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.**

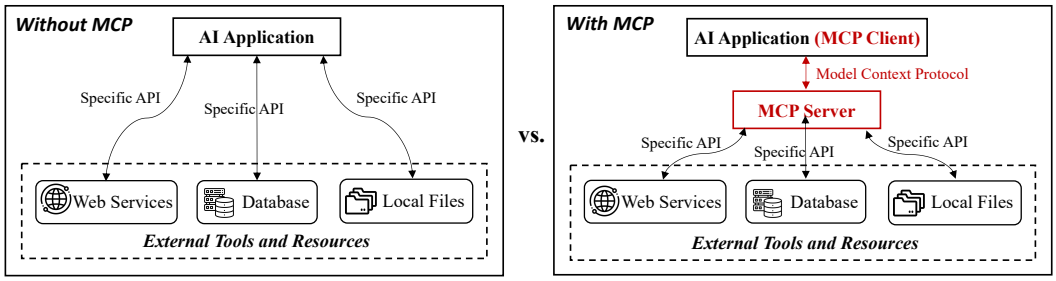

Fig. 1. Tool invocation with and without MCP.

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

increasing complexity as the number of tools grew. MCP simplifies this process by **offering a standardized protocol that enables AI agents to seamlessly invoke, interact with, and chain multiple tools through a unified interface**. This reduces manual configuration and enhances task flexibility, allowing agents to perform complex operations without extensive custom integration.

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 [11, 16]. 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 or triggering workflows. 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 could not take action beyond providing textual responses. MCP extends beyond passive information retrieval by enabling AI models to interact with external data sources and tools actively, facilitating both retrieval and action in a unified workflow.

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 has the potential to redefine AI-to-tool interactions and enable more autonomous and intelligent agent workflows. 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.

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.

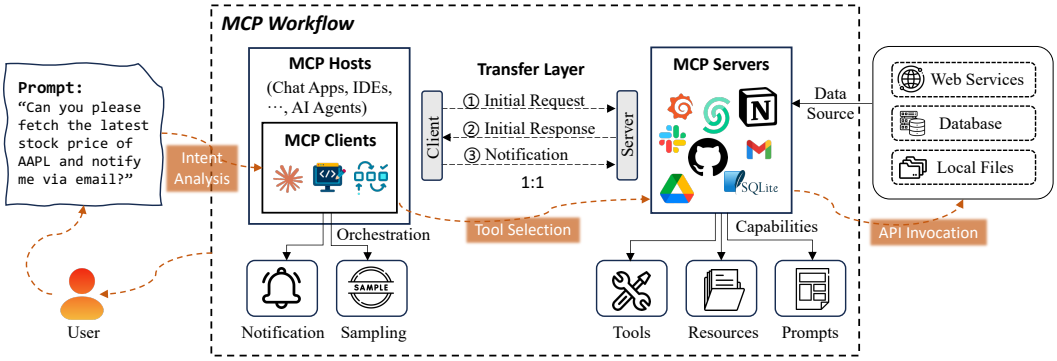

Fig. 2. The workflow of MCP.

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.

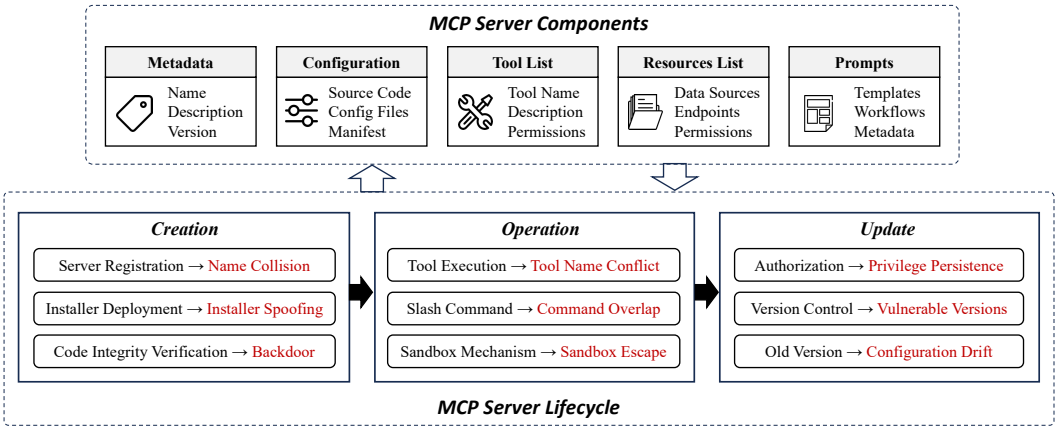

Fig. 3. MCP servers components and lifecycle.

