Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions

XINYI HOU, Huazhong University of Science and Technology, China
YANJIE ZHAO, Huazhong University of Science and Technology, China
SHENAO WANG, Huazhong University of Science and Technology, China
HAOYU WANG*, Huazhong University of Science and Technology, China

The Model Context Protocol (MCP) is an emerging open standard that defines a unified, bi-directional communication and dynamic discovery protocol between AI models and external tools or resources, aiming to enhance interoperability and reduce fragmentation across diverse systems. This paper presents a systematic study of MCP from both architectural and security perspectives. We first define the full lifecycle of an MCP server, comprising four phases (creation, deployment, operation, and maintenance), further decomposed into 16 key activities that capture its functional evolution. Building on this lifecycle analysis, we construct a comprehensive threat taxonomy that categorizes security and privacy risks across four major attacker types: malicious developers, external attackers, malicious users, and security flaws, encompassing 16 distinct threat scenarios. To validate these risks, we develop and analyze real-world case studies that demonstrate concrete attack surfaces and vulnerability manifestations within MCP implementations. Based on these findings, the paper proposes a set of fine-grained, actionable security safeguards tailored to each lifecycle phase and threat category, offering practical guidance for secure MCP adoption. We also analyze the current MCP landscape, covering industry adoption, integration patterns, and supporting tools, to identify its technological strengths as well as existing limitations that constrain broader deployment. Finally, we outline future research and development directions aimed at strengthening MCP's standardization, trust boundaries, and sustainable growth within the evolving ecosystem of tool-augmented AI systems. All collected data and implementation examples are publicly available at https://github.com/security-pride/MCP_Landscape.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Security and privacy → Software and application security**; • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: Model Context Protocol, MCP, Vision paper, Security

ACM Reference Format:
Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. 1, 1 (October 2025), 37 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, the vision of autonomous AI agents capable of interacting with a wide range of tools and data sources has gained significant momentum. This progress accelerated in 2023

*Haoyu Wang is the corresponding author (haoyuwang@hust.edu.cn).

Authors' addresses: Xinyi Hou, xinyihou@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Yanjie Zhao, yanjie_zhao@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Shenao Wang, shenaowang@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Haoyu Wang, haoyuwang@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China.

ACM XXXX-XXXX/2025/10-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

with the introduction of **function calling** by OpenAI, which allowed language models to invoke external APIs in a structured way [48]. This advancement expanded the capabilities of LLMs, enabling them to retrieve real-time data, perform computations, and interact with external systems. As function calling gained adoption, an ecosystem formed around it. OpenAI introduced the **ChatGPT plugin** [47], allowing developers to build callable tools for ChatGPT. LLM app stores such as Coze [8] and Yuanqi [63] have launched their **plugin stores**, supporting tools specifically designed for their platforms. Frameworks like LangChain [36] and LlamaIndex [39] provided standardized **tool interfaces**, making it easier to integrate LLMs with external services. Other AI providers, including Anthropic, Google, and Meta, introduced similar mechanisms, further driving adoption. Despite these advancements, **integrating tools remains fragmented**. Developers must manually define interfaces, manage authentication, and handle execution logic for each service. Function calling mechanisms vary across platforms, requiring redundant implementations. Moreover, existing agent frameworks already support a degree of autonomous tool selection, but they generally operate over **predefined or hardcoded integrations, limiting interoperability and long-term maintainability**.

In late 2024, Anthropic launched the **Model Context Protocol (MCP)** [4], a universal protocol for standardizing the definition, discovery, and invocation of external tools for AI applications. Drawing inspiration from the Language Server Protocol (LSP) [30], MCP introduces several innovations that extend beyond conventional function calling. First, it provides a **protocol-based standard** that decouples tool implementation from usage, enabling developers to publish and describe external functions dynamically, independent of any single model or agent framework. Second, MCP supports **dynamic discovery and schema negotiation**: the client can list available tools at runtime, retrieve their capabilities, and invoke them in a uniform manner, without requiring prior hardcoding or platform-specific adapters. Third, MCP enables **bi-directional communication channels**, allowing not only model-to-tool requests but also tool-initiated events and notifications back to the host. Finally, MCP designs access control and capability negotiation as first-class features, offering a foundation for more auditable and secure AI-to-tool interactions. These architectural differences position MCP as more than just an advanced function-calling mechanism. It shifts the paradigm from *tool bindings hardcoded per application* toward an *interoperable ecosystem of composable, discoverable network services*. Since its release, MCP has rapidly grown into a foundational architecture for AI-native applications: thousands of independently developed MCP servers expose model-accessible interfaces to services such as GitHub [52], Slack [53], and Blender [1], while platforms like Cursor [19] and Claude Desktop [3] demonstrate how MCP-enabled clients can flexibly extend functionality by connecting to new servers on demand. This approach transforms developer tools, productivity platforms, and creative environments into a truly interoperable and multimodal AI ecosystem.

Despite the rapid adoption of MCP, its ecosystem is still in the early stages, with key areas such as security, tool discoverability, and remote deployment lacking comprehensive solutions. These issues present untapped opportunities for further research and development. Although MCP is widely recognized for its potential in the industry, it has not yet been extensively analyzed in academic research. This gap motivates our work, which provides the first in-depth analysis of the MCP ecosystem by examining its architecture and workflow, formally defining the lifecycle of MCP servers across four phases and 16 key activities, and systematically analyzing security threats from multiple attacker perspectives. Our threat taxonomy identifies four major attacker types, including malicious developers, external attackers, malicious users, and security flaws, and covers 16 representative threat scenarios. These risks, such as tool poisoning, installer spoofing, and unauthorized access, are empirically validated through real-world case studies. Through this study, we present a thorough exploration of MCP's current landscape and offer a forward-looking vision that highlights key implications, outlines future research directions, and addresses the challenges that must be overcome to ensure its sustainable growth.

