ChatGPT

Anthropics Model Context Protocol (MCP) – Überblick und Tools

1. Überblick: Was ist das Model Context Protocol (MCP)?

Das **Model Context Protocol (MCP)** ist ein offener Standard von Anthropic, der die Interaktion zwischen KI-Modellen (insbesondere LLMs) und externen Tools oder Datenquellen vereinheitlicht [1] [2]. Statt für jede Kombination aus Modell und Tool individuelle Integrationen zu schreiben, definiert MCP eine einheitliche **Client-Server-Architektur** ähnlich der *Language Server Protocol (LSP)*-Architektur aus der Softwareentwicklung [3] [4]. Ein KI-Assistent (der *Client*, z.B. in einer IDE oder App) kann über MCP mit beliebigen *Servern* kommunizieren, die Tools, Daten oder Dienste bereitstellen [5] [6]. Dadurch entsteht eine „gemeinsame Sprache" zwischen Modellen und externen Systemen – vergleichbar mit einem universellen Übersetzer für KI [7].

MCP-Architektur: Eine Host-Anwendung (z.B. Claude Desktop oder eine IDE) enthält einen MCP-Client, der sich mit lokalen oder entfernten MCP-Servern verbindet. Diese Server bieten Tools, Datenquellen oder vordefinierte Prompts an, welche das KI-Modell nutzen kann [8] [9]. Der Datenaustausch erfolgt standardisiert über JSON-RPC 2.0 und kann lokal (STDIO) oder per HTTP+SSE über das Netzwerk laufen [10] [11].

Kernideen und Merkmale von MCP:

- *Standardisierte Schnittstelle:* Alle Werkzeuge stellen sich dem Modell mit einem **einheitlichen Eingabe-/Ausgabe-Schema** vor. Aufrufe folgen JSON-RPC 2.0 über eine REST-ähnliche Schnittstelle [10]. Das erleichtert die Integration verschiedener Tool-APIs und beseitigt Unterschiede zwischen Anbietern [12]. So werden KI-Modelle **anbieterneutral** angesteuert – MCP reduziert *Vendor Lock-in*, da Entwickler nicht mehr jede API (OpenAI, Anthropic, lokale Modelle etc.) separat ansprechen müssen [12] [13].

- *Kontext und Funktionserweiterung:* MCP ermöglicht **zweiwege**-Kommunikation: Das Modell kann Tools aufrufen und erhält strukturierte Ergebnisse zurück [1] [14]. Es gibt drei Arten von Fähigkeiten, die ein MCP-Server bereitstellen kann: **Tools** (ausführbare Funktionen wie `get_weather` oder `runTests`), **Resources** (Daten/Dateien, die als Kontext bereitgestellt werden) und **Prompts** (vordefinierte Eingabevorlagen, die das Verhalten des Modells steuern) [15] [16]. Dadurch lassen sich LLMs deutlich **kontextreicher und aktionsfähiger** machen, da sie sicher auf aktuelle Daten zugreifen und Aktionen durchführen können.

- *Reduktion des Integrationsaufwands:* Vor MCP musste für *jedes* Modell–Tool-Paar eine eigene Integration gebaut werden, was zu einem unüberschaubaren **N×M-Problem** führte. MCP reduziert dies zum **M+N-Problem**: Man entwickelt *M* verschiedene Clients (für verschiedene Apps/LLMs) und *N* Server (für verschiedene Tools), anstatt M×N Integrationen [17]. Die Standardisierung spart enorm Aufwand und erhöht die **Interoperabilität** in komplexen Systemlandschaften [18].

Integration vereinheitlicht: Links: Ohne MCP sind für 3 AI-Anwendungen und 3 Datenquellen 9 spezifische Integrationen nötig (jeder Pfeil). Rechts: Mit MCP genügen 3 MCP-Server und 3 MCP-Clients (insgesamt 6 Implementierungen), da alle nach dem gleichen Protokoll kommunizieren [17].