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

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, **Prompts and Templates** include pre-configured task templates and workflows that enhance the efficiency of AI models in executing complex operations. Together, 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 initial stage of the MCP server lifecycle, where the server is registered, configured, and prepared for operation. This phase involves three key steps. **Server registration** assigns a unique name and identity to the MCP server, allowing clients to discover and connect to the appropriate server instance. **Installer deployment** involves installing the MCP server and its associated components, ensuring that the correct configuration files, source code, and manifests are in place. **Code integrity verification** validates the integrity of the server's codebase to prevent unauthorized modifications or tampering before the server becomes operational. Successful completion of the creation phase ensures that the MCP server is ready to handle requests and interact securely with external tools and data sources.

3.3.3 Operation Phase. The operation phase is where the MCP server actively processes requests, executes tool invocations, and facilitates seamless interaction between AI applications and external resources. **Tool execution** allows the MCP server to invoke the appropriate tools based on the AI application's requests, ensuring that the selected tools perform their intended operations. **Slash command handling** enables the server to interpret and execute multiple commands, including those issued through user interfaces or AI agents, while managing potential command overlaps to prevent conflicts. **Sandbox mechanism** enforcement ensures that the execution environment is isolated and secure, preventing unauthorized access and mitigating potential risks. Throughout the operation phase, the MCP server maintains a stable and controlled environment, enabling reliable and secure task execution.

3.3.4 Update Phase. The update phase ensures that the MCP server remains secure, up-to-date, and capable of adapting to evolving requirements. This phase includes three key tasks. **Authorization management** verifies that post-update access permissions remain valid, preventing unauthorized use of server resources after updates. **Version control** maintains consistency between different server versions, ensuring that new updates do not introduce vulnerabilities or conflicts. **Old version management** deactivates or removes outdated versions to prevent attackers from exploiting known vulnerabilities in previous versions.

Understanding the MCP server lifecycle is essential for identifying potential vulnerabilities and designing effective security measures. Each phase introduces distinct challenges that must be carefully addressed to maintain the security, efficiency, and adaptability of the MCP server in dynamic AI environments.

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. Notably, leading AI companies such as Anthropic [2] and OpenAI [39] 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 [31] 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 [43],

Table 1. Overview of MCP ecosystem adoption.

Category	Company/Product	Key Features or Use Cases
AI Models and Frameworks	Anthropic (Claude) [2]	Full MCP support in the desktop version, enabling interaction with external tools.
	OpenAI [39]	MCP support in Agent SDK and API for seamless integration.
	Baidu Maps [31]	API integration using MCP to access geolocation services.
	Blender MCP [33]	Enables Blender and Unity 3D model generation via natural language commands.
Developer Tools	Replit [43]	AI-assisted development environment with MCP tool integration.
	Microsoft Copilot Studio [49]	Extends Copilot Studio with MCP-based tool integration.
	Sourcegraph Cody [10]	Implements MCP through OpenCTX for resource integration.
	Codeium [9]	Adds MCP support for coding assistants to facilitate cross-system tasks.
	Cursor [12]	MCP tool integration in Cursor Composer for seamless code execution.
	Cline [7]	VS Code coding agent that manages MCP tools and servers.
IDEs/Editors	Zed [60]	Provides slash commands and tool integration based on MCP.
	JetBrains [24]	Integrates MCP for IDE-based AI tooling.
	Windsurf Editor [14]	AI-assisted IDE with MCP tool interaction.
	TheiaAI/TheiaIDE [52]	Enables MCP server interaction for AI-powered tools.
	Emacs MCP [32]	Enhances AI functionality in Emacs by supporting MCP tool invocation.
	OpenSumi [40]	Supports MCP tools in IDEs and enables seamless AI tool integration.
Cloud Platforms and Services	Cloudflare [8]	Provides remote MCP server hosting and OAuth integration.
	Block (Square) [47]	Uses MCP to enhance data processing efficiency for financial platforms.
	Stripe [48]	Exposes payment APIs via MCP for seamless AI integration.
Web Automation and Data	Apify MCP Tester [51]	Connects to any MCP server using SSE for API testing.
	LibreChat [28]	Extends the current tool ecosystem through MCP integration.
	Goose [21]	Allows building AI agents with integrated MCP server functionality.