Our contributions are as follows:

- We provide the first analysis of the MCP ecosystem, detailing its architecture, components, and workflow.
- We identify the key components of MCP servers and define their lifecycle, encompassing the stages of creation, deployment, operation, and maintenance, across 16 key activities.
- We construct the systematic threat taxonomy for MCP, identifying four attacker archetypes (i.e., malicious developers, external attackers, malicious users, and security flaws) and 16 threat scenarios that together reveal the MCP's primary security exposure points.
- We examine the current MCP ecosystem landscape, analyzing the adoption, diversity, and use cases across various industries and platforms.
- We discuss the implications of MCP's rapid adoption, identifying key challenges for stakeholders, and outline future research directions on security, scalability, and governance to ensure its sustainable growth.

The remainder of this paper is structured as follows: § 2 compares tool invocation with and without MCP, highlighting the motivation for this study. § 3 outlines the architecture of MCP, detailing the roles of the MCP host, client, and server, as well as the lifecycle of the MCP server. § 4 examines the current MCP landscape, focusing on key industry players and adoption trends. § 5 analyzes security and privacy risks of the MCP server and proposes mitigation strategies. § 6 explores implications, future challenges, and recommendations to enhance MCP's scalability and security in dynamic AI environments. § 7 reviews prior work on tool integration and security in LLM applications. Finally, § 8 concludes the whole paper.

2 BACKGROUND AND MOTIVATION

2.1 AI Tooling

Before the introduction of MCP, AI applications relied on various methods, such as manual API wiring, plugin-based interfaces, and agent frameworks, to interact with external tools. As shown in Figure 1, these approaches required integrating each external service with a specific API, leading to increased complexity and limited scalability. **MCP addresses these challenges by providing a standardized protocol that enables seamless and flexible interaction with multiple tools.**

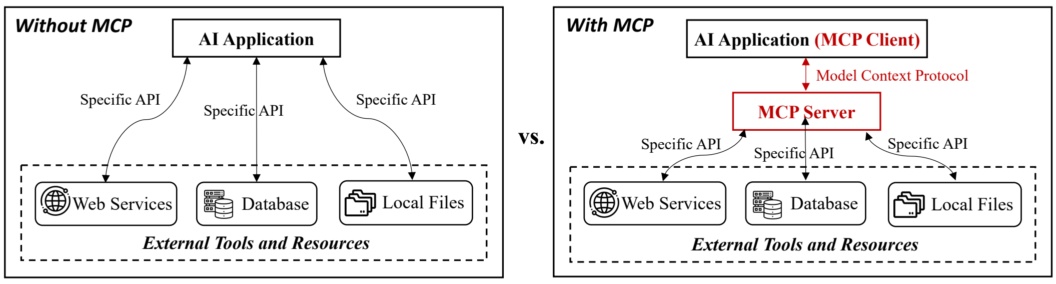

Fig. 1. Tool invocation with and without MCP. Without MCP, an AI application interacts with external tools and resources such as web services, databases, and local files through specific APIs. With MCP, the AI application functions as an MCP client that communicates with an MCP server using the MCP protocol, which provides a unified interface for tool access.

2.1.1 *Manual API Wiring.*

In traditional implementations, developers had to establish manual API connections for each tool or service that an AI application interacted with. This process **required custom authentication, data transformation, and error handling for every integration**. As the number of APIs increased, the maintenance burden became significant, often leading to tightly coupled and fragile systems that were difficult to scale or modify. MCP eliminates this complexity by offering a unified interface, allowing AI models to connect with multiple tools dynamically without the need for custom API wiring.

2.1.2 *Standardized Plugin Interfaces.*

To reduce the complexity of manual wiring, plugin-based interfaces such as OpenAI ChatGPT Plugins, introduced in November 2023 [47], allowed AI models to connect with external tools through standardized API schemas like OpenAPI. For example, in the OpenAI Plugin ecosystem, plugins like Zapier allowed models to perform predefined actions, such as sending emails or updating CRM records. However, these interactions were often **one-directional and could not maintain state or coordinate multiple steps in a task**. There are also new LLM app stores [80] that offer a web services plugin store, like ByteDance Coze [8] and Tencent Yuanqi [63]. Although these platforms increased the number of tools available, they also produced separate ecosystems where plugins are **platform-specific**, which restricts limiting cross-platform compatibility and requiring duplicate maintenance efforts. MCP stands out by being open-source and platform-agnostic, supporting **bi-directional communication channels** that allow not only model-to-tool invocations but also tool-initiated events and notifications, which are not possible in one-directional plugin designs.

2.1.3 *AI Agent Tool Integration.*

The emergence of AI agent frameworks like LangChain [36] and similar tool orchestration frameworks provided a structured way for models to invoke external tools through predefined interfaces, improving automation and adaptability [70]. However, integrating and maintaining these tools remained largely manual, requiring custom implementations and increasing complexity as the number of tools grew. MCP simplifies this process by **introducing a protocol-level abstraction that unifies how tools are described and discovered across platforms**. This protocol, in contrast to framework-specific integrations, makes it possible for tools created by various developers to work together, eliminating the need for duplicate maintenance and promoting a shared ecosystem.

2.1.4 *Retrieval-Augmented Generation (RAG) and Vector Database.*

Contextual information retrieval methods, such as RAG, leverage vector-based search to retrieve relevant knowledge from databases or knowledge bases, enabling models to supplement responses with up-to-date information [18, 24]. While this approach addressed the problem of knowledge cutoff and improved model accuracy, it was limited to **passive retrieval of information**. It did not inherently allow models to perform active operations, such as modifying data. For example, a RAG-based system could retrieve relevant sections from a product documentation database to assist a customer support AI. However, if the AI needed to update customer records or escalate an issue to human support, it can't take action beyond providing textual responses. MCP extends beyond passive information retrieval by **providing a standardized protocol for both retrieval and action**, allowing AI systems to combine knowledge access with secure tool execution under a unified framework.

