# Understanding the Model Context Protocol (MCP) and the Role of Claude Desktop

## 1. Introduction

The rapid advancement of Large Language Models (LLMs) has highlighted a critical challenge: their isolation from the vast external data sources and tools necessary to perform complex, real-world tasks effectively.1 Integrating these models with diverse systems like databases, APIs, local filesystems, and business applications often requires bespoke, complex implementations for each connection, hindering scalability and interoperability.1 To address this, Anthropic introduced and open-sourced the Model Context Protocol (MCP) in late 2024.1

MCP is proposed as an open standard designed specifically to standardize how AI applications (such as chatbots, IDE assistants, or custom agents) connect with and exchange context with external tools, data sources, and systems.1 Its primary aim is to simplify the integration process, enabling AI models to access necessary information and capabilities reliably, ultimately leading to better, more relevant responses.1

This report provides a comprehensive analysis of the Model Context Protocol. It details the core definition and purpose of MCP, examines its client-server architecture and fundamental components, explains the communication mechanisms and interaction flows, and specifically investigates the role and implementation of Claude Desktop as an MCP Host application. Furthermore, it contextualizes MCP by comparing it with traditional integration methods and LLM function calling, and concludes with an overview of the current MCP ecosystem and its future outlook.

## 2. MCP: Definition and Purpose

The Model Context Protocol (MCP) is fundamentally an open protocol specification that standardizes the interface between AI applications and external systems providing data or executable functions.5 Often described using the analogy of a "USB-C port for AI applications" 2, MCP aims to provide a universal, plug-and-play connection method. Just as USB-C allows various peripherals to connect to a computer via a standard interface, MCP intends to allow diverse AI models and applications to connect to various tools and data sources through a single, consistent protocol, regardless of who developed either component.4 It is important to note that MCP itself is a protocol, akin to HTTP or SMTP, not a framework like LangChain or a specific tool.9

The core problem MCP addresses is the "M×N integration complexity".2 Before MCP, connecting *M* different AI applications to *N* different external tools or data sources potentially required *M* times *N* unique, custom integrations.3 This approach is difficult to scale and maintain.1 MCP transforms this into an "M+N problem".3 Application developers build *M* MCP clients (one for each application, capable of talking to any MCP server), and tool/data source providers build *N* MCP servers (one for each system, capable of talking to any MCP client).5 This standardization significantly reduces the integration effort required to build complex, interconnected AI systems.2

The primary goal is to simplify the development process by replacing fragmented, custom integrations with a single, reliable protocol, thereby enabling AI systems to securely access the data and tools they need to perform tasks effectively and provide contextually relevant outputs.1 It facilitates secure, two-way connections, allowing AI not only to retrieve information but also to trigger actions in external systems.1

## 3. Core MCP Architecture: Hosts, Clients, and Servers

MCP operates on a client-server architecture specifically designed for interactions between AI applications and external systems.2 This architecture comprises three primary components: Hosts, Clients, and Servers.

* **MCP Hosts:** These are the user-facing AI applications or environments where the interaction originates. Examples include Anthropic's Claude Desktop application, AI-enhanced Integrated Development Environments (IDEs) like Cursor or Zed, custom agents built with libraries like LangGraph, or platforms like Microsoft Copilot Studio.2 The Host coordinates the overall system, manages the LLM interaction, and decides which MCP servers to connect to based on the required capabilities.2
* **MCP Clients:** Clients act as intermediaries residing within the Host application.3 Each MCP Client is responsible for managing the connection and communication with *one specific* MCP Server, maintaining a dedicated one-to-one (1:1), stateful connection.2 The Host spawns and manages these client instances.2 Key responsibilities include connection management, capability discovery from the server, routing messages (requests, responses, notifications) between the Host and the Server according to the MCP specification, and ensuring security and isolation between different server connections.2
* **MCP Servers:** These are typically external programs that expose specific capabilities—access to data sources, APIs, or functions—to AI applications via the standardized MCP interface.2 Servers act as wrappers or bridges between the MCP world and the underlying system (e.g., a database, a Git repository, a local filesystem, a web service API like GitHub or Slack).2 They listen for requests from their connected Client, execute the requested operation (e.g., query data, call an API, run a script), and return the results according to the protocol.5 Servers can connect to local data sources (files, databases) or remote services.7

This architecture provides a clear separation of concerns: the Host focuses on user interaction and LLM orchestration, the Client handles protocol communication for a single connection, and the Server encapsulates the logic for interacting with a specific external system.2 This modularity facilitates scalability and maintainability in complex AI applications.2

**Table 1: MCP Components and Roles**

| **Component** | **Location** | **Role** | **Relationship Example** |
| --- | --- | --- | --- |
| **MCP Host** | User-facing AI Application | Coordinates overall system, manages LLM, initiates connections, manages clients, enforces policies 2 | Claude Desktop, Cursor IDE, Custom Agent 5 |
| **MCP Client** | Within the Host Application | Manages 1:1 stateful connection to a specific server, handles message routing, discovery, security/isolation 2 | An internal component within Claude Desktop connecting to the Filesystem MCP Server |
| **MCP Server** | External Program | Exposes capabilities (Tools, Resources, Prompts) of an underlying system via MCP, executes requests 2 | Filesystem Server, GitHub Server, Database Server 10 |