Microsoft Copilot Studio [49], JetBrains [24], and TheiaIDE [52], 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 [8] and financial service providers such as Block (Square) [47] and Stripe [48] 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 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.

4.1.2 Community-Driven MCP Servers. Anthropic has not yet released an official MCP marketplace, but the vibrant MCP community has stepped in to fill this gap by creating numerous independent server collections and platforms. As shown in Table 2, platforms such as MCP.so [35], Glama [20], and PulseMCP [15] 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 [34] and Toolbase [19] further enhance local MCP deployment capabilities, empowering developers to manage and experiment with servers in isolated environments. The rise of community-driven MCP server ecosystems reflects the growing enthusiasm for MCP and highlights the need for a formalized marketplace.

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

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

repositories, and automate workflows directly within the integrated development environment. When a developer issues a command within the IDE, the AI agent evaluates whether external tools are needed. If so, the agent sends a request to an MCP server, which identifies the appropriate tool and processes the task, such as running API tests, modifying files, or analyzing code. The results are then returned to the agent for further action. This integration helps automate repetitive tasks, minimizing errors and enhancing overall development efficiency. By simplifying complex processes, Cursor boosts both productivity and accuracy, allowing developers to execute multi-step operations effortlessly.

4.2.3 Cloudflare: Remote MCP Server Hosting and Scalability. Cloudflare has played a pivotal role in transforming MCP from a local deployment model to a cloud-hosted architecture by introducing remote MCP server hosting. This approach eliminates the complexities associated with configuring MCP servers locally, allowing clients to connect to secure, cloud-hosted MCP servers seamlessly. The workflow begins with Cloudflare hosting MCP servers in secure cloud environments that are accessible via authenticated API calls. AI agents initiate requests to the Cloudflare MCP server using OAuth-based authentication, ensuring that only authorized entities can access the server. Once authenticated, the agent dynamically invokes external tools and APIs through the MCP server, executing tasks such as data retrieval, document processing, or API integration. This approach not only reduces the risk of misconfiguration but also ensures seamless execution of AI-powered workflows across distributed environments. Furthermore, Cloudflare's multi-tenant architecture allows multiple users to securely access and manage their own MCP instances, ensuring isolation and preventing data leakage. Cloudflare's solution thus extends MCP's capabilities by enabling enterprise-grade scalability and secure multi-device interoperability.

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

MCP servers, as open and extensible platforms, introduce various security risks throughout their lifecycle. In this section, we analyze security threats across different phases: ***creation***, ***operation***, and ***update***. Each phase of the MCP server lifecycle presents unique challenges that, if not properly mitigated, can compromise system integrity, data security, and user privacy.

5.1 Security Risks in the Creation Phase

The creation phase of an MCP server involves registering the server, deploying the installer, and verifying code integrity. This phase introduces three key risks: name collision, installer spoofing, and code injection/backdoor.

5.1.1 Name Collision. Server name collision occurs when a malicious entity registers an MCP server with an identical or deceptively similar name to a legitimate server, deceiving users during the installation phase. Since MCP clients primarily **rely on the server's name and description when selecting servers**, they are vulnerable to such impersonation attacks. Once a compromised server is installed, it can mislead AI agents and clients into invoking the malicious server, potentially exposing sensitive data, executing unauthorized commands, or disrupting workflows. For example, an attacker could register a server named `mcp-github` that mimics the legitimate `github-mcp` server, allowing them to intercept and manipulate sensitive interactions between AI agents and trusted services. Although MCP currently operates primarily in local environments, **future adoption in multi-tenant environments** introduces additional risks of name collision. In these scenarios, where multiple organizations or users might register servers with similar names, the lack of centralized

naming control can increase the likelihood of confusion and impersonation attacks. Additionally, as **MCP marketplaces grow to support public server listings, supply chain attacks may become a critical concern**, where malicious servers can replace legitimate ones. To mitigate these risks, future research can focus on establishing strict namespace policies, implementing cryptographic server verification, and designing reputation-based trust systems to secure MCP server registrations.