- *Offener Standard und Akzeptanz:* Anthropic veröffentlichte MCP im **November 2024** als Open-Source-Standard [19] [20]. Seitdem entstand eine aktive Open-Source-Community – die offizielle MCP-Server-Sammlung hat zehntausende Stars auf GitHub [21] [22]. Ein wichtiger Meilenstein war **März 2025**, als Konkurrent OpenAI ankündigte, MCP für seine Modelle zu unterstützen [23]. Damit entwickelt sich MCP rasch zum Industriestandard für KI-Tool-Integrationen. Zahlreiche Unternehmen (z.B. Block, Apollo) und Entwickler-Tools (Zed, Replit, Codeium, Sourcegraph) integrieren MCP bereits in ihre Plattformen [24].

- *Technische Details:* MCP-Nachrichten sind JSON-RPC-Objekte, welche vom Modell per HTTP(S) an den Server gesendet werden. Ein typischer **Tool-Aufruf** besteht aus einem JSON mit Methode `"tools/call"`, dem Tool-Namen und Arguments [25]. Der Server antwortet mit einem JSON, das im `"result"` die Ausgabe enthält (z.B. Text, JSON-Daten etc.) [26]. Die Kommunikation kann **streaming** (Server-Sent Events für Echtzeit-Updates) oder synchron erfolgen [27]. Für lokale Tools (z.B. auf dem gleichen Rechner) kann der STDIO-Modus genutzt werden, um Latenz zu minimieren [28]. Wichtig ist, dass alle Tools dem Modell ihre Schnittstelle in standardisierter Form bekanntgeben (Name, Parameter, Beschreibung), damit das Modell weiß, *wie* es sie benutzen kann.

Zusammengefasst schafft MCP somit ein **einheitliches „Grammar" für KI-gestützte Anwendungen** [29]. Dies erlaubt es verschiedenen KI-Agenten und Anwendungen, sich mit denselben Tools und Datenquellen zu verbinden, ohne jedes Mal das Rad neu zu erfinden. Die KI-Assistenten werden dadurch über ihre Trainingsdaten hinaus nützlich: Sie können aktuelle Unternehmensdaten abrufen, Code aus Repositorien lesen, Web-Suchen durchführen, Befehle am System ausführen u.v.m., alles kontrolliert über ein Protokoll [30] [2].

2. Spezialisierte MCP-Tools für Programmierung und Entwicklung

Ein Hauptanwendungsfeld von MCP ist die **KI-Unterstützung für Entwickler** – oft als "AI coding assistant" oder *vibe coding* bezeichnet [31]. Hierbei nutzen LLMs das Protokoll, um direkt in Entwicklungsumgebungen einzugreifen: Code lesen, schreiben, testen, dokumentieren und Entwicklungs-Workflows automatisieren. In den letzten Monaten sind zahlreiche MCP-Server entstanden, die speziell auf die Bedürfnisse von Programmierern zugeschnitten sind:

- **Dateisystem- und Code-Zugriff:** Der *Filesystem MCP Server* ermöglicht es einem KI-Assistenten, **lokale Projektdateien** zu durchsuchen, zu lesen und auch zu verändern – mit strikten Sicherheitskontrollen (z.B. nur schreibend in erlaubten Ordnern) [32] [33]. So kann das Modell etwa Quellcode-Dateien öffnen, Codeblöcke daraus als Kontext ziehen oder neue Codezeilen in eine Datei einfügen. Typische Funktionen sind etwa Dateien lesen (`read_file`), schreiben/erstellen (`write_file`), im Code nach Mustern suchen (`search_content`), Verzeichnisse auflisten (`directory_tree`) oder Code-Snippets gezielt ersetzen [34]. Entwickler nutzen dies, um z.B. per Sprachkommando "Öffne mir die Funktion XY" ausführen zu lassen oder Boilerplate-Code automatisch in einer Datei zu generieren. Die *Everything*-Referenzimplementierung von Anthropic demonstriert viele solche Tools in Aktion [35].

