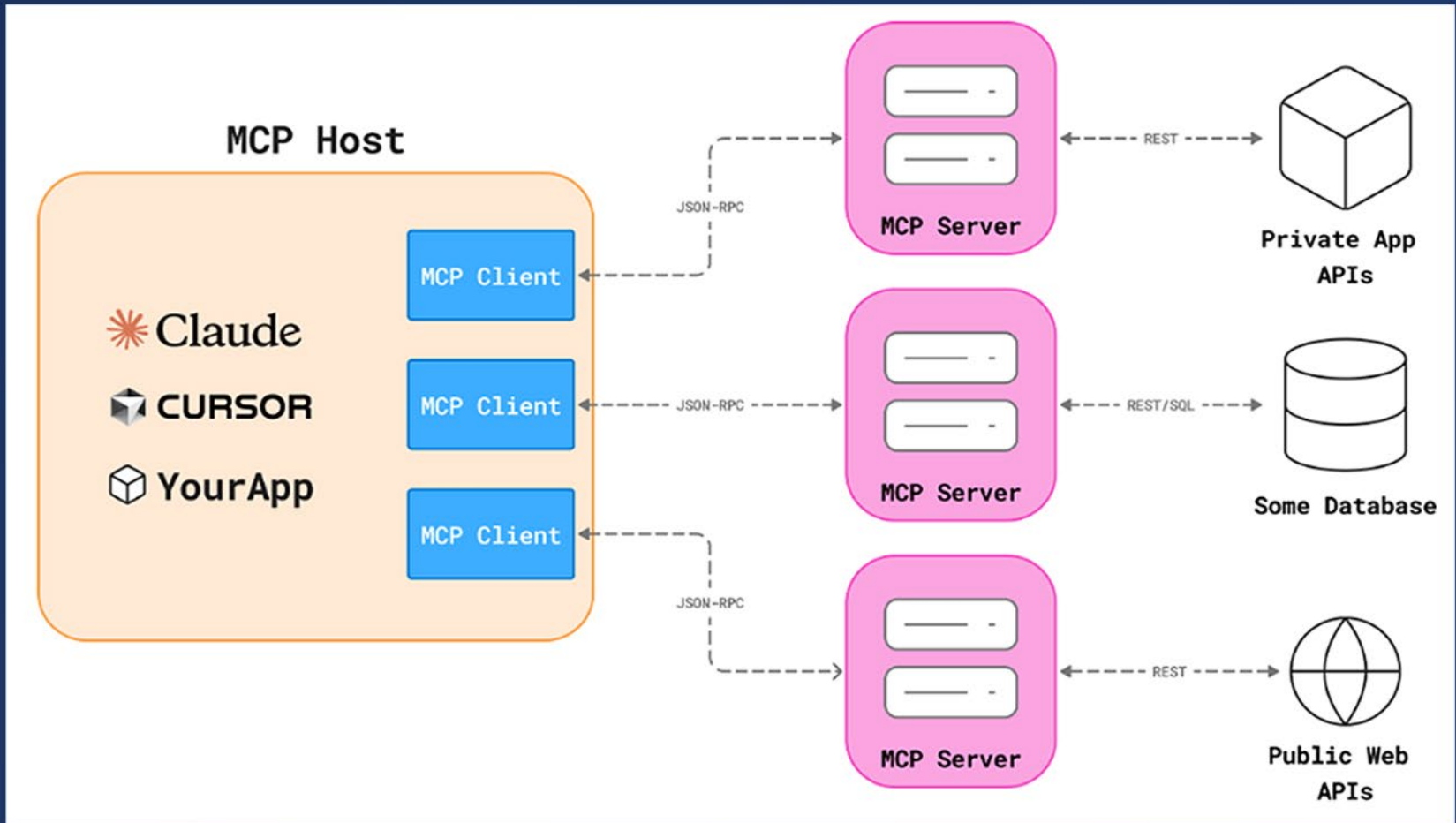
Transport: STDIO (lokal) or Streamable HTTP transport+SSE (remote)