2.2 Motivation

MCP has rapidly gained traction in the AI community due to its ability to standardize how AI models interact with external tools, fetch data, and execute operations. By addressing the limitations of manual API wiring, plugin interfaces, and agent frameworks, MCP redefines AI-to-tool interactions by enabling **interoperable, secure, and maintainable workflows across heterogeneous**

environments. Unlike prior approaches, MCP incorporates access control, capability negotiation, and schema discovery as protocol primitives, which distinguishes it from conventional function-calling mechanisms. Despite its growing adoption and promising potential, MCP is still in its early stages, with an evolving ecosystem that remains incomplete. Many key aspects, such as security and tool discoverability, are yet to be fully addressed, leaving ample room for future research and improvement. Moreover, while MCP has gained rapid adoption in the industry, it is still largely unexplored in academia.

Motivated by this gap, this paper is **the first to analyze the current MCP landscape, examine its emerging ecosystem, and identify potential security risks**. Additionally, we outline a vision for MCP's future development and highlight the key challenges that must be addressed to support its long-term success.

3 MCP ARCHITECTURE

3.1 Core Components

The MCP architecture is composed of three core components: ***MCP host***, ***MCP client***, and ***MCP server***. These components collaborate to facilitate seamless communication between AI applications, external tools, and data sources, ensuring that operations are secure and properly managed. As shown in Figure 2, in a typical workflow, the user sends a prompt to the MCP client, which **analyzes the intent**, **selects the appropriate tools** via the MCP server, and **invokes external APIs** to retrieve and process the required information before **notifying** the user of the results.

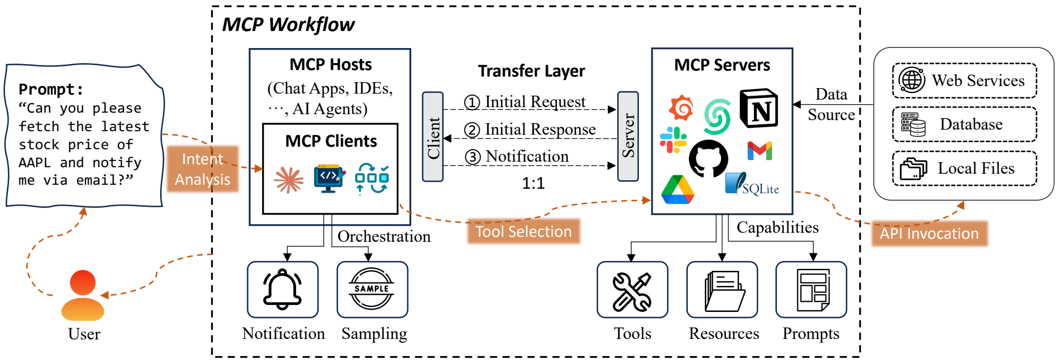

Fig. 2. The workflow of MCP. A user prompt is processed through a series of stages involving intent analysis, tool selection, and API invocation across the MCP host, client, and server. The MCP server provides tools, resources, and prompts that enable interaction with external data sources such as web services, databases, and local files. The notation "1:1" in the transfer layer indicates a one-to-one communication link between each MCP client and MCP server during request and response exchange.

3.1.1 MCP Host. The MCP host is an AI application that provides the environment for executing AI-based tasks while running the MCP client. It integrates interactive tools and data to enable smooth communication with external services. Examples include Claude Desktop for AI-assisted content creation, Cursor, an AI-powered IDE for code completion and software development, and AI agents that function as autonomous systems for executing complex tasks. The MCP host hosts the MCP client and ensures communication with external MCP servers.

3.1.2 MCP Client. In the MCP **host–client–server** architecture, the MCP client acts as an intermediary component within the host environment, maintaining a one-to-one communication link with its corresponding MCP server. Operating on behalf of the host, the client initiates requests to the MCP server, queries available functions, and retrieves responses that describe the server's tools, resources, and capabilities. This design enables the host to seamlessly access and utilize external functionalities provided by MCP servers. In addition to managing requests and responses, the MCP client processes **notifications** from MCP servers, providing real-time updates about task progress and system status. It also performs **sampling** to gather data on tool usage and performance, enabling optimization and informed decision-making. The MCP client communicates through the transport layer with MCP servers, facilitating secure, reliable data exchange and smooth interaction between the host and external resources.

3.1.3 MCP Server. The MCP server enables the MCP host and client to access external systems and execute operations, offering three core capabilities: ***tools, resources, and prompts***.

- **Tools: Enabling external operations**. Tools in MCP enable the server to invoke external services and APIs to execute operations on behalf of AI models. When a host application (such as an AI assistant) needs to perform an operation, it first queries the MCP server through the client to obtain the list of available tools and their capabilities. Based on the task context, the host then selects an appropriate tool and issues an invocation request via the client. The MCP server executes the corresponding operation through the external service or API and returns the result to the client, which forwards it to the host. For example, if an AI model requires real-time weather data or sentiment analysis, the host identifies the corresponding tool from the server's advertised capabilities, the server performs the API call, and the resulting data is delivered back to the host. Unlike traditional function-calling interfaces that are confined within a single model or framework, tools in MCP are described and accessed through a **standardized, model-agnostic protocol**. This design allows tools to be dynamically discovered, described, and invoked across heterogeneous AI systems, ensuring cross-platform interoperability rather than provider-specific integration. Furthermore, MCP tools support **bi-directional communication** between the host and the service, enabling richer interactions such as asynchronous updates or event streaming. Once configured, these tools adhere to a **supply-and-consume model** that promotes modularity and reusability, allowing independently developed tools to interoperate seamlessly and improving system efficiency and extensibility.
- **Resources: Exposing data to AI models**. Resources provide access to structured and unstructured datasets that the MCP server can expose to AI models. These datasets may come from local storage, databases, or cloud platforms. When an AI model requests specific data, the MCP server retrieves and processes the relevant information, enabling the model to make data-driven decisions. For example, a recommendation system may access customer interaction logs, or a document summarization task may query a text repository.
- **Prompts: Reusable templates for workflow optimization**. Prompts are predefined templates and workflows that the MCP server generates and maintains to optimize AI responses and streamline repetitive tasks. They ensure consistency in responses and improve task execution efficiency. For instance, a customer support chatbot may use prompt templates to provide uniform and accurate responses, while an annotation task may rely on predefined prompts to maintain consistency in data labeling.