- **Code-Analyse und LSP-Integration:** Um ein besseres Verständnis für den Code zu erlangen, können KI-Modelle auch klassische Sprach-Analyse-Tools nutzen. Ein Beispiel ist der *MCP*

Language Server [36] : Dieser MCP-Server koppelt einen normalen *Language Server* (wie `gopls` für Go, `rust-analyzer` für Rust, `pyright` für Python etc.) ein. Damit bekommt das LLM semantische **IDE-Funktionen** an die Hand – es kann z.B. "Gehe zur Definition dieser Methode" ausführen oder Compiler-Diagnosemeldungen abfragen [36] [37] . Das Modell ruft entsprechende Tools (z.B. `find_definition`, `list_references`, `get_diagnostics`) über MCP auf, der Language-Server liefert die Ergebnisse zurück (z.B. Fundstelle im Code, Fehlermeldungen). So können KI-Assistenten konsistent komplexe Codebasen navigieren und tiefergehende Analysen durchführen, ähnlich wie ein Entwickler es in VS Code per LSP tut.

- **Repository- und Versionskontrolle:** Für die **Verwaltung von Git-Repositorien** gibt es offizielle MCP-Server von Diensten wie GitHub und GitLab. Der *GitHub MCP Server* etwa ermöglicht es, Repos zu durchsuchen, Commits oder Branches anzulegen, Pull Requests zu prüfen oder Issues abzurufen – alles über KI-Befehle [38] [39] . Ein KI-Assistent könnte somit z.B. auf Anweisung hin einen neuen Branch erstellen und Codeänderungen commiten. Unternehmen nutzen dies, um Routineaufgaben in der Codeverwaltung zu automatisieren. Ähnlich existiert ein GitLab-Server für projektbezogene Aktionen [40] . Diese Tools machen die KI zum "Repo-Assistenten", der Änderungen nicht nur vorschlagen, sondern direkt im Code-Repo umsetzen kann (mit entsprechender Kontrolle).

- **Ausführen und Testen von Code:** Ein anderer wichtiger Baustein ist der *Docker MCP Server*. Dieser dient als **sicheres Sandbox-Umfeld**, in dem die KI Code ausführen oder Programme starten kann, ohne das Host-System zu gefährden [41] [42] . Der Server stellt Tools bereit, um Container zu starten, Befehle darin laufen zu lassen und Ergebnisse zurückzugeben. So kann ein Modell z.B. Test-Skripte in einer isolierten Docker-Umgebung ausführen und dem Entwickler die Ausgabe oder Fehler zurückmelden – wertvoll fürs automatisierte Testen oder Bauen von Software. Dabei bietet der Docker-Server komfortable Funktionen wie Container auflisten, neue Container mit gewünschtem Image + Abhängigkeiten starten, Skripte im Container ausführen und nach Nutzung wieder aufräumen [43] [44] . Die KI kann so in *jedem* beliebigen Tech-Stack arbeiten, da für nahezu jede Sprache ein Docker-Image bereitsteht (Python, Node, Java, etc.), was truly polyglotte Entwicklungsassistenzen ermöglicht [45] [46] .

- **API-Dokumentationen und externe Libraries:** Speziell für API-zentrische Entwicklung existieren Tools wie *Apidog MCP Server*. Apidog nimmt komplette API-Dokumentationen (z.B. OpenAPI/Swagger Specs) und macht sie für das LLM **abfragbar**. Das Modell kann Fragen stellen wie "Wie heißt der Parameter für Endpoint X?" oder "Gib ein Beispiel für den Payload dieser API" – der MCP-Server durchsucht die API-Doku und liefert die Antwort [47] . Dies beschleunigt die Integration externer Services erheblich, da Entwickler nicht mehr manuell in Doku wühlen müssen: Die KI zieht sich benötigte Infos selbst aus den bereitgestellten Specs und Code-Beispielen.