## 4. MCP Protocol Primitives

The MCP specification defines a set of core concepts, referred to as "primitives," that structure the communication and capabilities exchanged between Clients and Servers. These primitives are designed specifically for the needs of AI agents interacting with external systems.4

* **Tools (Model-controlled):** These represent executable functions or actions that an LLM can decide to invoke via the MCP Server to interact with the external world.4 Examples include sending an email, querying a database, creating a GitHub issue, searching the web, or manipulating files.5 Servers expose Tools along with descriptions and input schemas (often JSON Schema).5 The Host application typically parses these definitions and makes them available to the LLM (e.g., via function calling mechanisms).5 The LLM then determines *when* and *how* to use these tools based on the user's request and the provided descriptions.5
* **Resources (Application-controlled):** These represent data or content provided by the Server to the Host/LLM to enrich its context.4 Resources are typically read-only information, such as database records, file contents, API responses, code snippets, or log entries.4 The Host application, rather than the LLM directly, usually controls when to request or utilize these resources to provide relevant context for the LLM's task.5
* **Prompts (User-controlled / Server-defined):** These are pre-defined prompt templates or workflows exposed by the Server.4 They can guide the interaction between the AI and the external system for specific tasks, potentially simplifying complex operations or ensuring consistency.5 While often initiated by user actions, these prompts are defined by the server.4
* **Sampling (Server-initiated):** This is a more advanced primitive that allows an MCP Server to request an LLM completion *from* the Host/Client.3 This reverses the typical flow, enabling servers to leverage the LLM's reasoning or generation capabilities as part of their own processing logic for complex, multi-step tasks.3 The Host/Client retains full control over model selection, privacy, cost, and has the authority to approve or deny sampling requests, often requiring human-in-the-loop confirmation to prevent misuse.3
* **Roots (Client-defined):** Roots define specific locations (e.g., directories in a filesystem) within the Host's environment that a Server is authorized to interact with.3 They establish security boundaries, allowing the Client/Host to inform the Server about relevant resources and their locations while restricting access to authorized areas only.3 This is crucial for security when servers access local resources.

These primitives provide a structured vocabulary for AI agents to understand and interact with the capabilities offered by diverse external systems through the standardized MCP channel.

**Table 2: MCP Primitives**

| **Primitive** | **Description** | **Control / Initiation** | **Example Use Case** |
| --- | --- | --- | --- |
| **Tools** | Executable functions/actions exposed by the Server 4 | Model / LLM | LLM decides to call create\_github\_issue tool based on user request 5 |
| **Resources** | Data/content provided by the Server for context 5 | Application / Host | Host requests content of a specific file (Resource) to include in the LLM prompt 5 |
| **Prompts** | Pre-defined prompt templates/workflows exposed by the Server 4 | User / Server | User selects a "Summarize Document" prompt provided by a document server 5 |
| **Sampling** | Server requests LLM completion from the Host/Client 3 | Server | A complex workflow server asks the LLM to generate a sub-component description mid-process 3 |
| **Roots** | Host-defined access boundaries (e.g., file paths) for the Server 3 | Client / Host | Host tells Filesystem server it can only access /Users/username/Documents via a defined Root 3 |

## 5. MCP Communication and Interaction Flow

MCP defines specific mechanisms for communication between Clients and Servers, a lifecycle for their connection, and a clear interaction flow, particularly when invoking Tools.

### 5.1. Communication Mechanisms and Message Format

MCP supports multiple transport layers to accommodate different deployment scenarios 9:

* **stdio (Standard Input/Output):** Used primarily when the Client and Server run on the same machine.5 Communication happens via the standard input and output streams of the server process, making it simple and effective for local integrations like accessing local files or running local scripts.5 Claude Desktop utilizes this for local server support.17 Debugging requires care, as console.log output might interfere with the protocol messages on stdout.17
* **HTTP with Server-Sent Events (SSE):** Suitable for distributed architectures where the Client connects to a remote Server via HTTP.5 After an initial setup, the Server can push messages (events) asynchronously to the Client over a persistent connection using the SSE standard, while the Client sends commands via standard HTTP POST requests.5

Regardless of the transport mechanism, MCP uses **JSON-RPC 2.0** as its core messaging format.9 This provides a standardized, lightweight structure for requests, responses, and notifications.9

* **Requests:** Initiate an operation, containing jsonrpc, id, method, and optional params.9
* **Responses:** Reply to requests, containing jsonrpc, the matching id, and either a result or an error object.9
* **Notifications:** One-way messages not requiring a response, containing jsonrpc, method, and optional params.9

### 5.2. Connection Lifecycle

The connection between an MCP Client and Server follows a structured lifecycle 7:

1. **Initialization Phase:** When a connection is established, the Client and Server exchange initialize messages.9 They negotiate the protocol version and exchange information about their capabilities (e.g., supported features, server capabilities like available tools/resources).5 Compatibility is evaluated before proceeding.9
2. **Operation Phase:** Once initialized, normal protocol communication occurs.9 The Client sends requests (e.g., tools/list, tools/call, requests for resources) to the Server, and the Server sends responses or notifications back.5 This phase constitutes the main interaction period.
3. **Shutdown Phase:** When the connection is no longer needed (e.g., the user closes the Host application or disables the integration), a graceful termination process is initiated.9 Typically, the Client sends a shutdown request, the Server acknowledges it, and the Client may send a final exit notification before the connection is closed and resources are released.9

### 5.3. Interaction Flow: Using an MCP Tool

The following steps detail the typical flow when an LLM within a Host application decides to use an MCP Tool provided by a Server 5:

1. **Initialization & Discovery:** The Host (e.g., Claude Desktop) starts, creates MCP Clients for configured Servers, and performs the initialization handshake. The Client requests the list of capabilities (tools/list request) from the Server.5 The Server responds with descriptions of its available Tools, Resources, etc..5
2. **Context Provision & LLM Decision:** The Host application provides relevant context (user query, conversation history, potentially Resources) to the LLM. It also provides the LLM with the discovered Tool descriptions, often formatted for function calling.5 Based on the user's request (e.g., "Create a bug ticket for the login page crash") and the available tools, the LLM determines it needs to use a specific Tool (e.g., create\_ticket).5
3. **Invocation Request:** The Host application identifies the LLM's intent to use the tool. It directs the appropriate MCP Client to send a tools/call request to the corresponding Server.5 This JSON-RPC request includes the method ("tools/call"), a unique id, and params containing the tool name (create\_ticket) and necessary arguments extracted by the LLM (e.g., { "title": "Login page crash", "description": "..." }).9
4. **Server Execution:** The MCP Server receives the tools/call request.5 It validates the request, identifies the target tool (create\_ticket), and executes the underlying logic associated with that tool (e.g., making an API call to a service like Linear or Jira to create the ticket).5
5. **Server Response:** Once the action is completed (successfully or with an error), the Server formulates a JSON-RPC response.5 If successful, the response contains the original id and a result object (e.g., { "ticket\_id": "BUG-123", "url": "..." }).9 If an error occurred, it contains an error object.9 The Server sends this response back to the Client.5
6. **Completion & Final Response:** The Client receives the response and relays it to the Host application.5 The Host incorporates the result (e.g., the new ticket ID and URL) into the LLM's context.5 The LLM can then use this new information to generate a final, informed response for the user (e.g., "OK, I've created bug ticket BUG-123 for the login page crash. You can view it here:").5

This interaction flow highlights the decoupling achieved by MCP. The Host and LLM operate based on the standardized MCP interface (tool names, descriptions, parameters) without needing intimate knowledge of the specific external system's API or implementation details.5 The Server encapsulates that complexity. The LLM acts as the intelligent orchestrator, deciding *which* tool fits the user's need and extracting the necessary information, while MCP provides the robust, standardized plumbing to execute that decision.5

## 6. Claude Desktop: An MCP Host Implementation

Claude Desktop is a native application for macOS and Windows developed by Anthropic, providing users with access to the Claude family of AI models.23 Beyond being a chat interface, Claude Desktop functions as a key **MCP Host** application.1 This means it is designed to leverage the Model Context Protocol to allow the underlying Claude AI model to interact with external tools and data sources via MCP Servers.16 This capability significantly extends Claude's functionality beyond its core text generation and reasoning abilities, enabling it to perform actions like accessing local files, interacting with version control systems, or searching the web through configured servers.16 It's important to note that this MCP server integration is specifically a feature of the Claude Desktop application, not the web-based interface at claude.ai, primarily because it involves running and managing local server processes.20

### 6.1. Managing MCP Clients and Servers

As an MCP Host, Claude Desktop is responsible for managing the lifecycle of connections to MCP Servers. It achieves this primarily through a user-defined configuration file.16 When Claude Desktop starts, it reads this configuration file to identify which MCP Servers the user wants to make available.20 For each server defined in the configuration, Claude Desktop launches the specified server process locally, typically using the stdio transport mechanism for communication.5 Internally, Claude Desktop manages an MCP Client instance for each launched server, handling the 1:1 connection, message routing, and protocol adherence.5 This local execution model makes Claude Desktop a powerful tool for developers building and testing their own MCP servers.1

### 6.2. Configuring MCP Servers: claude\_desktop\_config.json

The integration of MCP servers into Claude Desktop is controlled via a JSON configuration file named claude\_desktop\_config.json.16

