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 a standardized interface designed to enable seamless interaction between AI models and external tools and resources, breaking down data silos and facilitating interoperability across diverse systems. This paper provides a comprehensive overview of MCP, focusing on its core components, workflow, and the lifecycle of MCP servers, which consists of three key phases: creation, operation, and update. We analyze the security and privacy risks associated with each phase and propose strategies to mitigate potential threats. The paper also examines the current MCP landscape, including its adoption by industry leaders and various use cases, as well as the tools and platforms supporting its integration. We explore future directions for MCP, highlighting the challenges and opportunities that will influence its adoption and evolution within the broader AI ecosystem. Finally, we offer recommendations for MCP stakeholders to ensure its secure and sustainable development as the AI landscape continues to evolve.

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 (April 2025), 20 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 with the introduction of **function calling** by OpenAI, which allowed language models to invoke external APIs in a structured way [38]. 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** [37], allowing developers to build callable tools for ChatGPT. LLM app stores such as Coze [4] and Yuanqi [50] have launched their **plugin stores**, supporting tools specifically designed for

*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/2025/4-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

their platforms. Frameworks like LangChain [26] and LlamaIndex [29] 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. Additionally, current approaches rely on **predefined workflows, limiting AI agents' flexibility in dynamically discovering and orchestrating tools**.

In late 2024, Anthropic introduced the Model Context Protocol (MCP)[3], a general-purpose protocol standardizing AI-tool interactions. Inspired by the Language Server Protocol (LSP) [22], MCP provides a flexible framework for AI applications to communicate with external tools dynamically. Instead of relying on predefined tool mappings, MCP allows AI agents to autonomously discover, select, and orchestrate tools based on task context. It also supports human-in-the-loop mechanisms, enabling users to inject data or approve actions as needed. By unifying interfaces, MCP simplifies the development of AI applications and improves their flexibility in handling complex workflows. Since its release, MCP has rapidly grown from a niche protocol to a key foundation for AI-native application development. A thriving ecosystem has emerged, with thousands of community-driven MCP servers enabling model access to systems like GitHub [41], Slack [42], and even 3D design tools like Blender [1]. Tools like Cursor [12] and Claude Desktop [2] demonstrate how MCP clients can extend their capabilities by installing new servers, turning developer tools, productivity platforms, and creative environments alike into multi-modal AI agents.

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 in research motivates this paper, which provides the first analysis of the MCP ecosystem, examining its architecture and workflow, defining the lifecycle of MCP servers, and identifying potential security risks at each stage, such as installer spoofing and tool name conflict. 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:

(1) We provide the first analysis of the MCP ecosystem, detailing its architecture, components, and workflow.
(2) We identify the key components of MCP servers and define their lifecycle, encompassing the stages of creation, operation, and update. We also highlight potential security risks associated with each phase, offering insights into safeguarding AI-to-tool interactions.
(3) We examine the current MCP ecosystem landscape, analyzing the adoption, diversity, and use cases across various industries and platforms.
(4) 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 across the MCP server lifecycle 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.**

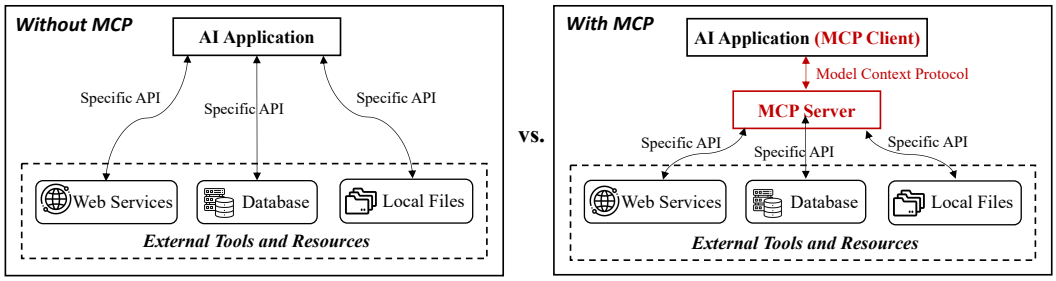

Fig. 1. Tool invocation with and without MCP.

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 [37], 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**. New LLM app stores [62] such as ByteDance Coze [4] and Tencent Yuanqi [50] have also emerged, offering a plugin store for web services. While these platforms expanded available tool options, they created isolated ecosystems where plugins are **platform-specific**, limiting cross-platform compatibility and requiring duplicate maintenance efforts. MCP stands out by being open-source and platform-agnostic, enabling AI applications to engage in rich two-way interactions with external tools, facilitating complex workflows.

2.1.3 AI Agent Tool Integration. The emergence of AI agent frameworks like LangChain [26] and similar tool orchestration frameworks provided a structured way for models to invoke external tools through predefined interfaces, improving automation and adaptability [55]. However, integrating and maintaining these tools remained largely manual, requiring custom implementations and

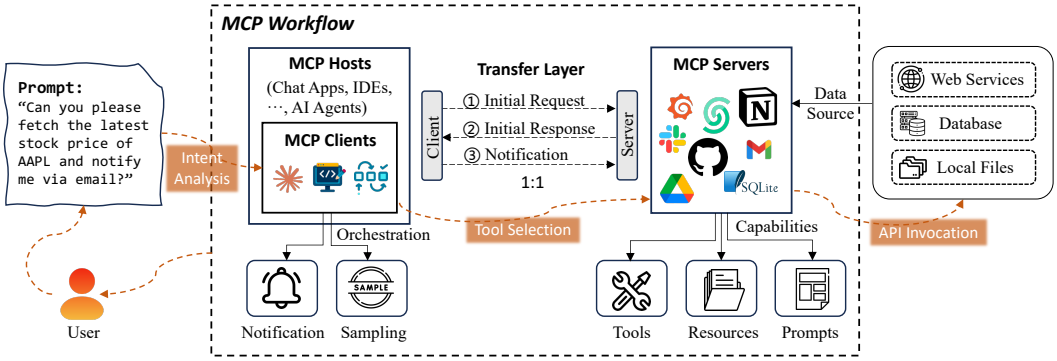

Fig. 2. The workflow of MCP.