- **Autonome Code-Agenten:** Ein besonders umfangreiches Beispiel ist **Hanzo MCP** (GitHub: `hanzoai/mcp`). Dieser MCP-Server bündelt zahlreiche **Code-Management-Funktionen** zu einem mächtigen Entwicklungsassistenten für Claude & Co. [48] . Hanzo MCP erlaubt dem KI-Modell, komplette Projekte zu analysieren, relevante Dateien zu finden, Änderungen an Code vorzunehmen und sogar Teilaufgaben an spezialisierte Sub-Agenten zu delegieren [49] [50] . Die Features umfassen u.a.: *Code verstehen* (Projektstruktur scannen, Patterns finden), *Code modifizieren* (gezielte Einfügungen/Löschungen in Dateien mit Berechtigungsabfrage), *Dateioperationen* (Dateien/Ordner anlegen, verschieben, löschen mit `run_command`), *Content-Suche und -Ersetzung* (Refactoring bestimmter Muster), *Projekt-Analyse* (Abhängigkeiten, Build-Skripte erkennen) und *Befehlsausführung* für Tests oder Builds [49] [34] . All das geschieht in einem kontrollierten Rahmen – Hanzo MCP verlangt z.B. explizite Bestätigung, bevor es eine Datei

überschreibt, und begrenzt den Zugriff auf definierte Verzeichnisse [51] [52]. Mit solchen Tools kann ein LLM tatsächlich als *künstlicher Pair-Programmierer* agieren, der Codeänderungen vorschlägt **und** auf Wunsch direkt umsetzt (innerhalb definierter Grenzen). Entwickler berichten, dass dies besonders bei großen Refactorings oder repetitiven Änderungen enorme Zeitersparnis bringt, weil die KI viele kleine Änderungen automatisch durchführen kann [49] [50].

- **Mehrere LLM-Provider nutzen:** Da MCP die **Modell-APIs abstrahiert**, können Entwickler je nach Aufgabe verschiedene Sprachmodelle einsetzen. Viele MCP-Clients (z.B. Claude Desktop, Cursor IDE) unterstützen gängige Anbieter über Plugins oder Bridges. Es gibt Projekte wie *LiteLLM*, die Dutzende Model-Backends unter einer Schnittstelle vereinen – so lässt sich z.B. OpenAI GPT-4, Anthropic Claude 2 oder ein lokales Mistral-Modell gleichermaßen einbinden [53]. In der Praxis könnte man z.B. GPT-4 als Hauptmodell nutzen, aber bestimmte Tools an ein schnelleres lokales Modell delegieren. MCP hält Eingabe/Output dabei so standardisiert, dass die Umstellung eines Modells keine Änderung der Tool-Integration erfordert [13]. Für Entwickler mit API-Schlüsseln zu vielen LLMs (wie in der Anfrage erwähnt) bedeutet dies maximale Flexibilität ohne zusätzlichen Code-Aufwand.

Integration in IDEs und Workflows: MCP-Tools für Entwickler werden meist über spezielle Clients in IDEs oder Editor-Apps aktiv. Zwei prominente Beispiele sind *Claude Desktop* (Anthropics eigene Desktop-App für Claude) und *Cursor* (eine AI-gestützte Code-IDE). In beiden kann man MCP-Server konfigurieren und aktivieren:

- In **Claude Desktop** genügt es oft, den MCP-Server zu installieren und der App bekannt zu machen. Hanzo MCP z.B. bietet einen Befehl `hanzo-mcp --install`, der den Server in Claude Desktop registriert [54] [55]. Anschließend kann man in der Claude-GUI den neuen "Hanzo"-Server aus einem Dropdown auswählen und Claude hat sofort Zugriff auf dessen Tools [56]. Claude Desktop kommt bereits mit integriertem Support für lokale MCP-Server, sodass Entwickler z.B. ihren Code-Ordner freigeben oder einen GitHub-Token hinterlegen können, um die entsprechenden Server (Filesystem, GitHub etc.) zu nutzen [16] [57].

- In **Cursor** (und ähnlichen IDEs) lassen sich MCP-Server ebenfalls leicht hinzufügen. Über die Einstellungen (`.cursor/mcp.json` oder Settings-Menü) kann man neue Server eintragen, einen Namen vergeben und den Startbefehl angeben [58] [59]. Beispiel: Um den BraveSearch-Server in Cursor zu nutzen, fügt man einen Eintrag hinzu, der intern den Befehl `npx @modelcontextprotocol/server-brave-search` mit dem eigenen API-Key startet [60]. Cursor zeigt aktive Server mit einem grünen Indikator und listet die verfügbaren Tools auf [61]. Der eingebaute "Composer Agent" entscheidet dann automatisch, wann welches Tool sinnvoll ist, und fragt im Kontext des Code-Gesprächs z.B. bei Bedarf den Such-Server oder den Filesystem-Server an [62]. So verschmelzen die Tools mit dem normalen Coding-Chatflow.