* **Location:**
  + macOS: ~/Library/Application Support/Claude/claude\_desktop\_config.json 16
  + Windows: %APPDATA%\Claude\claude\_desktop\_config.json 20 (Users may need to create this file manually if it doesn't exist 16).
* **Structure:** The file contains a single top-level JSON object. Within this object, a key named "mcpServers" holds another JSON object.16 This inner object contains key-value pairs where each key is a user-chosen name for the server (e.g., "filesystem", "github", "brave-search"), and the value is an object specifying how to run that server.16
* **Key Configuration Parameters** 20:
  + command (String): The executable command used to start the server process. Common examples include npx (for Node.js packages), python, docker, or a path to a specific executable.16
  + args (Array of Strings): An array containing the command-line arguments to pass to the command.16 This typically includes flags for the command (e.g., -y for npx), the server package or script name (e.g., @modelcontextprotocol/server-filesystem), and any arguments required by the server itself (e.g., allowed file paths for the filesystem server: "/Users/username/Desktop", "/Users/username/Downloads").20 Paths should be absolute.20
  + env (Object, Optional): A JSON object specifying environment variables to be set for the server process.16 This is commonly used to pass API keys or other sensitive configuration details securely to the server without hardcoding them in the args.16 Example: "env": { "GITHUB\_TOKEN": "YOUR\_TOKEN\_HERE" }.
* **Example Configuration Snippet:**  
  JSON  
  {  
   "mcpServers": {  
   "filesystem": {  
   "command": "npx",  
   "args": [  
   "-y",  
   "@modelcontextprotocol/server-filesystem",  
   "/Users/your\_username/Documents",  
   "/Users/your\_username/Projects"  
   ]  
   },  
   "github": {  
   "command": "npx",  
   "args": ["-y", "@modelcontextprotocol/server-github"],  
   "env": {  
   "GITHUB\_TOKEN": "YOUR\_GITHUB\_PAT"  
   }  
   },  
   "puppeteer-docker": {  
   "command": "docker",  
   "args":  
   }  
   }  
  }  
    
  *(Note: Users need prerequisites like Node.js/npm installed for npx commands 16 or Docker Desktop for docker commands.31 API keys/tokens must be obtained from the respective services.16)*

This configuration file serves as the central control panel for defining which external capabilities are made available to Claude through MCP within the Desktop application. Its declarative structure provides users with significant control over how servers are launched and the environment they operate in, including security aspects like file access permissions (via args) and secret management (via env).

### 6.3. User Experience and Interaction

After configuring the claude\_desktop\_config.json file, the user must restart Claude Desktop for the changes to take effect.20 Upon successful launch and connection to the configured servers, the application's UI typically indicates the availability of MCP tools, often via a specific icon (e.g., a hammer or plugs icon near the input box).20 Clicking this icon might reveal the list of specific tools provided by the connected servers (e.g., filesystem:readFile, github:create\_issue, brave\_web\_search).16

Users can then interact with these tools using natural language prompts directed at Claude. For instance, a user might ask:

* "Can you summarize the main points from the report located in my Documents folder?" (using the filesystem server) 20
* "Create a new branch named 'feature/mcp-integration' in my 'project-x' GitHub repository." (using the GitHub server) 16
* "Write a short Python script to list files in my Projects directory and save it to my Desktop." (using the filesystem server) 20

Claude, leveraging its understanding and the tool descriptions provided via MCP, will determine the appropriate tool to call, formulate the request, and interact with the MCP server via the Host application to fulfill the user's request. For potentially sensitive actions, Claude Desktop may present consent screens to the user before executing the tool call.31 Troubleshooting involves checking the configuration file syntax, ensuring paths are correct and absolute, restarting the application, and examining log files (mcp.log and mcp-server-SERVERNAME.log in the Claude log directory) for errors.20

Claude Desktop's role as an MCP Host serves as a practical demonstration and key enabler for the protocol, especially for developers. By providing a straightforward mechanism (claude\_desktop\_config.json) to integrate and test local MCP servers, Anthropic significantly lowers the barrier to entry for building and experimenting with MCP, thereby driving its adoption.1

### 6.4. Distinguishing MCP from Anthropic's "Computer Use" Feature

It is essential to differentiate the functionality provided by MCP within Claude Desktop from Anthropic's separate, newer feature known as "Computer Use".32

* **MCP** focuses on enabling Claude to interact with *specific, pre-defined external systems and capabilities* through dedicated **MCP Servers**.5 The interaction is structured around the MCP protocol's primitives (Tools, Resources, Prompts). Claude Desktop acts as an MCP Host, connecting to these servers based on the claude\_desktop\_config.json file.20
* **Computer Use** is a more general, agentic capability allowing Claude models (initially via API, potentially integrated into Desktop later) to *directly interact with the user's computer interface* – observing the screen, controlling the mouse cursor, clicking buttons, and typing text – much like a human user would.32 Its goal is to automate tasks across various applications *without necessarily requiring a dedicated MCP server* for each application or action, although it might internally leverage specific tools (like bash or text\_editor which *could* be exposed via MCP) or similar low-level interaction protocols.32

While both features aim to make Claude more capable by connecting it to external functionalities, they operate at different levels of abstraction. MCP provides standardized connections to defined services, whereas Computer Use aims for direct, human-like manipulation of the graphical user interface and underlying system commands.34 They represent distinct approaches within Anthropic's broader strategy to enhance AI agent capabilities.