SDKs: TypeScript, Python, Go, Kotlin, Swift, Java, C#, Ruby, Rust and PHP

MCP Server: MCP is for AI what HTTP is for the Web



MCP Server: Use cases



Rapid Development & Adoption

March 2025

~100

MCP Server

June 2025

~2000

MCP Server

November 2025

6488

MCP Server

Main Players

Anthropic

Claude Desktop, Claude Code CLI (& IDE & WEB) and Cloud API with MCP Support

Google, Open AI, Docker, etc., Open Source

Gemini CLI, Codex CLI (ChatGPT only basic connectors), Goose, LM Studio etc.

Microsoft

Copilot & Windows Integration

Cloud-Infrastructure

Cloudflare, Vercel, Hosting-Provider

API Wrapper: The Main Use Case

Transform existing REST APIs into intelligent MCP tools

Direct Translation

Every REST endpoint becomes an MCP tool

Capability Aggregation

Bundle multiple endpoints into high-level tools

Context-Aware Wrapping

Multi-step workflows with session logic

Hybrid Pattern

Selective exposition of specific APIs

Further Use Cases

Customer-Relationship-Management Integration

- ▶ Contact and organization management
- ▶ Access to customer history
- ▶ Multi-system integration

Enterprise Data

- ▶ Database access (MySQL, PostgreSQL)
- ▶ File systems with access control
- ▶ Internal documentation (Confluence)

Development Tools

- ▶ Git repository operations
- ▶ Code execution and testing
- ▶ Browser automation (~ Playwright)

Automation

- ▶ Workflow orchestration
- ▶ Multi-tool coordination
- ▶ Edge Devices

Python MCP Server: Architecture (example: REST API)

3-layer model for robust implementation

Layer 1: FastMCP Framework - High-level API with @mcp.tool decorator for automatic validation and tool discovery

Layer 2: Tool Handlers - Python functions that wrap external REST APIs and implement business logic

Layer 3: JSON-RPC Transport - STDIO (local) or Streamable HTTP transport & SSE (remote) for communication with LLM hosts

Tool Definitions & Input Schemas

Automatic JSON schema generation from Python type hints

FastMCP automatically generates standardized tool definitions

- ▶ Type-safe input validation through Pydantic
- ▶ JSON schema validation for every parameter
- ▶ Optional fields support for flexible APIs
- ▶ Precise error messages for validation failures

Security Functions: 4-Layer Stack

Production-grade Security with FastAPI

Layer	Technology	Function
1. Authentication	API Key Validation	Verification of client identity
2. Rate Limiting	slowapi (10 req/min)	DoS protection
3. Input Validation	Pydantic + JSON Schema	Injection protection
4. Access Control	Least Privilege	Tool-specific authorization

REST API Wrapper Example

JSONPlaceholder API integration with 7 MCP tools

Architecture: REST API → FastMCP Server → MCP Client (~ Claude Desktop)

7 MCP Tools:

▸ `get_posts()`

▸ `get_post_by_id()`

▸ `create_post()`

▸ `update_post()`

▸ `delete_post()`

▸ `get_user_posts()`

▸ `get_user()`

MCP API-Type Compatibility Matrix

Universal API integration for the AI era

REST API ✅ Fully supported	SOAP API ✅ Supported	gRPC API ✅ Supported	GraphQL API ✅ Supported	WebHooks ✅ Supported	WebSockets ⚠️ Community	WebRTC ❌ Not Supported
--------------------------------------	--------------------------------	--------------------------------	-----------------------------------	--------------------------------	-----------------------------------	----------------------------------

Fully Supported (5)

- ▶ REST: OpenAPI YAML/JSON Converter
- ▶ SOAP: WSDL → MCP (open-mcp.org)
- ▶ gRPC: Protobuf Schema Support
- ▶ GraphQL: GraphQL Schema Converter
- ▶ WebHooks: n8n MCP Trigger Nodes