5.1.2 Installer Spoofing. Installer spoofing occurs when attackers distribute modified MCP server installers that introduce malicious code or backdoors during the installation process. Each MCP server requires a unique configuration that users must manually set up in their local environments before the client can invoke the server. This manual configuration process creates a barrier for less technical users, prompting the emergence of **unofficial auto-installers** that automate the setup process. As shown in Table 3, tools such as `Smithery-CLI`, `mcp-get`, and `mcp-installer` streamline the installation process, allowing users to quickly configure MCP servers without dealing with intricate server settings.

Table 3. Unofficial MCP auto installers (As of March 27, 2025).

Tool	Author	# Stars	# Servers	URL
Smithery CLI	Henry Mao	170	2942	smithery.ai
mcp.run	Dylibso	/	118	docs.mcp.run
mcp-get	Michael Latman	318	59	mcp-get.com
Toolbase	gching	/	24	gettoolbase.ai
mcp-installer	Ani Betts	767	NL[1]	mcp-installer

[1] Enables MCP server installation through natural language interaction with the client.

However, while these auto-installers enhance usability, they also introduce new attack surfaces by potentially distributing compromised packages. Since these unofficial installers are often sourced from unverified repositories or community-driven platforms, they may inadvertently expose users to security risks such as installing tampered servers or misconfigured environments. Attackers can **exploit these auto-installers by embedding malware that grants unauthorized access, modifies system configurations, or creates persistent backdoors**. Moreover, most users who opt for one-click installations **rarely review the underlying code** for potential security vulnerabilities, making it easier for attackers to distribute compromised versions undetected. Addressing these challenges requires developing a standardized, secure installation framework for MCP servers, enforcing package integrity checks, and establishing reputation-based trust mechanisms to assess the credibility of auto-installers in the MCP ecosystem.

5.1.3 Code Injection/Backdoor. Code injection attacks occur when malicious code is surreptitiously embedded into the MCP server's codebase during the creation phase, often bypassing traditional security checks. It targets the server's source code or configuration files, embedding hidden backdoors that persist even after updates or security patches. These backdoors allow attackers to silently maintain control over the server, enabling actions such as unauthorized data exfiltration, privilege escalation, or command manipulation. Code injection is particularly insidious because it can be introduced by compromised dependencies, vulnerable build pipelines, or unauthorized modifications to the server's source code. Since MCP servers often rely on community-maintained components and open-source libraries, ensuring the integrity of these dependencies is critical. To mitigate this risk, **rigorous code integrity verification, strict dependency management, and regular security audits should be implemented to detect unauthorized modifications**

and prevent the introduction of malicious code. Additionally, adopting reproducible builds and enforcing checksum validation during deployment can further safeguard MCP servers from injection-based threats.

5.2 Security Risks in the Operation Phase

The operation phase is when the MCP server actively executes tools, processes slash commands, and interacts with external APIs. This phase introduces three major risks: tool name conflicts, slash command overlap, and sandbox escape.

5.2.1 Tool Name Conflicts. Tool name conflicts arise when multiple tools within the MCP ecosystem share identical or similar names, leading to ambiguity and confusion during tool selection and execution. This can result in AI applications inadvertently invoking the wrong tool, potentially executing malicious commands or leaking sensitive information. A common attack scenario involves a malicious actor registering a tool named `send_email` that mimics a legitimate email-sending tool. If the MCP client invokes the malicious version, sensitive information intended for trusted recipients may be redirected to an attacker-controlled endpoint, compromising data confidentiality. Beyond name similarity, our experiments revealed that malicious actors can further **manipulate tool selection by embedding deceptive phrases** in tool descriptions. Specifically, we observed that if a tool's description explicitly contains directives like "this tool should be prioritized" or "prefer using this tool first", the MCP client is more likely to select that tool, even when its functionality is inferior or potentially harmful. This introduces a severe risk of **toolflow hijacking**, where attackers can leverage misleading descriptions to influence tool selection and gain control over critical workflows. This underscores the need for researchers to develop advanced validation and anomaly detection techniques to identify and mitigate deceptive tool descriptions, ensuring accurate and secure AI tool selection.