3.2 Transport Layer and Communication

The transport layer ensures secure, bidirectional communication, allowing for real-time interaction and efficient data exchange between the host environment and external systems. The transport layer manages the transmission of initial requests from the client, the delivery of server responses detailing available capabilities, and the exchange of notifications that keep the client informed of ongoing updates. Communication between the MCP client and the MCP server follows a structured process, beginning with an **initial request** from the client to query the server's functionalities. Upon receiving the request, the server responds with an **initial response** listing the available tools, resources, and prompts the client can leverage. Once the connection is established, the system maintains a continuous exchange of **notifications** to ensure that changes in server status or updates are communicated back to the client in real time. This structured communication ensures high-performance interactions and keeps AI models synchronized with external resources, enhancing the effectiveness of AI applications.

3.3 MCP Server Lifecycle

Figure 3 summarizes the complete lifecycle of an MCP server from the server's perspective. This lifecycle is primarily derived from the official protocol documentation and a systematic analysis of actual MCP operational workflows. Based on the transitions of primary participant roles, the lifecycle is divided into four sequential phases: *creation*, *deployment*, *operation*, and *maintenance*. In the *creation* phase, the main actor is the developer, who defines metadata, declares capabilities, and implements the MCP server. The *deployment* phase covers the process in which the developer releases the server to a public platform, and users deploy it to an MCP host while the client establishes its connection. The *operation* phase corresponds to the runtime period when users actively interact with the server through the MCP system. Finally, the *maintenance* phase involves version iteration, configuration updates, and continuous optimization of the deployed MCP server. The following subsection introduces the key activities of each phase in detail.

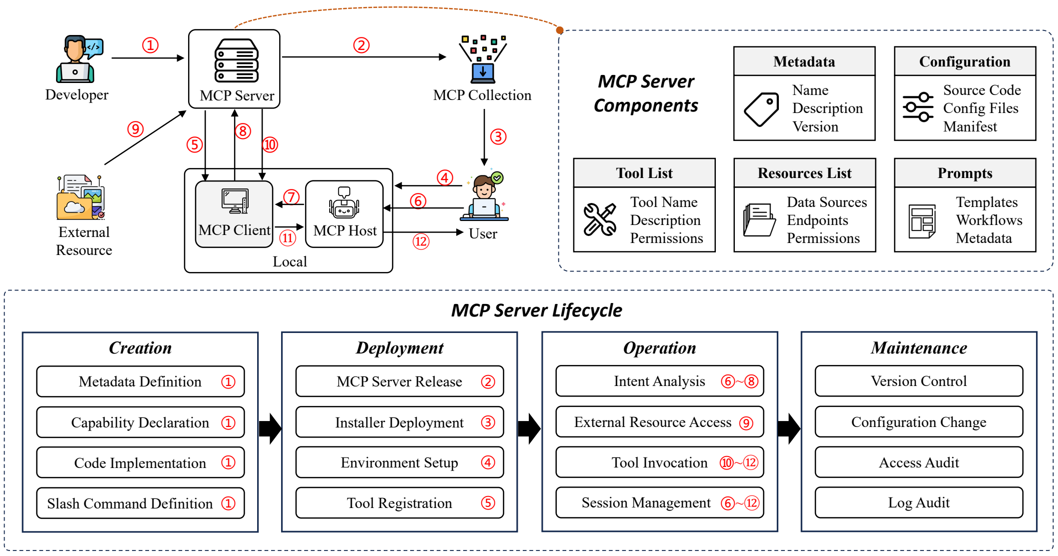

Fig. 3. MCP server components and lifecycle. The upper part presents the time-ordered interaction flow among the developer, MCP server, host, client, user, and external resources. The lower part summarizes the main lifecycle phases, including creation, deployment, operation, and maintenance. The numbered circles (e.g., ①②③) indicate sequential actions in the upper process, which correspond to key phases in the lifecycle.

3.3.1 *MCP Server Components.* The MCP server is responsible for managing external tools, data sources, and workflows, providing AI models with the necessary resources to perform tasks efficiently and securely. It comprises several key components that ensure smooth and effective operations. **Metadata** includes essential information about the server, such as its name, version, and description, allowing clients to identify and interact with the appropriate server. **Configuration** involves the source code, configuration files, and manifest, which define the server's operational parameters, environment settings, and security policies. **Tool list** stores a catalog of available tools, detailing their functionalities, input-output formats, and access permissions, ensuring proper tool management and security. **Resources list** governs access to external data sources, including web APIs, databases, and local files, specifying allowed endpoints and their associated permissions. Finally, **Prompt templates** include pre-configured task templates and workflows that enhance the efficiency of AI models in executing complex operations. These components enable MCP servers to provide seamless tool integration, data retrieval, and task orchestration for AI-powered applications.