Partial / Special Cases

- ▶ **WebSockets:** Community project (Rust Implementation)
- ▶ yonaka15/mcp-server-runner
- ▶ Status: Work-In-Progress (WIP)
- ▶ Non-official, but functional
- ▶ **WebRTC:** Not supported

Key Finding

MCP is **API-Type agnostisch** - many popular APIs can be easily integrated

Limitations of the Standard Approach

The 'old-fashioned' MCP problem

When all tool definitions must be loaded in the context window:

- ▶ Token bloat (excessive token consumption)
- ▶ High API costs
- ▶ Security Vulnerabilities

Security Statistics

Command Injection Vulnerabilities

Path Traversal Issues

SSRF (Server-Side Request
Forgery)

Code-Execution with MCP

The Game-Changer Solution

The LLM writes code to call MCP tools programmatically

Instead of making direct tool calls, the model generates code that uses MCP tools in an optimized way.

Token Reduction

98.7%

From 150.000 to 2.000 tokens

Cost Savings

Drastic reduction of API costs with equal performance

Agent Skills

Progressive Disclosure: Loading complexity on demand

Organized folder structure with instructions and scripts that work according to the "Progressive Disclosure" principle.

Schicht	Token-Overhead	Table
Metadata Layer	Minimal (~10-50)	Skill name, short description
Core Instructions	~1000 Tokens	SKILL.md with instructions
Details on Demand	As needed	Separat files, only when necessary

MCP vs Skills vs Code-Execution

Complementary architecture layers, not alternatives

Layer	Function	Field of application
MCP Protokoll	Standardized connection between AI and tools/data sources	Infrastructure level
Skills	Procedural knowledge with step-by-step instructions	Complex workflows
Code-Execution	LLM-generated code for efficient tool usage	Optimized execution

MCP vs A2A Protocol

Different problems, complementary solutions

Aspect	MCP	A2A (Agent-to-Agent)
Purpose	Context for single LLM & data sources	Multi-agent orchestration
Communication	LLM ↔ Tools/Data sources	Agent ↔ Agent
Use-Case	Data access, API integration	Distributed specialized systems
Hierarchy	Client (master) -> Server (worker)	Peer-to-Peer

A2A is the future of AI agents: the focus will no longer be on human-to-AI interaction, but on intelligent multi-agent systems that communicate directly with each other through A2A.

Conclusion & Outlook

Technical Innovation

MCP becomes the universal standard for AI tool integration - the "HTTP of the AI Age"

Exponential Growth

From ~ 100 servers (March 2025) to 6500+ (November 2025) - Network effect in action

Optimization

Code execution and other innovative approaches can reduce token consumption

Enterprise-Ready

OAuth 2.1, Fine-Grained Auth, Standardization - Production-Grade Security

Web resources:

<https://www.anthropic.com/news/model-context-protocol> | <https://www.open-mcp.org/> |

<https://github.com/modelcontextprotocol> | <https://www.claudemcp.com> |

<https://gofastmcp.com/> | <https://fastmcp.cloud> | <https://www.pulsemcp.com> |

<https://registry.modelcontextprotocol.io>



Model Context Protocol

Statistics

Data Sources: MCP Registry (baseline metadata) and GitHub Repos (readme & codebase)