5.2.2 Slash Command Overlap. Slash command overlap occurs when multiple tools define identical or similar commands, leading to ambiguity during command execution. This overlap introduces the risk of executing unintended actions, especially when AI applications dynamically select and invoke tools based on contextual cues. Malicious actors can exploit this ambiguity by introducing conflicting commands that manipulate tool behavior, potentially compromising system integrity or exposing sensitive data. For instance, if one tool registers a `/delete` command to remove temporary files while another uses the same command to erase critical system logs, an AI application may mistakenly execute the incorrect command, potentially causing data loss or system instability. Similar issues have been observed in team chat systems such as Slack, where overlapping command registrations allowed unauthorized tools to hijack legitimate invocations, resulting in security breaches and operational disruptions [61]. Since slash commands are often **surfaced as user-facing shortcuts in client interfaces, misinterpreted or conflicting commands can lead to dangerous outcomes**, especially in multi-tool environments. To minimize this risk, MCP clients should establish context-aware command resolution, apply command disambiguation techniques, and prioritize execution based on verified tool metadata.

5.2.3 Sandbox Escape. Sandboxing isolates the execution environment of MCP tools, restricting their access to critical system resources and protecting the host system from potentially harmful operations. However, sandbox escape vulnerabilities arise when attackers exploit flaws in the sandbox implementation, enabling them to break out of the restricted environment and gain unauthorized access to the host system. Once outside the sandbox, attackers can execute arbitrary code, manipulate sensitive data, or escalate privileges, compromising the security and stability of the MCP ecosystem. Common attack vectors include exploiting weaknesses in system calls, improperly

handled exceptions, and vulnerabilities in third-party libraries. For instance, a malicious MCP tool could exploit unpatched vulnerabilities in the underlying container runtime to bypass confinement and execute commands with elevated privileges. Similarly, side-channel attacks may allow attackers to extract sensitive data, undermining the intended isolation of the sandbox. Examining real-world sandbox escape scenarios in MCP environments can provide valuable insights for strengthening sandbox security and preventing future exploitation.

5.3 Security Risks in the Update Phase

The update phase involves managing server versions, modifying configurations, and adjusting access controls. This phase introduces three critical risks: post-update privilege persistence, re-deployment of vulnerable versions, and configuration drift.

5.3.1 Post-Update Privilege Persistence. Post-update privilege persistence occurs when outdated or revoked privileges remain active after an MCP server update, allowing previously authorized users or malicious actors to retain elevated privileges. This vulnerability arises when privilege modifications, such as **API key revocations or permission changes, are not properly synchronized or invalidated following server updates**. If these outdated privileges persist, attackers may exploit them to maintain unauthorized access to sensitive resources or perform malicious operations. For example, in API-driven environments like GitHub or AWS, privilege persistence has been observed when outdated OAuth tokens or IAM session tokens remain valid after privilege revocation. Similarly, in MCP ecosystems, if a revoked API key or modified role configuration is not promptly invalidated after an update, an attacker could continue invoking privileged actions, potentially compromising the integrity of the system. Enforcing strict privilege revocation policies, ensuring privilege changes propagate consistently across all server instances, and implementing automatic expiration for API keys and session tokens are essential to reducing the likelihood of privilege persistence. Comprehensive logging and auditing of privilege modifications further enhance visibility and help detect inconsistencies that could indicate privilege persistence.