3.3.2 Creation Phase. The creation phase is the starting point of the MCP server lifecycle. During this stage, developers establish the server's essential data structures and capabilities, transforming conceptual requirements into a fully defined and executable MCP component. The result is a standardized service that can be reliably integrated into subsequent deployment and operation processes. **Metadata definition** focuses on describing the server's identity through its name, version, description, and supported protocol version. These data form the foundation for interoperability and version control. **Capability declaration** specifies the standardized functions the server provides, such as tools, resources, or prompts, together with their operational boundaries and permission requirements. This step determines how the server will respond to standardized requests. Vague or inaccurate declarations may lead to capability misuse or potential security violations. Many security vulnerabilities identified in MCP deployments originate from flaws introduced at this stage. Therefore, scanning and validation of capability declarations constitute an essential step in strengthening MCP server security. **Code implementation** connects the declared capabilities with concrete request handlers that process input data, execute corresponding logic, and generate structured results. The quality and security of this implementation directly influence the stability and safety of all exposed functions. Poorly designed or insecure code can introduce vulnerabilities that compromise both reliability and user trust. **Slash command definition** establishes optional user-interaction commands that correspond to specific prompts. Clear and coherent command definitions enhance usability and improve the flexibility and intuitiveness of prompt invocation during subsequent operations.

3.3.3 Deployment Phase. In this stage, the server that has been designed and implemented is prepared, packaged, and released into an operational environment where clients and hosts can interact with it through standardized interfaces. **MCP server release** involves packaging the finalized server codebase, configuration files, and metadata into a distributable form. This step may include version tagging, dependency documentation, and integrity verification to guarantee that the deployed artifact remains consistent with the validated build. Developers can subsequently publish their packaged servers to various MCP server markets, as illustrated in Table 2, allowing end users to discover and install servers directly from these repositories. Most existing markets are maintained by third-party platforms, but Anthropic has also begun developing an official MCP registry aimed at providing verified listings, enhanced security trust, and unified distribution management for the MCP ecosystem. **Installer deployment** handles the distribution and installation of the server package within target systems. Installers can use container images, package managers, or automated scripts to streamline deployment. Clear installation workflows reduce setup errors and make the integration process predictable. Security validation of installers, such as checksum verification and signature authentication, is necessary to prevent tampering or injection of malicious binaries. **Environment setup** ensures that the deployed MCP server operates under the correct runtime configuration. This includes defining environment variables, access credentials, logging policies, and network permissions required for communication with clients and hosts. Configuration isolation and principle-of-least-privilege practices help mitigate risks of unauthorized access or data leakage during runtime. **Tool registration** finalizes deployment by registering the server's capabilities with the hosting application or orchestration system. During this process, tools, resources, and prompts declared during creation are discovered and made available to connected clients.

3.3.4 Operation Phase. The operation phase represents the runtime stage of the MCP server lifecycle, where the deployed server actively interacts with users, clients, and external resources. During this phase, the server interprets user intent, mediates resource access, invokes registered tools, and manages ongoing sessions to deliver reliable and contextual responses. **Intent analysis** is the initial operation stage in which user inputs are parsed, contextualized, and mapped to the most suitable MCP capabilities. The host or client forwards user requests to the MCP server, where intent interpretation logic determines whether to trigger a tool, retrieve a resource, or engage a prompt. Accurate intent analysis directly affects usability and effectiveness. Misinterpretation can lead to incorrect tool execution or unnecessary resource calls, potentially increasing latency or amplifying risk exposure. **External resource access** occurs when the server needs to obtain supplementary data from third-party systems or knowledge repositories. Such access is handled via predefined resource interfaces and is strictly governed by security permissions defined during creation. Each access request must comply with authentication, authorization, and sandboxing policies to prevent data leakage or external dependency compromise. **Tool invocation** is the execution stage where the MCP server or client triggers a registered tool according to the interpreted intent. Each tool invocation includes structured parameter passing, execution monitoring, and result serialization before returning structured outputs to the user session. Reliability and isolation at this stage determine runtime stability. **Session management** maintains the logical continuity between user interactions and server processes. It includes establishing, monitoring, and closing session contexts that connect the MCP host, client, and user interface.

3.3.5 Maintenance Phase. After deployment and operation, an MCP server must be periodically maintained to ensure its reliability, security, and compliance with evolving system requirements. **Version control** ensures that all updates to the MCP server, including bug fixes and capability extensions, are tracked through an auditable revision system. Proper version management enables rollback to stable releases, facilitates controlled feature rollout, and supports compatibility testing across client environments. **Configuration change** management governs modifications to runtime parameters, environmental variables, or authorization policies. All configuration adjustments should follow a controlled workflow, such as change-request approval and pre-deployment validation, to prevent inadvertent disruptions. **Access audit** records and reviews all authentication and authorization events related to the MCP server. This includes tracking user sessions, privilege escalations, and external resource permissions utilized during operation. Regular access auditing allows administrators to identify abnormal patterns, enforce least-privilege policies, and comply with organizational or regulatory security frameworks. **Log audit** focuses on continuous collection and analysis of operational logs generated by the server, tools, and clients. Centralized log aggregation and integrity-protected storage support forensic traceability and incident response. Automated log analysis, through anomaly detection or correlation with known attack signatures, can reveal early indicators of compromise or performance degradation. Effective log auditing transforms runtime data into actionable intelligence for system governance.

4 CURRENT LANDSCAPE

4.1 Ecosystem Overview

4.1.1 Key Adopters. Table 1 demonstrates how MCP has gained significant traction across diverse sectors, signaling its growing importance in enabling seamless AI-to-tool interactions. The dataset summarized in this table was compiled through manual inspection, drawing on official documentation from the earliest MCP supporters and extended via community discussions and repository mining. We primarily included mature products and companies with verifiable MCP integration. Although this collection provides a representative snapshot of the current MCP landscape, we acknowledge that it is not exhaustive. To enhance transparency and facilitate future updates, the dataset will be maintained as a public repository[1], enabling ongoing community contributions and periodic verification.