Agentic Workflow: Local LLM gpt-oss:20b for data collection & GitHub analysis

Planning and development: Claude-4.5 and GPT-4.51-codex

Execution: Python analytics stack, frontend: Streamlit dashboard



MCP-Serverregistrierung – Dashboard Phase 3

Gesamtzahl der Server

6488

Analysiert

5997

Analyserate %

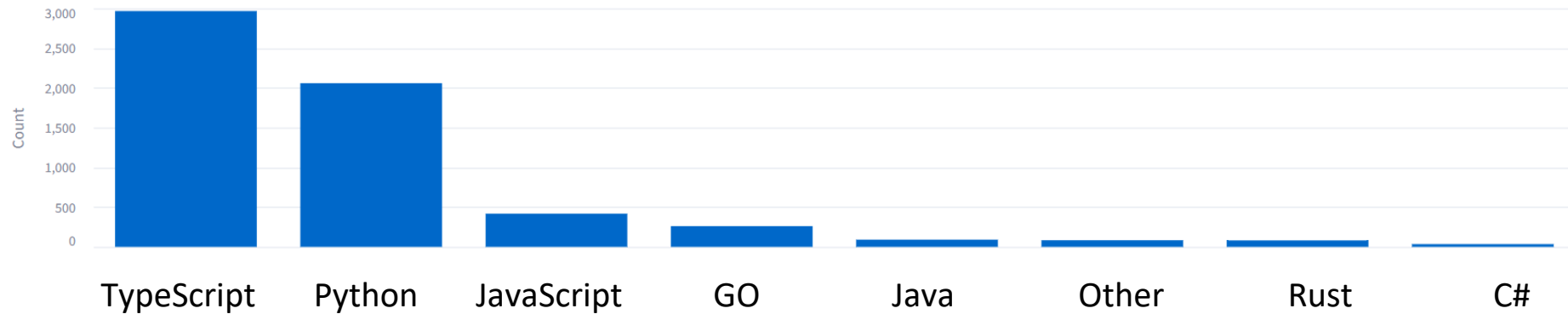
92,43

Generiert am

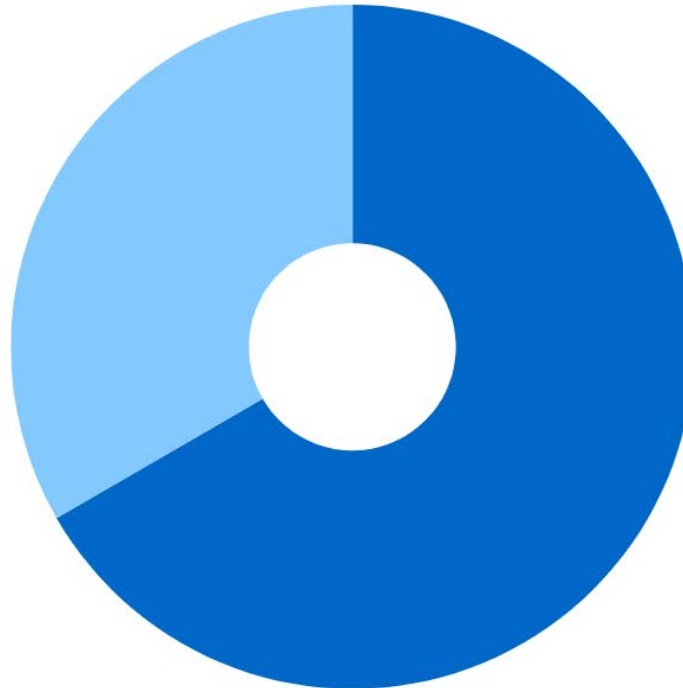
2025-11-17T20:07:0...

[Überblick](#) [Technologie-Einblicke](#) [Top-Server](#) [Werkstatt](#) [Rohdaten](#)