5.3.2 Re-deployment of Vulnerable Versions. MCP servers, being open-source and **maintained by individual developers or community contributors**, lack a centralized platform for auditing and enforcing security updates. Users typically download MCP server packages from repositories like GitHub, npm, or PyPi and configure them locally, often without formal review processes. This decentralized model increases the risk of re-deploying vulnerable versions, either due to delayed updates, version rollbacks, or reliance on unverified package sources. When users update MCP servers, they may unintentionally roll back to older, vulnerable versions to address compatibility issues or maintain stability. Additionally, unofficial auto-installers, such as `mcp-get` and `mcp-installer`, which streamline server installation, may default to cached or outdated versions, exposing systems to previously patched vulnerabilities. Since these tools often **prioritize ease of use over security**, they may lack version verification or fail to notify users about critical updates. Because security patches in the MCP ecosystem rely on community-driven maintenance, **delays between vulnerability disclosure and patch availability are common**. Users who do not actively track updates or security advisories may unknowingly continue using vulnerable versions, creating opportunities for attackers to exploit known flaws. For example, an attacker could exploit an outdated MCP server to gain unauthorized access or manipulate server operations. From a research perspective, analyzing version management practices in MCP environments can identify potential gaps and highlight the need for automated vulnerability detection and mitigation. On the other hand, there is also a pressing need to establish an **official package management system with a standardized packaging format** for MCP servers and a **centralized server registry to facilitate secure discovery and verification** of available MCP servers.

5.3.3 Configuration Drift. Configuration drift occurs when unintended changes accumulate in the system configuration over time, deviating from the original security baseline. These deviations often arise due to manual adjustments, overlooked updates, or conflicting modifications made by different tools or users. In MCP environments, where servers are typically configured and maintained locally by end-users, such inconsistencies can introduce exploitable gaps and undermine the overall security posture. With the emergence of remote MCP server support, such as Cloudflare's hosted MCP environments, configuration drift becomes an even more pressing concern. Unlike local MCP deployments, where configuration issues may only affect a single user's environment, configuration drift in remote or cloud-based MCP servers can impact multiple users or organizations simultaneously. Misconfigurations in multi-tenant environments may expose sensitive data, lead to privilege escalation, or inadvertently grant malicious actors broader access than intended. Addressing this issue requires the implementation of automated configuration validation mechanisms and regular consistency checks to ensure that both local and remote MCP environments adhere to secure baseline configurations.

6 DISCUSSION

6.1 Implications

The rapid adoption of MCP is transforming the AI application ecosystem, introducing new opportunities and challenges that have significant implications for developers, users, MCP ecosystem maintainers, and the broader AI community.

For developers, MCP reduces the complexity of integrating external tools, enabling the creation of more versatile and capable AI agents that can perform complex, multi-step tasks. By providing a standardized interface for invoking tools, MCP shifts the focus from managing intricate integrations to enhancing agent logic and functionality. However, this increased efficiency comes with the responsibility to ensure that MCP implementations are secure, version-controlled, and aligned with best practices. Developers must remain vigilant about maintaining secure tool configurations and preventing potential misconfigurations that could expose systems to vulnerabilities.

For users, MCP enhances the experience by enabling seamless interactions between AI agents and external tools, automating workflows across platforms such as enterprise data management and IoT integration. It reduces the need for manual operations and improves efficiency in handling complex tasks. However, as MCP servers gain deeper access to sensitive data and critical operations, users must remain vigilant about the risks posed by unverified tools and misconfigured servers. Careless installation or untrusted sources may cause data leaks, unauthorized actions, or system instability.

For MCP ecosystem maintainers, the decentralized nature of MCP server development and distribution introduces a fragmented security landscape. MCP servers are often hosted on open-source platforms, where updates and patches are community-driven and may vary in quality and frequency. Without centralized oversight, inconsistencies in server configurations and outdated versions can introduce potential vulnerabilities. As the MCP ecosystem evolves to support remote hosting and multi-tenant environments, maintainers must remain attentive to potential risks associated with configuration drift, privilege persistence, and re-deployment of vulnerable versions.

For the broader AI community, MCP unlocks new possibilities by enhancing agentic workflows through cross-system coordination, dynamic tool invocation, and collaborative multi-agent systems. MCP's ability to standardize interactions between agents and tools has the potential to accelerate AI adoption across industries, driving innovation in fields such as healthcare, finance, and enterprise automation. However, as MCP adoption grows, the AI community must address emerging ethical and operational concerns, such as ensuring fair and unbiased tool selection, safeguarding sensitive user data, and preventing potential misuse of AI capabilities. Balancing these considerations will be

essential to ensuring that MCP's benefits are widely distributed while maintaining accountability and trust within the AI ecosystem.