3.1.2 MCP Client. The MCP client acts as an intermediary within the host environment, managing communication between the MCP host and one or more MCP servers. It initiates requests to MCP servers, queries available functions, and retrieves responses that describe the server's capabilities. This ensures seamless interaction between the host and external tools. 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 allow the MCP server to invoke external services and APIs to execute operations on behalf of AI models. When the client requests an operation, the MCP server identifies the appropriate tool, interacts with the service, and returns the result. For instance, if an AI model requires real-time weather data or sentiment analysis, the MCP server connects to the relevant API, retrieves the data, and delivers it to the host. Unlike traditional function calling, which requires multiple steps and separates invocation from execution, Tools of MCP servers streamline this process by allowing the model to autonomously select and invoke the appropriate tool based on context. Once configured, these tools follow a standardized supply-and-consume model, making them modular, reusable, and easily accessible to other applications, enhancing system efficiency and flexibility.
- **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

The MCP server lifecycle as shown in Figure 3 consists of three key phases: ***creation***, ***operation***, and ***update***. Each phase defines critical activities that ensure the secure and efficient functioning of the MCP server, enabling seamless interaction between AI models and external tools, resources, and prompts.

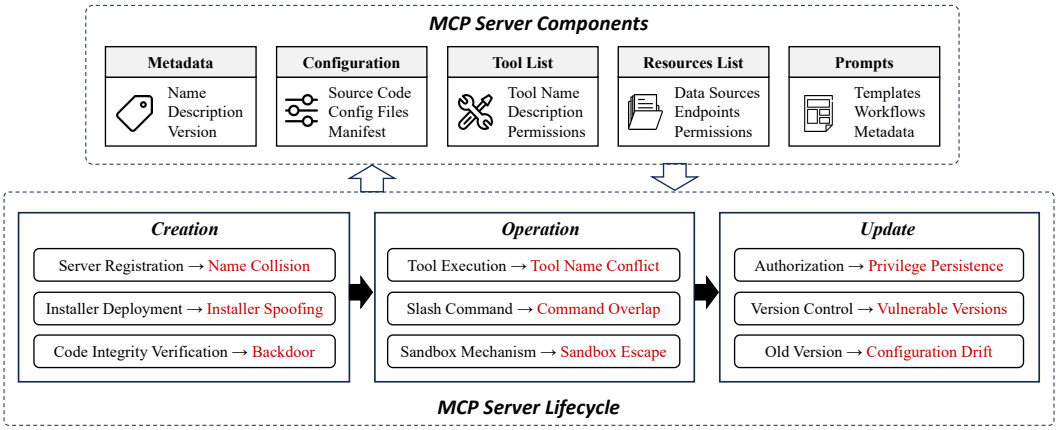

Fig. 3. MCP servers components and lifecycle.

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

Table 2. Overview of MCP server collections and deployment modes (As of March 27, 2025).

Collection	Author	Mode	# Servers	URL
MCP.so	mcp.so	Website	4774	mcp.so
Glama	glama.ai	Website	3356	glama.ai
PulseMCP	Antanavicius et al.	Website	3164	pulsemcp.com
Smithery	Henry Mao	Website	2942	smithery.ai
Dockmaster	mcp-dockmaster	Desktop App	517	mcp-dockmaster.com
Official Collection	**Anthropic**	**GitHub Repo**	**320**	modelcontextprotocol/servers
AiMCP	Hekmon	Website	313	aimcp.info
MCP.run	mcp.run	Website	114	mcp.run
Awesome MCP Servers	Stephen Akinyemi	GitHub Repo	88	appcypher/mcp-servers
mcp-get registry	Michael Latman	Website	59	mcp-get.com
Awesome MCP Servers	wong2	Website	34	mcpservers.org
OpenTools	opentoolsteam	Website	25	opentools.com
Toolbase	gching	Desktop App	24	gettoolbase.ai
make inference	mkinf	Website	20	mkinf.io
Awesome Crypto MCP Servers	Luke Fan	GitHub Repo	13	badkk/crypto-mcp-servers

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. OpenAI has adopted MCP to standardize AI-to-tool communication, recognizing its potential to enhance integration with external tools. Recently, OpenAI introduced MCP support in its Agent SDK, enabling developers to create AI agents that seamlessly interact with external tools. In a typical workflow, developers use the Agent SDK to define tasks that require external tool invocation. When an AI agent encounters a task like retrieving data from an API or querying a database, the SDK routes the request through an MCP server. The request is transmitted via the MCP protocol, ensuring proper formatting and real-time response delivery to the agent. OpenAI's plan to integrate MCP into the Responses API will streamline AI-to-tool communication, allowing AI models like ChatGPT to interact with tools dynamically without extra configuration. Additionally, OpenAI aims to extend MCP support to ChatGPT desktop applications, enabling AI assistants to handle various user tasks by connecting to remote MCP servers, further bridging the gap between AI models and external systems.

4.2.2 Cursor: Enhancing Software Development with MCP-Powered Code Assistants. Cursor uses MCP to enhance software development by enabling AI-powered code assistants that automate complex tasks. With MCP, Cursor allows AI agents to interact with external APIs, access code