Programmiersprachen



Bereitstellungstyp



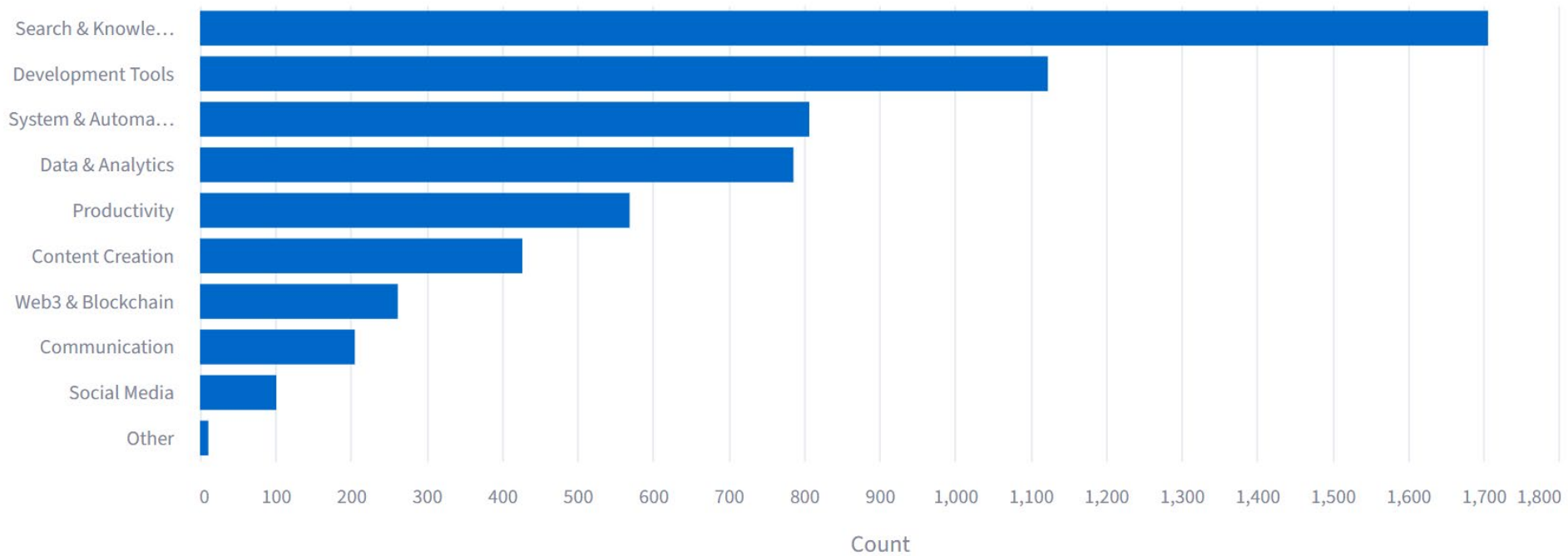
deployment

● local

● remote



Workshop-Kategorien



Technologiefilter

API-Wrapper - RE... x

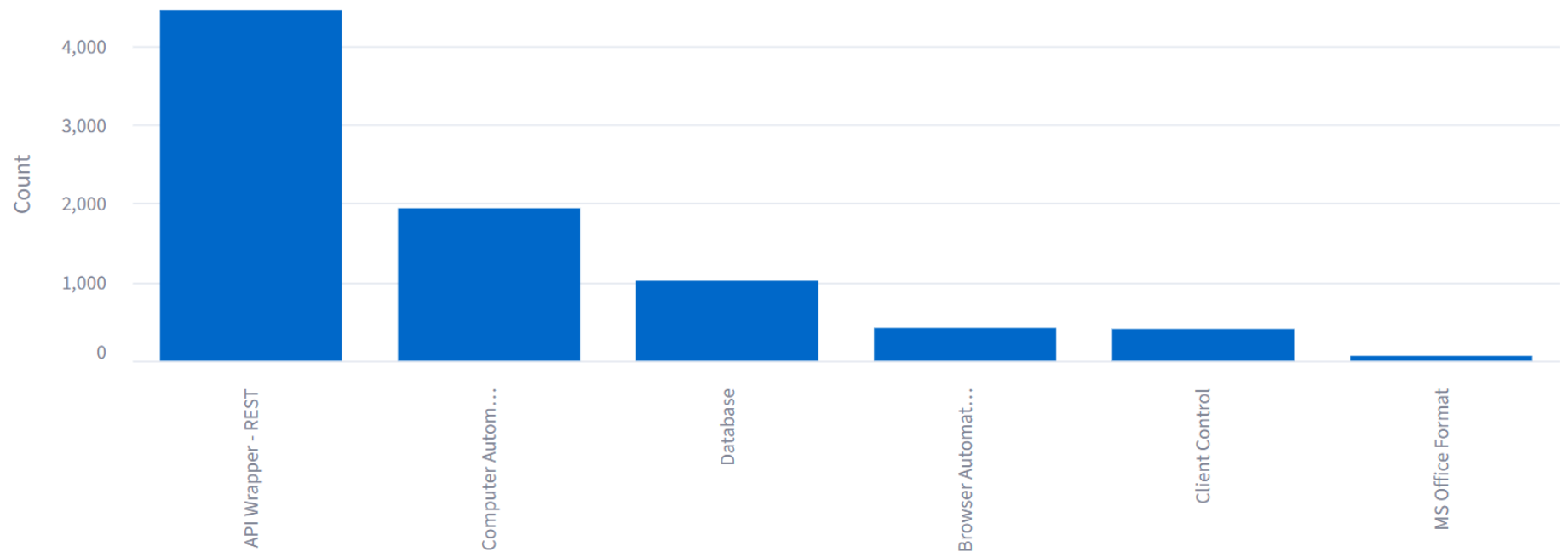
Computerautom... x

Datenbank x

Browserautomat... x

Kundensteuerung x

MS Office-Format x



Die beliebtesten Server nach GitHub-Sternen

Mindeststerne

63474

Workshop-Kategorie

Optionen auswählen

	name	stars	use_case	general_category	te
0	MarkItDown	83016	PDF, Word und PowerPoint Dokumente in strukturierten Markdown umwandeln	Content Creation	
1	Sequential Thinking	72682	Dynamic problem decomposition and iterative solution refinement for complex tasks	Search & Knowledge	
2	Demo (Everything)	72682	MCP Client Testing	Development Tools	
3	Fetch	72682	Web Content Retrieval and Markdown Conversion	Search & Knowledge	
4	Filesystem	72682	Secure, controlled access to a local file system for reading, writing, searching, and ma	System & Automation	
5	Knowledge Graph Memory	72682	Persistente semantische Wissensdatenbank für KI-Agenten, die Informationen speich	Search & Knowledge	

Workshop-Erkunder

Workshop-Kategorie

Daten & Analysen



Top 5 – Daten & Analysen

- Arize Phoenix (7686 ★) – LLM-Prompt-Engineering, Datenexploration und Experimentmanagement über mehrere LLM-Anbieter hinweg
- AWS Cost Analysis (7382 ★) – Analyse der AWS-Kosten für LLM-API-Aufrufe und Erstellung von detaillierten Kostenberichten