Table 1. Overview of MCP ecosystem adoption (updated to Sept. 2025).

Category	Company/Product	Key Features or Use Cases
AI Models and Frameworks	Anthropic (Claude) [3]	Full MCP support in the desktop version, enabling interaction with external tools.
	OpenAI [50]	MCP support integrated across products and within the Agent SDK for seamless interoperability.
	Google DeepMind [20]	Added MCP protocol support for the Gemini model family, enabling standardized tool invocation.
	Baidu Maps [41]	API integration using MCP to access geolocation and spatial services.
	Blender MCP [43]	Enables Blender and Unity 3D model generation via natural language commands.
Developer Tools	Replit [55]	AI-assisted development environment with integrated MCP tools.
	Microsoft Copilot Studio [62]	Officially announced MCP support in March 2025, enabling unified tool integration.
	Sourcegraph Cody [17]	Implements MCP through OpenCTX for resource integration.
	Codeium [16]	Adds MCP support for coding assistants to facilitate cross-system tasks.
	Cursor [19]	MCP tool integration in Cursor Composer for seamless code execution.
	Cline [11]	VS Code coding agent that manages MCP tools and servers.
IDEs/Editors	Zed [77]	Provides slash commands and tool integration based on MCP.
	JetBrains [33]	Integrates MCP for IDE-based AI tooling.
	Windsurf Editor [22]	AI-assisted IDE with MCP tool interaction.
	TheiaAI/TheiaIDE [65]	Enables MCP server interaction for AI-powered tools.
	Emacs MCP [42]	Enhances AI functionality in Emacs by supporting MCP tool invocation.
	OpenSumi [51]	Supports MCP tools in IDEs and enables seamless AI tool integration.
Cloud Platforms and Services	Cloudflare [15]	Provides remote MCP server hosting, OAuth integration, and scalable multi-tenant infrastructure.
	Tencent Cloud [14]	Added MCP plugin and transport support, enabling AI SDK-level integration.
	Alibaba Cloud Bailian [12]	Introduced full lifecycle MCP service for large-scale batch integration.
	Huawei Cloud [13]	Launched AI-native run platform with built-in MCP module to accelerate trusted AI deployment.
	Block (Square) [60]	Uses MCP to enhance data processing efficiency for financial platforms.
	Stripe [61]	Exposes payment APIs via MCP for seamless AI integration.
	Alipay / Ant Group [2, 29]	Released "Alipay MCP Server" and "MCP Zone", offering unified payment and tool orchestration.
Web Automation and Data	Apify MCP Tester [64]	Connects to any MCP server using SSE for API testing.
	LibreChat [38]	Extends the current tool ecosystem through MCP integration.
	Baidu Create Conference [5]	Established a dedicated MCP ecosystem forum to promote developer collaboration.

Notably, leading AI companies such as Anthropic [3] and OpenAI [50] have integrated MCP to enhance agent capabilities and improve multi-step task execution. This adoption by industry pioneers has set a precedent, encouraging other major players to follow suit. Chinese tech giants like Baidu [41] have also incorporated MCP into their ecosystems, highlighting the protocol's potential to standardize AI workflows across global markets. Developer tools and IDEs, including Replit [55], Microsoft Copilot Studio [62], JetBrains [33], and TheiaIDE [65], leverage MCP to facilitate agentic workflows and streamline cross-platform operations. This trend indicates a shift toward embedding MCP in developer environments to enhance productivity and reduce manual integration efforts. Furthermore, cloud platforms like Cloudflare [15] and financial service providers such as Block (Square) [60] and Stripe [61] are exploring MCP to improve security, scalability, and governance in multi-tenant environments. The widespread adoption of MCP by these industry leaders not only highlights its growing relevance but also points to its potential as a foundational layer in AI-powered ecosystems. As more companies integrate MCP into their operations, the

[1]https://github.com/security-pride/MCP_Landscape

protocol is set to play a central role in shaping the future of AI tool integration. Looking ahead, MCP is poised to become a key enabler of AI-driven workflows, driving more secure, scalable, and efficient AI ecosystems across industries.

While these examples show clear momentum, the overall MCP ecosystem remains uneven in maturity. Many community-hosted servers are small experiments or early prototypes. Only a limited number demonstrate strong reliability, stable maintenance, or standardized documentation. A deeper issue lies in platform governance. Many large internet platforms operate as walled gardens. They treat data as a controlled resource and limit access to interfaces or user behavior data to maintain a competitive advantage. In such systems, the open and composable design that MCP promotes may face natural resistance. We note that platforms might only expose low-risk or secondary functions through MCP connections. For example, an online map service might allow MCP access to public location data or route information, but not to personalized navigation history or user-generated content. Similarly, a social media platform could open an MCP interface for basic profile lookup or public post retrieval, while keeping private messages and recommendation algorithms restricted. These partial integrations show that platform openness often depends on strategic and privacy considerations rather than technical feasibility. This reflects a tension between open interoperability and proprietary data control. The success of MCP will partly depend on whether large platforms are willing to share meaningful data and support cross-system collaboration.

4.1.2 Community-Driven MCP Servers. Although Anthropic has not yet released an official MCP marketplace, the broader community has actively filled this gap by creating numerous independent MCP server collections and discovery platforms. As summarized in Table 2, we identified and catalogued all publicly available MCP server directories accessible as of September 2025. These include a diverse range of deployment mode, including websites, GitHub repositories, and desktop applications, demonstrating how third-party developers are collectively working to establish a complete and sustainable MCP ecosystem. The data presented in Table 2 were collected and verified manually. We began by surveying known MCP-related repositories on GitHub and popular community forums, followed by direct examination of listed directory or application to confirm its activity, accessibility, and declared number of hosted servers. Most platforms explicitly report total server counts, which we cross-checked against their public listings; where such data were missing, we manually counts from available catalog pages. This collection process yielded a consolidated dataset encompassing 26 major MCP collections. Platforms such as MCP.so [45], Glama [28], and PulseMCP [23] host thousands of servers, allowing users to discover and integrate a wide range of tools and services. These community-driven platforms have significantly accelerated the adoption of MCP by providing accessible repositories where developers can publish, manage, and share their MCP servers. Desktop-based solutions like Dockmaster [44] and Toolbase [27] further enhance local MCP deployment capabilities, empowering developers to manage and experiment with servers in isolated environments.