  Tipp: Bei Windows-Entwicklungsumgebungen (z.B. Nutzung von WSL unter Windows) gibt es ein paar Besonderheiten. Einige MCP-CLI-Befehle müssen mit `cmd /c` als Präfix gestartet werden, damit kein *Client Closed*-Fehler auftritt, und erforderliche Abhängigkeiten (Node, Python etc.) sollten in der Windows-Umgebung verfügbar sein, nicht nur in WSL [63]. In Cursor etwa empfiehlt es sich, Node.js auch auf Windows zu installieren (falls der MCP-Server als Windows-Prozess laufen soll), da sonst die Kommunikation abbricht [64]. Solche Details sind in den Dokus vieler MCP-Projekte vermerkt, um plattformbedingte Hürden zu überwinden.

Zusammengefasst eröffnen MCP-Tools Entwicklern eine **neue Dimension der Automatisierung**. Routineaufgaben (Suche, Codeänderungen, Tests) können direkt vom KI-Assistenten durchgeführt werden, während der Entwickler den höheren Kontext vorgibt. Wichtig ist, dass Sicherheit und Kontrolle dabei erhalten bleiben – durch Berechtigungsabfragen, Zugriffslimits und Logging. Viele Entwickler berichten bereits, dass KI-Copiloten mit MCP-Unterstützung ihnen erlauben, sich auf die kreativen Teile des Codens zu konzentrieren, während repetitive oder informationslastige Schritte von der KI erledigt werden [51] [65].

3. Allgemeine Möglichkeiten und Projekte mit MCP

Über die Programmierung hinaus hat MCP ein **breites Spektrum an Anwendungen**. Prinzipiell kann jedes System mit einer API oder Datenquelle als MCP-Server dienen. Im Folgenden einige wichtige Einsatzgebiete und Projekte, die zeigen, *was heute schon alles mit MCP möglich ist*:

- **Team-Kommunikation und DevOps:** Der *Slack MCP Server* integriert KI in die alltägliche Kommunikationsplattform vieler Entwickler-Teams. Damit kann ein KI-Assistent in Slack-Channels Nachrichten lesen und schreiben, Dateien posten oder sogar Channel-Verwaltung übernehmen [66] [67]. So sind z.B. **automatisierte Benachrichtigungen** ein Anwendungsfall: Build- und Deployment-Pipelines senden ihre CI/CD-Statusmeldungen an die KI, die dann in Echtzeit in den richtigen Slack-Channel gepostet werden [68]. Oder das Modell terminiert via `reminders.add` regelmäßige Erinnerungen für Meetings/Deadlines [68]. Auch komplexere Workflows sind denkbar – etwa könnte die KI bei bestimmten Alarmmeldungen automatisch einen Thread starten und Lösungsansätze vorschlagen. Slack-Server bieten dafür Tools wie *Channel Management* (Channels anlegen, archivieren, Mitgliedslisten anzeigen) und *Message Ops* (Nachrichten planen, reagieren, Threads antworten) [69]. Dies erweitert Slack zum **AI-kollaborativen Workspace**, wo die KI wie ein Teammitglied agieren kann (z.B. Fragen beantworten anhand vergangener Diskussionen, dank Vektor-Suche im Channel-Verlauf) [70]. Viele dieser Integrationen werden offiziell von Slack unterstützt und nutzen sichere Bot-Tokens und OAuth, um den Zugang streng zu kontrollieren [71].