- AntV Chart Generator (3163 ★) – Erzeugung von Datenvisualisierungen (Diagramme) aus strukturierten Daten für Dashboards, Berichte oder Präsentationen
- Cloudflare Workers Observability (3137 ★) – Überwachung und Fehlerbehebung von Cloudflare Workers-Bereitstellungen mithilfe von Protokollen, Traces und Metriken



 **Model Context Protocol**

MCP Server Examples

Sequential Thinking MCP: Context Generation

"Thinking" is NOT real intelligence – it is structured context generation

What is Thinking really?

- | | | |
|----------------------|---|----------------------------|
| ✗ Claude thinks | → | ☑ Verbalized thought steps |
| ✗ Agent conversation | → | ☑ Additional context |
| ✗ Server controls | → | ☑ MCP Template Engine |

Why It Works - 5 Key Drivers:

- Structure Forces Completeness - Systematic thinking
- Context Accumulation - Context grows with each step
- Error Detection - Errors become explicit
- Transparency - Traceable
- Refinement Opportunity - Iterative improvement

Computer & Browser Automation

Desktop-Commander-MCP Features

- File System: read, write, search (ripgrep), edit_block (diff-based)
- Terminal: bash commands, interactive processes, session management
- Process Control: start, read output, interact, force terminate
- Search: Smart fuzzy search with character-level diffs
- Security: Sandboxing, allowed directories, blocked commands
- ⚠ Security: Browser/Desktop control requires STRICT sandboxing!

Computer & Browser Automation

AI Assistant controls Browsers!