## 7. MCP in Context: Comparison with Other Integration Methods

MCP enters an existing landscape of methods for integrating software systems and enabling AI models to interact with external functionalities. Understanding its relationship to these established approaches clarifies its specific value proposition.

### 7.1. MCP vs. Traditional APIs (OpenAPI, GraphQL, SOAP)

Protocols and standards like REST (often described using OpenAPI), GraphQL, and SOAP have long existed to facilitate communication between different software components, particularly over networks.5 While these are effective for traditional application-to-application communication, MCP was designed *specifically* with the unique requirements of modern AI agents and LLMs in mind.5

Key differences include:

* **Purpose-Built for AI:** MCP incorporates concepts like dynamic discovery, context management (Resources), and bidirectional communication (Sampling) tailored to how AI agents operate, whereas traditional APIs often follow a more rigid, stateless request-response pattern.3
* **Standardized Interaction Layer:** MCP aims to act as a unified protocol layer, allowing one MCP client integration to potentially access many different tools exposed via MCP servers.8 Traditional API integration typically requires building and maintaining separate connectors for each specific API endpoint or service.1
* **Dynamic Discovery:** MCP allows clients to dynamically query servers at runtime to discover available capabilities (Tools, Resources, Prompts).5 While OpenAPI allows for API definition discovery, MCP integrates this discovery into the operational flow for AI agents.
* **Stateful, Bidirectional Communication:** MCP connections are stateful, and the protocol supports bidirectional communication patterns (especially via SSE and the Sampling primitive), which is beneficial for ongoing agentic tasks, unlike the typically stateless nature of REST APIs.3

### 7.2. MCP vs. LLM Function Calling / Tool Use

Most major LLM providers (Anthropic, OpenAI, Google) offer a feature often referred to as "function calling" or "tool use".11 This capability allows developers to define external functions (tools) that the LLM can choose to invoke as part of its response generation process. The LLM, based on the prompt and tool descriptions, outputs a structured request (typically JSON) indicating which function to call and with what arguments.13

MCP is not a replacement for function calling; rather, it is **complementary** and acts as a **standardization layer** on top of or alongside it.13

* **Function Calling:** This is the LLM's ability to *decide* to call an external function and specify *what* function to call with *which* arguments, based on its understanding of the user's request and the provided tool descriptions.38 It's primarily about the LLM's reasoning and intent generation.
* **MCP:** This is the **protocol** that standardizes *how* that function call is discovered by the Host, *how* the request is routed to the correct external server, *how* the server executes the function, and *how* the results are returned to the Host/LLM.38 It defines the communication structure and the roles of the Client and Server in fulfilling the LLM's request.22

Key distinctions and synergies:

* **Standardization vs. Proprietary Implementation:** Function calling implementations are often specific to the LLM vendor (e.g., OpenAI's format differs from Google's).11 MCP provides an open, vendor-agnostic standard for the *execution* part of the process.2 An MCP server built for a specific tool (e.g., interacting with Salesforce) could theoretically be used by any LLM integrated with an MCP-compliant Host, regardless of the LLM vendor, promoting interoperability.11
* **Scope:** Basic function calling typically focuses only on invoking specific actions. MCP defines a richer interaction model including not just Tools, but also Resources (for context), Prompts (for guided workflows), Sampling (for server-initiated LLM calls), and Roots (for security).3
* **Workflow:** In an MCP-enabled system, the LLM uses its native function calling capability to generate the intent. The MCP Host then translates this intent into a standardized MCP tools/call request, sends it to the appropriate MCP Server via the MCP Client, receives the standardized response, and feeds the result back into the LLM's context.22

Therefore, MCP leverages the intelligence of LLM function calling but provides a standardized, robust, and interoperable framework for the actual execution and communication with external systems. This separation allows tool builders to focus on creating MCP servers for their services, and application builders to integrate various tools consistently across different LLMs and platforms.40

### 7.3. Key Differentiators Summarized

MCP distinguishes itself through:

* **Standardization:** Its core value is providing a common, open protocol to reduce integration complexity.3
* **Bidirectionality & Statefulness:** Supports persistent connections and advanced interactions like Sampling.3
* **Dynamic Discovery:** Enables runtime discovery of server capabilities.5
* **AI-Native Design:** Incorporates primitives (Tools, Resources, Prompts, Sampling, Roots) specifically tailored for LLM/agent requirements.3