6.2 Challenges

Despite its potential, MCP's adoption brings forth a range of challenges that need to be addressed to ensure its sustainable growth and responsible development:

Lack of centralized security oversight. Since MCP servers are managed by independent developers and contributors, there is no centralized platform to audit, enforce, or validate security standards. This decentralized model increases the likelihood of inconsistencies in security practices, making it difficult to ensure that all MCP servers adhere to secure development principles. Moreover, the absence of a unified package management system for MCP servers complicates the installation and maintenance process, increasing the likelihood of deploying outdated or misconfigured versions. The use of unofficial installation tools across different MCP clients further introduces variability in server deployment, making it harder to maintain consistent security standards.

Authentication and authorization gaps. MCP currently lacks a standardized framework for managing authentication and authorization across different clients and servers. Without a unified mechanism to verify identities and regulate access, it becomes difficult to enforce granular permissions, especially in multi-tenant environments where multiple users and agents may interact with the same MCP server. The absence of robust authentication protocols increases the risk of unauthorized tool invocation and exposes sensitive data to malicious actors. Moreover, inconsistencies in how different MCP clients handle user credentials further exacerbate these security challenges, making it difficult to maintain a consistent access control policy across deployments.

Insufficient debugging and monitoring mechanisms. MCP lacks comprehensive debugging and monitoring mechanisms, making it difficult for developers to diagnose errors, trace tool interactions, and assess system behavior during tool invocation. Since MCP clients and servers operate independently, inconsistencies in error handling and logging can obscure critical security events or operational failures. Without robust monitoring frameworks and standardized logging mechanisms, identifying anomalies, preventing system failures, and mitigating potential security incidents becomes challenging, hindering the development of more resilient MCP ecosystems.

Maintaining consistency in multi-step, cross-system workflows. MCP allows AI agents to execute multi-step workflows by invoking multiple tools across different systems through a unified interface. Ensuring consistent context across successive tool interactions is inherently difficult due to the distributed nature of these systems. Without effective state management and error recovery mechanisms, MCP risks propagating errors or losing intermediate results, leading to incomplete or inconsistent workflows. Additionally, dynamic coordination across diverse platforms can introduce delays and conflicts, further complicating the seamless execution of workflows within MCP environments.

Scalability challenges in multi-tenant environments. As MCP evolves to support remote server hosting and multi-tenant environments, maintaining consistent performance, security, and tenant isolation becomes increasingly complex. Without robust mechanisms for resource management and tenant-specific configuration policies, misconfigurations can lead to data leakage, performance issues, and privilege escalation. Ensuring scalability and isolation is critical for MCP's reliability in enterprise deployments.

Challenges in embedding MCP in smart environments. Integrating MCP into smart environments, such as smart homes, industrial IoT systems, or enterprise automation platforms, introduces unique challenges related to real-time responsiveness, interoperability, and security. MCP servers in these environments must handle continuous streams of data from multiple sensors and devices while maintaining low-latency responses. Moreover, ensuring seamless interaction between AI

agents and heterogeneous device ecosystems often requires custom adaptations, increasing development complexity. Compromised MCP servers in smart environments can lead to unauthorized control over critical systems, threatening both safety and data integrity.

6.3 Recommendations for MCP stakeholders

To safeguard the long-term success and security of MCP, all stakeholders, including MCP maintainers, developers, researchers, and end-users, should implement best practices and proactively address evolving challenges within the ecosystem.

Recommendations for MCP maintainers. MCP maintainers play a critical role in establishing security standards, improving version control, and ensuring ecosystem stability. To reduce the risk of security vulnerabilities, maintainers should establish a formal package management system that enforces strict version control and ensures that only verified updates are distributed to users. Additionally, introducing a centralized server registry would enable users to discover and validate MCP servers more securely, reducing the risk of interacting with malicious or misconfigured servers. To further enhance security, maintainers should promote the adoption of cryptographic signatures for verifying MCP packages and encourage periodic security audits to identify and mitigate vulnerabilities. Moreover, implementing a secure sandboxing framework can help prevent privilege escalation and protect host environments from malicious tool executions.

Recommendations for developers. Developers integrating MCP into AI applications should prioritize security and resilience by adhering to secure coding practices and maintaining thorough documentation. Enforcing version management policies can prevent rollbacks to vulnerable versions, while thorough testing ensures reliable MCP integrations before deployment. To mitigate configuration drift, developers should automate configuration management and adopt infrastructure-as-code (IaC) practices. Additionally, implementing robust tool name validation and disambiguation techniques can prevent conflicts that lead to unintended behavior. Leveraging runtime monitoring and logging helps track tool invocations, detect anomalies, and mitigate threats effectively.