- **Web-Suche und Informationen:** Mit MCP können LLMs selber im Web recherchieren. Der *Brave Search MCP Server* bietet beispielsweise datenschutzfreundliche Websuchen an [72]. Ein KI-Modell kann damit eine Internet-Suche durchführen, um aktuelle Informationen oder Dokumentationen zu finden – wichtig, da LLMs allein ja nur ihr Trainingswissen bis zu einem Stichtag haben. Die Brave-Integration erlaubt es, Suchanfragen mit Parametern zu verfeinern (Seitenzahl, Frische, Dateitypen etc.), was besonders für **technische Recherchen** nützlich ist [73]. So könnte ein Entwickler fragen: *"Finde die neueste Doku zu React Hooks"* – das LLM nutzt Brave Search und bekommt relevante Ergebnisse zurück, die es dem Nutzer präsentiert oder ins Coding-Kontextfenster einarbeitet. Neben Brave existieren auch *Google Search MCP Server* oder spezielle Varianten für Dokumentationssuchen. Diese Tools sorgen dafür, dass KI-Assistenten stets auf dem aktuellen Stand bleiben und **Live-Recherchen** durchführen können, ohne dass der Nutzer die Anwendung verlassen muss.

- **Datenbanken und Wissensabfragen:** Mehrere MCP-Server zielen darauf ab, **Datenbankinhalte natürlichsprachlich zugänglich** zu machen. Ein Beispiel ist der *PostgreSQL MCP Server*, der LLMs erlaubt, Lesezugriff auf SQL-Datenbanken zu nehmen [74]. Die KI kann dann etwa Fragen beantworten wie *"Wie viele Nutzer haben wir letzten Monat gewonnen?"* – der Server wandelt dies im Hintergrund in SQL um, fragt die DB ab und gibt das Ergebnis als strukturiertes Resultat zurück. Ähnlich gibt es Server für *SQLite*, *Redis* (Key-Value-Stores abfragen) oder *Graph-Datenbanken*. In Unternehmenskontexten spielt das eine große Rolle: Man kann

einem KI-Chatbot sicheren Read-Only-Zugriff auf bestimmte Tabellen geben, wodurch er komplexe Analytics-Fragen beantworten kann, ohne dass der Nutzer SQL beherrschen muss. Selbst für **Vektor-Datenbanken** (z.B. Qdrant, Pinecone) existieren MCP-Adapter – damit lässt sich semantische Suche implementieren, etwa um in einem Dokumentenbestand ähnliche Inhalte zu finden (nützlich für Chatbot-Gedächtnis) [75] [76].

- **Cloud- und IT-Infrastruktur:** Auch in DevOps und Cloud Operations wird MCP erprobt. Ein hervorstechendes Beispiel ist Cloudflares offizieller *Cloudflare MCP Server*. Dieser erlaubt KI-Modellen, über Cloudflare-APIs Dinge wie **DNS-Einträge, CDN-Cache oder Sicherheitsregeln** zu verwalten [77]. So könnte ein KI-Assistent auf Anfrage einen neuen DNS-Record für eine Domain anlegen oder bösartigen Traffic über WAF-Regeln blockieren [77]. Durch die Einbindung in Cloudflares Edge-Netzwerk können solche MCP-Server weltweit verteilt laufen und mit geringster Latenz arbeiten [78] [79]. Man sieht hier, dass MCP nicht auf die Entwickler-Welt beschränkt ist – er fungiert generell als **Brücke zwischen KI und SaaS/Cloud-Diensten**. Ähnlich hat z.B. AWS in einem Blog gezeigt, wie MCP-Server genutzt werden können, um KI Zugriff auf AWS-Services zu geben (S3-Buckets, DynamoDB, CloudWatch-Logs etc.), ohne für jeden Service eine separate Integration zu schreiben [80] [81]. Die KI kann somit mit einer einzigen standardisierten Methode quer über viele Systeme operieren, wobei überall die bestehenden Sicherheitsmechanismen (API-Keys, IAM-Rollen usw.) greifen [82].