Playwright MCP (Browser Layer)

- Open source test framework for web applications (Microsoft)
- Purpose: AI-Assistant controls BROWSERS autonomously
- End-to-End testing
- Web scraping automation
- Form filling (UI interactions)
- Performance testing
- Monitoring

-> See also AI browsers such as Comet (Perplexity) and Atlas (OpenAI)

NotebookLM MCP: The Browser Automation Hack

Integration without API - User Behavior Emulation with Playwright

The Problem

- No public API
- Automation seems blocked
- Normal integration: IMPOSSIBLE

The Solution

- Use web UI instead of API
- Imitate a normal user
- Browser Automation + Playwright



NotebookLM MCP: Automation Hack

Layer 1: MCP JSON-RPC (MCP Client ↔ Server)

Layer 2: MCP Server (Orchestration)

Layer 3: Playwright Browser (Automation)

Layer 4: NotebookLM + Gemini (Backend)

5 Critical Techniques

- Session Persistence - Save cookies, no re-login
- Stealth Mode - Bot detection bypass
- Human-like Behavior - typing delay, mouse movement
- Smart DOM Navigation - waitForSelector() for robust search
- Intelligent Polling - Wait for response with timeout

Insight:

TRADITIONAL	„I need an API to integrate“
NOTEBOOKLM MCP	„I just need a Browser“

→ **EVERYTHING with web UI is integratable!**

The Complete MCP Spectrum: 6 Patterns

From Data to Desktop - The Complete Automation Stack

The Automation Pyramid

Level 6: Windows MCP (Desktop)

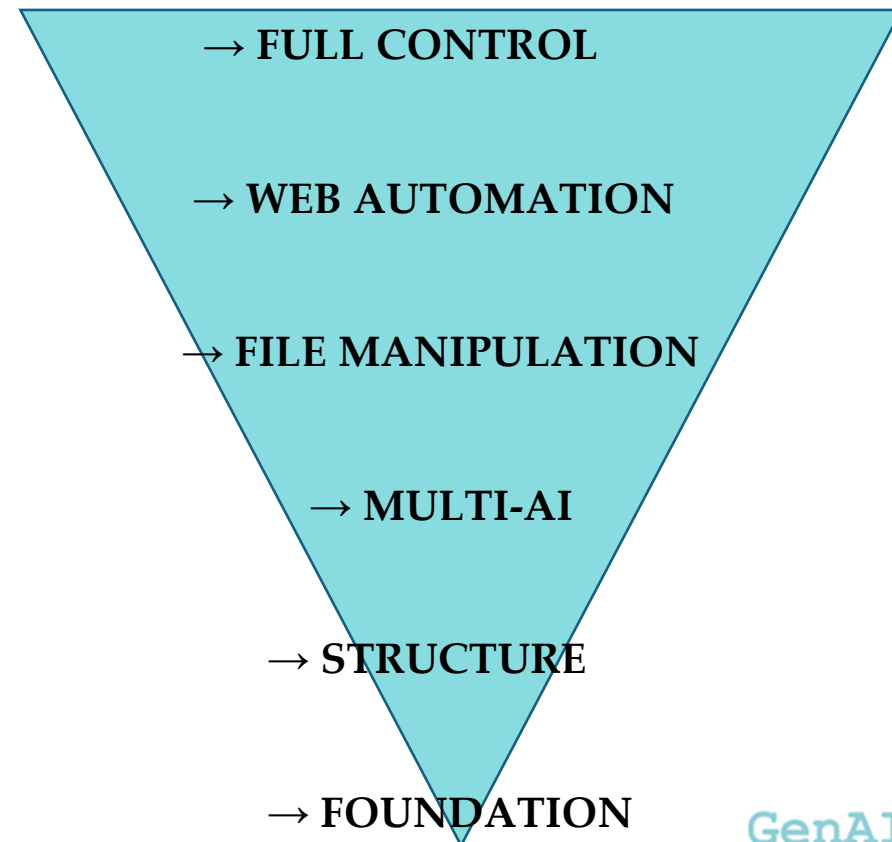
Level 5: Playwright MCP (Browser)