**Table 3: MCP vs. Function Calling Comparison**

| **Feature** | **Model Context Protocol (MCP)** | **Function Calling (General LLM Feature)** |
| --- | --- | --- |
| **Scope** | Full protocol for discovery, invocation, context exchange (Tools, Resources, etc.) 5 | LLM capability to request execution of specific external functions/tools 38 |
| **Standardization** | Open, vendor-agnostic standard 2 | Typically proprietary, vendor-specific formats and implementations 11 |
| **Control/Initiation** | Host/Client initiates most requests; Server can initiate via Sampling 3 | LLM initiates the request to call a function 38 |
| **Communication Model** | Stateful, bidirectional (stdio/SSE), JSON-RPC based 3 | Part of the LLM API's request/response cycle; generates structured output (e.g., JSON) 38 |
| **Ecosystem** | Growing ecosystem of clients, servers, SDKs across vendors 1 | Feature implemented within specific LLM provider APIs (OpenAI, Anthropic, Google, etc.) 38 |
| **Relationship** | Standardizes the *execution* framework for function calls and context exchange 38 | Provides the *intent* for MCP (or other mechanisms) to execute 22 |

## 8. The MCP Ecosystem and Future Outlook

MCP's potential impact is closely tied to the growth and adoption of its surrounding ecosystem, which is being actively fostered through its open-source nature and community engagement.

### 8.1. Open Source Foundation: Specification, SDKs, and Servers

A cornerstone of MCP is its status as an open standard, with the specification and core implementations being open-sourced by Anthropic.1 This transparency allows developers to understand, implement, and contribute to the protocol.

* **Specification:** The official MCP specification is publicly available, detailing the architecture, primitives, message formats, and lifecycle.1
* **SDKs:** To accelerate development and adoption, official Software Development Kits (SDKs) are provided for multiple popular programming languages, including TypeScript, Python, C#, Java, Kotlin, Rust, and Swift.1 These SDKs, hosted in the modelcontextprotocol GitHub organization, simplify the process of building both MCP Clients and Servers.43 Several SDKs are maintained in collaboration with other organizations (e.g., Microsoft for C#, Spring AI for Java, loopwork-ai for Swift), indicating broader industry support.19 This multi-language support and collaborative development strategy significantly lowers the barrier to entry for diverse developer communities, aiming to build momentum rapidly.
* **Servers:** A growing repository of pre-built MCP Servers, maintained by Anthropic and the community, is available.1 These servers provide ready-made integrations for common tools and services like filesystems, Git/GitHub, databases (PostgreSQL, SQLite), search engines (Brave), communication platforms (Slack), cloud storage (Google Drive), browser automation (Playwright, Puppeteer), and more.10 The open-source nature encourages the community to contribute new servers, enriching the ecosystem far faster than a single entity could achieve.43

### 8.2. Current Adoption and Implementations

Since its launch, MCP has seen notable adoption and integration efforts:

* **Hosts/Clients:** Key implementations include Anthropic's own Claude Desktop, AI-focused IDEs like Cursor, Continue, and Zed, and agent frameworks.5
* **Partnerships and Integrations:** Anthropic has highlighted early adopters like Block and Apollo, and collaborations with development tool companies such as Zed, Replit, Codeium, and Sourcegraph.1 Microsoft has also embraced MCP, integrating it into products like Copilot Studio and Semantic Kernel, and co-maintaining the C# SDK.19
* **Cross-Company Signals:** Significantly, OpenAI has added support for MCP to its Agents SDK, suggesting that MCP may be achieving traction even among major competitors and potentially fostering convergence towards a common standard.10

### 8.3. Potential Evolution and Impact

MCP holds the potential to become a foundational layer for the next generation of AI agents and applications.9

* **Interoperability:** If widely adopted, MCP could enable seamless interoperability, allowing developers to build tools and agents that work across different LLMs and platforms with minimal friction, reducing vendor lock-in.2
* **Complexity Management:** By standardizing the complex task of tool integration, MCP can help manage the growing complexity of building sophisticated, multi-tool AI agents, particularly in enterprise environments with fragmented systems.21
* **Innovation:** A standardized protocol can foster innovation by allowing developers to build upon a common foundation, creating specialized servers and clients that can be easily combined.3
* **Future Development:** The protocol itself is likely to evolve based on community feedback and the emerging needs of increasingly capable AI agents.

However, its ultimate success depends on continued growth of the ecosystem, broad adoption by both tool providers (building servers) and application developers (building clients), and potentially navigating competition from other proprietary or emerging standards.3 The early momentum and strategic open-source approach, coupled with significant initial adoption, position MCP favorably, but the landscape remains dynamic.

## 9. Conclusion

The Model Context Protocol (MCP) represents a significant effort to address a fundamental challenge in the practical application of AI: the seamless and scalable integration of LLMs with the external world. By proposing an open, standardized protocol specifically designed for AI agent interactions, MCP aims to replace fragmented, custom integrations with a unified "USB-C port" approach.1 Its client-server architecture, encompassing Hosts, Clients, and Servers, provides a modular structure, while its core primitives (Tools, Resources, Prompts, Sampling, Roots) offer a rich vocabulary for context exchange and action execution.4