Despite this rapid growth, the quality of community-maintained MCP markets remains uneven. Platforms like MCP.so host thousands of entries but lack formal security or identity verification mechanisms. To assess reliability, we randomly sampled 300 servers listed on MCP.so. Among them, 30 contained the term “MCP” in the project title but did not refer to the Model Context Protocol, and 18 were in active development or unavailable at the time of inspection. These findings suggest that large community directories may overstate their effective numbers, and server quality varies widely across listings. In contrast, platforms such as mcp-get implement verification and signing processes, but the number of verified servers remains very small and user adoption is limited. The overall landscape therefore reflects a trade-off between openness and quality control:

while community initiatives accelerate ecosystem growth, they also highlight the need for stronger validation standards and sustainable curation practices.

Table 2. Overview of MCP server collections and deployment modes (As of Sept. 2025).

Collection	Author	Mode	# Servers	URL
MCPWorld	Baidu	Website	26,404	mcpworld.com
MCP.so	mcpso	Website	16,592	mcp.so
MCP Servers Repository	mcprepository	Website	13,596	mcprepository.com
AIbase MCP	AIbase	Website	12,448	mcp.aibase.com
Glama	glama.ai	Website	9,415	glama.ai
Smithery	Henry Mao	Website	6,888	smithery.ai
PulseMCP	Antanavicius et al.	Website	6,072	pulsemcp.com
ModelScope MCP Marketplace	modelscope	Website	5,441	modelscope.cn/mcp
Awesome MCP Servers	wong2	Website	2,402	mcpservers.org
Cursor Directory MCP	Cursor	Website	1,800	cursor.directory/mcp
Official Collection[1]	Anthropic	GitHub Repo	1,204	modelcontextprotocol/servers
AiMCP	Hekmon	Website	907	aimcp.info
Dockmaster	mcp-dockmaster	Desktop App	516	mcp-dockmaster.com
MCP Market	mcpmarket.com	Website	463	mcpmarket.com
MCP.run	mcp.run	Website	242	mcp.run
Awesome MCP Servers	Stephen Akinyemi	GitHub Repo	217	appcypher/mcp-servers
CLine MCP Marketplace	Cline	Website	154	cline.bot/mcp-marketplace
Bailian MCP Market	Aliyun	Website	151	bailian.console.aliyun.com
OpenTools	opentoolsteam	Website	148	opentools.com
Awesome Remote MCP Servers	JAW9C	GitHub Repo	79	jaw9c/awesome-remote-mcp-servers
MCP Server Hub	mcpserverhub	Website	71	mcpserverhub.com
mcp-get	Michael Latman	Website	59	mcp-get.com
MCP Marketplace	Higress.ai	Website	50	mcp.higress.ai
Toolbase	gching	Desktop App	24	gettoolbase.ai
make inference	mkinf	Website	23	mkinf.io
Awesome Crypto MCP Servers	Luke Fan	GitHub Repo	12	badkk/crypto-mcp-servers

[1] **Official Collection** refers to the list of MCP servers curated by Anthropic. Anthropic has also launched the community-driven MCP Registry, a preview service that provides a centralized directory for discovering MCP servers (https://github.com/modelcontextprotocol/registry).

4.1.3 SDKs and Tools. With the continuous growth of community-driven tools and official SDKs, the MCP ecosystem is becoming increasingly accessible, allowing developers to integrate MCP into various applications and workflows efficiently. Official SDKs are available in multiple languages, including *TypeScript*, *Python*, *Java*, *Kotlin*, and *C#*, providing developers with versatile options to implement MCP in different environments. In addition to official SDKs, the community has contributed numerous frameworks and utilities that simplify MCP server development. Tools such as *EasyMCP* and *FastMCP* offer lightweight TypeScript-based solutions for quickly building MCP servers, while *FastAPI to MCP Auto Generator* enables the seamless exposure of FastAPI endpoints as MCP tools. For more complex scenarios, *Foxy Contexts* provides a Golang-based library to build MCP servers, and *Higress MCP Server Hosting* extends the API Gateway (based on Envoy) to host MCP servers with wasm plugins. Server generation and management platforms such as *Mintlify*, *Speakeasy*, and *Stainless* further enhance the ecosystem by **automating MCP server generation**, providing curated MCP server lists, and enabling faster deployment with minimal manual intervention. These platforms empower organizations to rapidly create and manage secure and well-documented MCP servers.

4.2 Use Cases

MCP has become a vital tool for AI applications to effectively communicate with external tools, APIs, and systems. By standardizing interactions, MCP simplifies complex workflows, boosting the efficiency of AI-driven applications. Below, we explore three key platforms (i.e., OpenAI, Cursor, and Cloudflare) that have successfully integrated MCP, highlighting their distinct use cases.