Level 4: Excel MCP (Files)

Level 3: NotebookLM MCP (AI)

Level 2: Sequential Thinking

Level 1: Standard MCP (Tools/Data)



My AI Experiments Lab with MCP Server

- MCP Server as cloud and local LLM API wrapper with session management and memory handling
- MCP Server for "Agent Brain". Vector store embeddings with semantic and metadata search of memories with data hygiene
- MCP and Relay (Post) Server for Agent2Agent communication (claude2claude, 2 instances of the desktop app)
- MCP Server for calling LLM CLI assistants (Claude Code CLI, Kimi CLI, Gemini CLI)

**Claude Desk 1
Orchestrator**

MCP
1



Relay
Queue

MCP
2

**Claude Desk 2
worker (polling)**

Aldersbach Monastery Research Lab

Multi-Component Research System

- MCP Server for Transkribus HTR API
- Access to digitized Aldersbach monastery HTR collections
- 32 specialized tools for document search and analysis
- Smart caching and rate limiting

Docker ML Container

- Named Entity Recognition with ensemble voting (Flair HIPE-2022, SpaCy de_core_news_sm, GLiNER), ELECTRA model for newspaper analysis
- Lazy loading: Models load on demand

SQLite Reference Database

- Biographical data on abbots, monks, employees as well as historical place names
- Fuzzy matching with Levenshtein distance for ML processes

Qdrant Vector Store MCP Server

- Aldersbach monastery charters and literature
- Semantic search capabilities for historical documents



ML Analysis in Practice: Understanding Named Entity Recognition

What ML Does: Automatic Information Extraction

Example Historical Text (from Transkribus):

“In 1650, Johann Schmidt was appointed Prior of Aldersbach Monastery”

Without ML (Traditional Search):

- ✗ You must know exactly what to search for
- ✗ Only finds exact text matches
- ✗ Result: Document IDs with keyword hits

With ML Named Entity Recognition:

- ☑ Automatically identifies ALL entities in the text:
 - Person: "Johann Schmidt" (Confidence: 88%)
 - Title: "Prior" (Confidence: 92%)
 - Location: "Aldersbach Monastery" (Confidence: 85%)
 - Date: "1650" (Confidence: 95%)

ML Service Integration in TK MCP Server: From API to NER Analysis

4-Layer Processing Pipeline

1. MCP Wrapper Layer (transkribus_server.py)
 - 32 @mcp.tool() definitions with logging and error handling
2. Tools Layer (ml_analysis.py)
 - Fetches transcription from Transkribus API (PAGE XML format)
 - Converts XML to plain text
 - HTTP POST to ML service
3. ML Service Layer
 - Ensemble NER with voting system
 - Majority vote + average confidence → Final prediction
4. Response Aggregation
 - Combines API metadata with ML results
 - Returns: persons, locations, organizations with confidence scores

ML Analysis in Practice: Understanding Named Entity Recognition

There are two separate workflows:

Workflow A: Normal Search (WITHOUT ML)

1. User asks Claude: "Search for documents about 'Prior'"
2. MCP Server → Transkribus API: Standard Full-Text Search
3. Transkribus responds: List of documents containing "Prior"
4. MCP Server → Claude context window: "Found: Doc 123, Doc 456..."
5. DONE - No ML involved

Workflow B: ML Analysis (EXPLICITLY requested)

1. User asks Claude: "Analyze document 123 with ML"
2. MCP Server FIRST fetches the transcription of Doc 123
3. MCP Server sends text to ML Service (localhost:8001)
4. ML Service analyzes → identifies persons, locations, etc.
5. MCP Server combines results
6. Claude context window: "Found: 3 persons (Johann Schmidt...), 2 locations..."



Model Context Protocol

robert.klugseder@oeaw.ac.at

and

 Claude