Claude Desktop serves as a prime example of an MCP Host implementation, demonstrating the practical application of the protocol.1 Its use of a configuration file (claude\_desktop\_config.json) to manage local MCP server connections makes it a valuable tool for developers experimenting with and building for the MCP ecosystem, effectively lowering the barrier to entry.16 It is crucial, however, to distinguish this structured, server-based interaction via MCP from Anthropic's more general "Computer Use" feature, which focuses on direct UI manipulation.32

Compared to traditional APIs, MCP offers AI-specific features like dynamic discovery and bidirectional communication.5 Crucially, MCP complements, rather than replaces, LLM function calling by providing the standardized execution framework for the intents generated by the LLM.38 Its primary advantage lies in its potential for interoperability and standardization across different models and platforms.2

Fueled by its open-source nature, growing SDK support across multiple languages, and an expanding repository of community-built servers, the MCP ecosystem shows promising early momentum.1 While its long-term dominance as *the* standard is not guaranteed in the rapidly evolving AI landscape, its thoughtful design, strong initial backing, and adoption signals from key players position MCP as a critical development towards building more capable, interconnected, and scalable AI systems.

#### Works cited

1. Introducing the Model Context Protocol - Anthropic, accessed April 8, 2025, <https://www.anthropic.com/news/model-context-protocol>
2. Anthropic's Model Context Protocol (MCP): A Deep Dive for Developers - Medium, accessed April 8, 2025, <https://medium.com/@amanatulla1606/anthropics-model-context-protocol-mcp-a-deep-dive-for-developers-1d3db39c9fdc>
3. An Introduction to Model Context Protocol - MCP 101 - DigitalOcean, accessed April 8, 2025, <https://www.digitalocean.com/community/tutorials/model-context-protocol>
4. The Model Context Protocol (MCP) by Anthropic: Origins, functionality, and impact - Wandb, accessed April 8, 2025, <https://wandb.ai/onlineinference/mcp/reports/The-Model-Context-Protocol-MCP-by-Anthropic-Origins-functionality-and-impact--VmlldzoxMTY5NDI4MQ>
5. Model Context Protocol (MCP) an overview - Philschmid, accessed April 8, 2025, <https://www.philschmid.de/mcp-introduction>
6. Model Context Protocol (MCP) - Anthropic API, accessed April 8, 2025, <https://docs.anthropic.com/en/docs/agents-and-tools/mcp>
7. Model Context Protocol: Introduction, accessed April 8, 2025, <https://modelcontextprotocol.io/introduction>
8. What is Model Context Protocol (MCP)? How it simplifies AI integrations compared to APIs | AI Agents That Work - Norah Sakal, accessed April 8, 2025, <https://norahsakal.com/blog/mcp-vs-api-model-context-protocol-explained/>
9. What is Model Context Protocol (MCP): Explained - Composio, accessed April 8, 2025, <https://composio.dev/blog/what-is-model-context-protocol-mcp-explained/>
10. What is MCP in AI? Model Context Protocol Simply Explained [No BS] - YouTube, accessed April 8, 2025, <https://www.youtube.com/watch?v=Xs9AwE2lyHg>
11. Model Context Protocol (MCP) and OpenAI's Stance | by Frank Goortani - Medium, accessed April 8, 2025, <https://medium.com/@FrankGoortani/model-context-protocol-mcp-and-openais-stance-496ead1da740>
12. Model Context Protocol (MCP), clearly explained (why it matters) - YouTube, accessed April 8, 2025, <https://www.youtube.com/watch?v=7j_NE6Pjv-E>
13. What Is the Model Context Protocol (MCP) and How It Works - Descope, accessed April 8, 2025, <https://www.descope.com/learn/post/mcp>
14. The Model Context Protocol (MCP) — A Complete Tutorial | by Dr. Nimrita Koul - Medium, accessed April 8, 2025, <https://medium.com/@nimritakoul01/the-model-context-protocol-mcp-a-complete-tutorial-a3abe8a7f4ef>
15. Claude MCP - Model Context Protocol, accessed April 8, 2025, <https://www.claudemcp.com/>
16. Implementing Model Context Protocol (MCP) with Claude Desktop: A Developer's Guide, accessed April 8, 2025, <https://guido-salimbeni.medium.com/implementing-model-context-protocol-mcp-with-claude-desktop-a-developers-guide-e36ab746617e>
17. Connecting Astra DB to Claude Desktop Using the Model Context Protocol (MCP), accessed April 8, 2025, <https://jherr2020.medium.com/connecting-astra-db-to-claude-desktop-using-the-model-context-protocol-mcp-e878e3902046>
18. Building Agents with Model Context Protocol - Full Workshop with Mahesh Murag of Anthropic - YouTube, accessed April 8, 2025, <https://www.youtube.com/watch?v=kQmXtrmQ5Zg>