- **Dateidienste und Knowledge-Management:** Neben dem bereits erwähnten lokalen Filesystem-Server gibt es auch Adapter für Cloud-Dateispeicher wie *Google Drive MCP Server* [83]. Damit kann ein Unternehmensassistent z.B. Firmendokumente auf Google Drive durchsuchen, relevante PDFs lesen und Inhalte zitieren – alles auf Anfrage des Nutzers und ohne manuelles Öffnen der Drive-Oberfläche. Ein weiterer interessanter Server ist *Memory (Knowledge Graph)* [84]: Dieser bietet dem KI-Modell eine Art externes Gedächtnis, in dem Fakten als Wissensgraf gespeichert und abgefragt werden können (oft mittels Vektordatenbank unter der Haube). So lassen sich langfristige Informationen oder Benutzerpräferenzen persistieren, die das Modell ansonsten über mehrere Sessions vergessen würde.

- **Weitere Community-Projekte:** Die MCP-Community ist sehr aktiv, es gibt nahezu täglich neue Server-Projekte auf GitHub. Einige Beispiele:

    - *EverArt MCP*: AI-gestützte **Bildgenerierung** – die KI kann auf Zuruf Grafiken erstellen lassen, indem der Server Stable Diffusion, DALL-E oder ähnliche Modelle ansteuert [85].
    - *Puppeteer MCP*: Web-Browsing und **Web-Scraping** – erlaubt dem Modell, eine Website aufzurufen und DOM-Elemente auszulesen oder Screenshots zu machen [86]. Nützlich etwa, um automatisch Daten von einer Webpage zu extrahieren, auf die kein API-Zugriff existiert.
    - *Sentry MCP*: Integration mit Sentry.io – ein KI-DevOps-Assistent könnte aktuelle Fehlerlogs oder Metriken abrufen, zusammenfassen und Lösungen vorschlagen [87].
    - *Time MCP*: Bietet der KI einfache Fähigkeiten wie Zeitzonen umrechnen, aktuelle Uhrzeit/Datum etc., damit solche Fragen ohne externen API-Aufruf beantwortet werden können [88].
    - *AgentRPC MCP*: Ein Projekt, das MCP mit Remote Procedure Calls verbindet – damit kann die KI theoretisch **beliebigen Code in Fremdsprachen oder auf anderen Maschinen** ausführen, indem der Server entsprechende RPC-Schnittstellen bereitstellt [89] [90].

Diese Vielfalt an Servern zeigt, dass MCP zu einem **generischen Werkzeugkasten** für KI-Funktionalität heranwächst. Dank des offenen Ökosystems teilen Unternehmen ihre MCP-Server als Open Source, und es gibt Plattformen (z.B. *mcp.so* oder Verzeichnisse wie *mcpmarket.com*), die verfügbare MCP-Services

auflisten und den Austausch fördern. Die Standardisierung fördert zudem offizielle Integrationen: GitHub selbst hat seinen MCP-Server in Go geschrieben und optimiert (schneller als der Python-Referenzcode) [91] [92], Slack und Cloudflare stellen offizielle Server bereit, und sogar Branchenspezifische Tools (wie Alibaba Cloud AnalyticDB, siehe Anthropic-Repo) tauchen auf [93] [94].

> **Sicherheit und Governance:** Ein zentrales Anliegen bei all diesen Möglichkeiten ist die Sicherheit. Da MCP-Server brücken zu potenziell sensiblen Systemen schlagen, wurden von Anfang an Mechanismen vorgesehen. Jeder MCP-Server läuft unter definierten Berechtigungen – z.B. begrenzte Ordnerzugriffe, nur Lesezugriff auf DB, API-Tokens mit begrenzten Scopes. Viele Server fordern eine Bestätigung vom Nutzer, bevor destruktive Aktionen ausgeführt werden (etwa Datei löschen), oder bauen eingebauten Schutz ein (siehe Filesystem-Server: erlaubte Dateitypen, Pfad-Whitelist [65]). Durch den Client-Server-Ansatz lässt sich auch zentral protokollieren, welche Aktionen das Modell anfordert, was für Audits hilfreich ist. Mit der wachsenden Verbreitung werden Best Practices entwickelt – inklusive Authentifizierung (OAuth für externe Dienste), Rate Limits, und "YOLO-Modes" nur auf ausdrücklichen Wunsch [95]. Unternehmen können so MCP einsetzen, ohne die Kontrolle über ihre Daten zu verlieren, weil sie genau festlegen, **welche Türen für die KI offen sind**.