4.2.1 OpenAI: MCP Integration in AI Agents and SDKs. We chose OpenAI as the case study because it holds significant influence in the field of agent development. The Agents SDK [49], as a mainstream development framework, covers a wide range of real-world applications. It incorporates basic capabilities such as managing conversations, invoking tools, task delegation, input and output verification, and session recording. Developers only need to write a small amount of Python code to enable the intelligent agent to handle multiple rounds of conversations automatically, flexibly call various tools, and even allow multiple intelligent agents to collaborate to complete complex tasks. After the MCP was added, the Agents SDK became more efficient in terms of tool integration. In the past, developers had to customize the connection logic separately for each new tool or external service. This was not only time-consuming but also prone to errors. Now, with the help of MCP, developers only need to configure the address of the MCP tool, and the agent can automatically discover, connect to, and call it, regardless of where these tools are deployed. The entire integration process has become more unified, and maintaining and extending new tools has become much simpler. ChatGPT currently supports MCP in developer mode. Users can directly connect MCP tools within ChatGPT, enabling them to do more than just query data. They can also write to external systems, trigger automated tasks, or link multiple tools together to perform complex operations. The introduction of MCP has transformed ChatGPT from a past question-and-answer assistant into a platform that can continuously expand its capabilities.

During the process of promoting and practicing MCP, OpenAI has accumulated rich experience and provided many applicable practices for the industry. Through this case, we can observe how MCP helps developers avoid detours and accelerate the transformation of ideas into practical applications. This not only makes the integration of tools smoother but also makes AI systems more open and flexible.

4.2.2 Cursor: Enhancing Software Development with MCP-Powered Code Assistants. In recent years, AI-assisted programming tools have significantly simplified the software development process. Tools like Cursor, Claude Code, and Cline enable developers to interact with the system using natural language. These assistants can automatically generate code, assist with debugging, and also perform structural reconfiguration. As a result, developers can focus more on business logic, leading to a significant improvement in development efficiency and code quality. The integration of the MCP protocol has further expanded the capabilities of AI-assisted programming tools. Taking Cursor as an example, MCP enables it to directly access external APIs, code repositories, and various automation tools. In practical applications, this process typically includes steps such as instruction parsing, tool scheduling, task distribution, and result feedback. Developers simply input tasks in Cursor, such as "Help me perform a page visit and take a screenshot on qq.com", and Cursor will analyze whether external tool support is needed and request services from the backend through the MCP protocol. The MCP server is responsible for scheduling the appropriate tools, such as Playwright MCP, to automatically complete tasks like page visits and screenshots, and return the results to the agent, ultimately presenting them to the developer. Even for more complex automated testing tasks, such as form submissions, AJAX requests, or multi-page navigation, developers do not need to leave the IDE environment. All processes can be automated and standardized.

The introduction of MCP has greatly enhanced the flexibility and scalability of Cursor. Developers no longer need to write repetitive adaptation codes for each new tool; they only need to declare the tool addresses to access different types of services, regardless of whether these tools are running locally, remotely, or in the cloud. This mechanism reduces the difficulty of expansion and maintenance. Cursor can also automatically coordinate different tools based on the current development scenario, helping developers complete multiple operations. As a result, the development experience becomes smoother and the efficiency is significantly improved. Of course, the integration of MCP also brings some challenges. For example, the data compatibility between different tools needs to be addressed, and permission management and security isolation are even more important. Distributed calls may cause network delays, which have an impact on the overall experience. Despite these issues, after integrating MCP, the intelligence and openness of the development environment have significantly improved. This case helps us understand the profound impact of MCP on actual software development and provides a reference for the future development of the industry.

4.2.3 Cloudflare: Remote MCP Server Hosting and Scalability. Cloudflare transforms MCP from a local-only technology into a scalable, cloud-based solution, as illustrated in Figure 4. By hosting MCP servers in the cloud, Cloudflare removes the need for users to configure and maintain servers on their own machines. This shift lowers the technical barrier for both developers and end users. Cloudflare integrates managed authentication with OAuth 2.0, which ensures that only authorized agents and users can access MCP servers and the tools they provide. The platform also supports persistent state and secure data storage through technologies like Durable Objects and Workers KV. Each MCP session can reliably maintain its own data, even as usage grows or as users switch devices. With these capabilities, MCP servers running on Cloudflare can connect to external APIs, automate multi-step workflows, and interact with a broad range of third-party services. Developers can focus on building useful features, while Cloudflare handles critical concerns such as security, scaling, and network connectivity. This multi-tenant design allows different users or organizations to safely share the same infrastructure without the risk of data leakage.

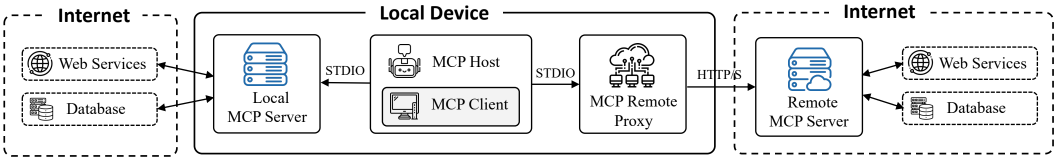

Fig. 4. Architecture of local and remote MCP communication via proxy.

Deploying MCP servers on a cloud platform introduces certain challenges. Developers must carefully define authorization scopes to limit access to sensitive resources. They also need to address privacy and data residency requirements for users in different regions. Real-time agent workflows may require special attention to maintain low latency and stable connections. Cloudflare provides tools to address many of these concerns, but secure tool design and session management remain important responsibilities for developers. Moving MCP to the cloud with Cloudflare extends the value of the protocol to a much wider audience. Both technical and non-technical users can now access secure, scalable AI-powered tools from any device. This cloud-native approach helps MCP become a universal interface for safe, extensible, and cross-device AI automation. Cloudflare's platform shows how modern cloud infrastructure can make advanced AI agent ecosystems accessible and practical in real-world applications.

The adoption of MCP by platforms like OpenAI, Cursor, and Cloudflare highlights its flexibility and growing role in AI-driven workflows, enhancing efficiency, adaptability, and scalability across development tools, enterprise applications, and cloud services.

5 SECURITY AND PRIVACY ANALYSIS

This section provides a comprehensive analysis of the security and privacy risks in the MCP ecosystem. To systematically describe potential vulnerabilities, we categorize the threats based on