19. Microsoft partners with Anthropic to create official C# SDK for Model Context Protocol, accessed April 8, 2025, <https://devblogs.microsoft.com/blog/microsoft-partners-with-anthropic-to-create-official-c-sdk-for-model-context-protocol>
20. For Claude Desktop Users - Model Context Protocol, accessed April 8, 2025, <https://modelcontextprotocol.io/quickstart/user>
21. Is Anthropic's Model Context Protocol Right for You? - WillowTree Apps, accessed April 8, 2025, <https://www.willowtreeapps.com/craft/is-anthropic-model-context-protocol-right-for-you>
22. Model Context Protocol (MCP): A comprehensive introduction for developers - Stytch, accessed April 8, 2025, <https://stytch.com/blog/model-context-protocol-introduction/>
23. claude.ai, accessed April 8, 2025, <https://claude.ai/#:~:text=Claude%20is%20a%20next%20generation,you%20do%20your%20best%20work.>
24. Claude, accessed April 8, 2025, <https://claude.ai/>
25. Meet Claude - Anthropic, accessed April 8, 2025, <https://www.anthropic.com/claude>
26. Anthropic Launches Claude AI Desktop Apps for Windows and macOS Users | by Cagri Sarigoz | BizStack — Entrepreneur's Business Stack, accessed April 8, 2025, <https://cagrisarigoz.medium.com/anthropic-launches-claude-ai-desktop-apps-for-windows-and-macos-users-a99d609e8d4b>
27. What is Claude Desktop and How-To Use it - YouTube, accessed April 8, 2025, <https://www.youtube.com/watch?v=lkt_g6RV1o0>
28. Installing Claude for Desktop | Anthropic Help Center, accessed April 8, 2025, <https://support.anthropic.com/en/articles/10065433-installing-claude-for-desktop>
29. Introducing the Model Context Protocol : r/ClaudeAI - Reddit, accessed April 8, 2025, <https://www.reddit.com/r/ClaudeAI/comments/1gzpf81/introducing_the_model_context_protocol/>
30. Model Context Protocol is everything I've wanted : r/ClaudeAI - Reddit, accessed April 8, 2025, <https://www.reddit.com/r/ClaudeAI/comments/1h1701e/model_context_protocol_is_everything_ive_wanted/>
31. The Model Context Protocol: Simplifying Building AI apps with Anthropic Claude Desktop and Docker, accessed April 8, 2025, <https://www.docker.com/blog/the-model-context-protocol-simplifying-building-ai-apps-with-anthropic-claude-desktop-and-docker/>
32. Anthropic Computer Use: Automate Your Desktop With Claude 3.5 - DataCamp, accessed April 8, 2025, <https://www.datacamp.com/blog/what-is-anthropic-computer-use>
33. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku - Anthropic, accessed April 8, 2025, <https://www.anthropic.com/news/3-5-models-and-computer-use>
34. Claude MCP vs OpenAI Operator : r/ClaudeAI - Reddit, accessed April 8, 2025, <https://www.reddit.com/r/ClaudeAI/comments/1iadlbt/claude_mcp_vs_openai_operator/>
35. Is Anthropic's Model Context Protocol (MCP) the Interoperability Standard We've Been Waiting For? | by Shuvro @ Nimesa | Medium, accessed April 8, 2025, <https://medium.com/@shuvro_25220/is-anthropics-model-context-protocol-mcp-the-interoperability-standard-we-ve-been-waiting-for-f2fe9e38110c>
36. Anthropic Now Lets Claude Take Control of Your Entire Computer - Futurism, accessed April 8, 2025, <https://futurism.com/the-byte/anthropic-claude-control-pc>
37. I gave Claude root access to my server... Model Context Protocol explained - YouTube, accessed April 8, 2025, <https://www.youtube.com/watch?v=HyzlYwjoXOQ>
38. Function Calling vs. Model Context Protocol (MCP): What You Need to Know, accessed April 8, 2025, <https://dev.to/fotiecodes/function-calling-vs-model-context-protocol-mcp-what-you-need-to-know-4nbo>
39. LLM Function-Calling vs. Model Context Protocol (MCP) - Gentoro, accessed April 8, 2025, <https://www.gentoro.com/blog/function-calling-vs-model-context-protocol-mcp>
40. What's MCP all about? Comparing MCP with LLM function calling - Neon, accessed April 8, 2025, <https://neon.tech/blog/mcp-vs-llm-function-calling>
41. Model Context Protocol vs Function Calling: What's the Big Difference? : r/ClaudeAI - Reddit, accessed April 8, 2025, <https://www.reddit.com/r/ClaudeAI/comments/1h0w1z6/model_context_protocol_vs_function_calling_whats/>
42. A Comprehensive Overview of the Model Context Protocol (MCP) | by Astropomeai, accessed April 8, 2025, <https://medium.com/@astropomeai/a-comprehensive-overview-of-the-model-context-protocol-mcp-f65150da0aa0>
43. Model Context Protocol - GitHub, accessed April 8, 2025, <https://github.com/modelcontextprotocol>
44. To MCP or Not to MCP Part 1: A Critical Analysis of Anthropic's Model Context Protocol - Sanjeev Mohan, accessed April 8, 2025, <https://sanjmo.medium.com/to-mcp-or-not-to-mcp-part-1-a-critical-analysis-of-anthropics-model-context-protocol-571a51cb9f05>