Fazit: Das Model Context Protocol erweitert KI-Systeme von isolierten Chatbots zu **mächtigen, kontextbewussten Assistenten**. Mit MCP können Modelle Informationen in Echtzeit einholen, Aktionen in der digitalen Welt durchführen und sich nahtlos in bestehende Workflows einbetten. Insbesondere im Entwicklungsbereich sehen wir bereits, wie MCP die Produktivität steigert – von automatisierten Code-Änderungen bis hin zu KI-gestützten Deployment-Prozessen. Doch auch generell lässt MCP die Grenzen dessen verschwimmen, was KI leisten kann, da praktisch jede API und jeder Datenbestand andockfähig wird. Die aktuelle Projektsammlung liefert einen eindrucksvollen *State of the Art* Überblick: ob **Coding, DevOps, Suche, Datenanalyse oder Office-Aufgaben**, für viele Domänen gibt es schon spezialisierte MCP-Tools. Mit der starken Unterstützung durch die Community und große Player (Anthropic, OpenAI, AWS u.a.) dürfte MCP sich weiter etablieren. Wir stehen am Anfang einer Ära, in der KI-Assistenten dank einheitlicher Protokolle tatsächlich *Teil unserer Software-Infrastruktur* werden – interoperabel, erweiterbar und kontrollierbar [96] [97].

Quellen: Die obigen Informationen wurden aus einer Vielzahl von aktuellen Artikeln, Blogs und Dokumentationen zusammengetragen, darunter offizielle Ankündigungen von Anthropic [20] [2], technische Analysen aus Entwickler-Blogs [4] [98], Praxisberichte (z.B. zu Hanzo MCP [48] [34]) sowie Community-Beiträge, welche die Top-MCP-Server 2025 detailliert vorstellen (Dev.to) [99] [33]. Diese Referenzen sind im Text verlinkt und bieten weiterführende Einblicke in die Funktionsweise und Anwendung von MCP.

---

[1] [10] [11] [12] [14] [25] [26] [27] [29] [98] MCP, A2A, ACP: What does it all mean?
https://akka.io/blog/mcp-a2a-acp-what-does-it-all-mean

[2] [6] [16] [20] [24] [30] [57] [96] Introducing the Model Context Protocol \ Anthropic
https://www.anthropic.com/news/model-context-protocol

[3] [4] [5] [15] [19] [23] [31] What is MCP?. The Model Context Protocol (MCP) is a... | by Suteja Kanuri | May, 2025 | Medium
https://sutejakanuri.medium.com/what-is-mcp-d670dd7785c0

7 8 9 17 18 80 81 82 97 Unlocking the power of Model Context Protocol (MCP) on AWS | Artificial Intelligence and Machine Learning
https://aws.amazon.com/blogs/machine-learning/unlocking-the-power-of-model-context-protocol-mcp-on-aws/

13 Model Context Protocol (MCP) vs Application Programming Interface(API)
https://www.linkedin.com/pulse/model-context-protocol-mcp-balaji-thiagarajan-fxb5f

21 22 35 40 83 84 85 86 87 88 89 90 93 94 GitHub - modelcontextprotocol/servers: Model Context Protocol Servers
https://github.com/modelcontextprotocol/servers

28 32 33 38 39 41 42 43 44 45 46 47 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 91 92 95 99 Top 10 MCP Servers for 2025 (Yes, GitHub's Included!) - DEV Community
https://dev.to/fallon_jimmy/top-10-mcp-servers-for-2025-yes-githubs-included-15jg

34 48 49 50 51 52 53 54 55 56 GitHub - hanzoai/mcp: Hanzo AI + Hanzo Dev exposed via MCP.
https://github.com/hanzoai/mcp

36 37 GitHub - isaacphi/mcp-language-server: mcp-language-server gives MCP enabled clients access semantic tools like get definition, references, rename, and diagnostics.
https://github.com/isaacphi/mcp-language-server