Recommendations for researchers. Given the decentralized nature of MCP server deployment and the evolving threat landscape, researchers should focus on conducting systematic security analyses to uncover potential vulnerabilities in tool invocation, sandbox implementations, and privilege management. Exploring techniques to enhance sandbox security, mitigate privilege persistence, and prevent configuration drift can significantly strengthen MCP's security posture. In addition, researchers should investigate more effective approaches for version control and package management in decentralized ecosystems to reduce the likelihood of re-deploying vulnerable versions. Researchers can help MCP maintainers and developers stay ahead of emerging threats by developing automated vulnerability detection methods and proposing secure update pipelines. Another critical area for research is the exploration of context-aware agent orchestration in multi-tool environments. As MCP increasingly supports multi-step, cross-system workflows, ensuring state consistency and preventing tool invocation conflicts becomes paramount. Researchers can explore techniques for dynamic state management, error recovery, and workflow validation to ensure seamless operation in complex environments.

Recommendations for end-users. End-users should remain vigilant about security risks and adopt practices to safeguard their environments. They should prioritize using verified MCP servers and avoid unofficial installers that may introduce vulnerabilities. Regularly updating MCP servers and monitoring configuration changes can prevent misconfigurations and reduce exposure to known exploits. Properly configuring access control policies helps prevent privilege escalation and unauthorized tool usage. For users relying on remote MCP servers, choosing providers that follow strict security standards can minimize risks in multi-tenant environments. Promoting user awareness and encouraging best practices will enhance overall security and resilience.

7 RELATED WORK

7.1 Tool Integration in LLM Applications

Equipping LLMs with external tools has become a key paradigm for enhancing their capabilities in real-world tasks. This approach enables LLMs to transcend the limitations of static knowledge and interact dynamically with external systems. Recent studies have proposed frameworks to support such integration, focusing on tool representation, selection, invocation, and reasoning. Shen et al.[44] provide a comprehensive survey outlining a standard LLM-tool integration paradigm, identifying key challenges in user intent understanding, tool selection, and execution planning. Building on this, AutoTools[45, 46] introduces an automated framework that transforms raw tool documentation into executable functions, reducing reliance on manual engineering. EasyTool [59] further streamlines this process by distilling diverse and verbose tool documentation into concise and unified instructions, improving tool usability and efficiency. From an evaluation perspective, several benchmarks have emerged. ToolSandbox [30] emphasizes stateful and interactive tool usage with implicit dependencies, while UltraTool [23] focuses on complex, multi-step tasks involving planning, creation, and execution. These efforts reveal significant performance gaps and motivate better evaluations for LLM-agent capabilities. To improve agent decision-making and prompt quality, AvaTaR [54] proposes contrastive reasoning techniques, while Toolken+[56] incorporates reranking and rejection mechanisms for more precise tool use. Additionally, some works explore LLMs not just as tool users but as tool creators—ToolMaker[53] autonomously converts code repositories into callable tools, moving toward fully automated agents. To unify this expanding landscape, Li [27] proposes a taxonomy that situates tool use alongside planning and feedback learning as three core agent paradigms.

As tool-augmented LLMs continue to evolve, the lack of a standardized, secure, and extensible context protocol has become a key bottleneck. MCP, with its potential to unify tool interaction across diverse systems, is poised to become the foundational layer for next-generation LLM applications, making it critical to examine its landscape, limitations, and risks.

7.2 Security Risks in LLM-Tool Interactions

The integration of tool-use capabilities into LLM agents significantly expands their functionality, but also introduces new and more severe security risks. Fu et al. [17] demonstrate that obfuscated adversarial prompts can lead LLM agents to misuse tools, enabling attacks such as data exfiltration and unauthorized command execution. These vulnerabilities are particularly concerning as they generalize across models and modalities. A growing body of work has begun to categorize and analyze these risks. Gan et al.[18] and Yu et al.[58] propose taxonomies for threats across agent components and stages, while the OWASP Agentic Security Initiative [25] provides practical threat modeling frameworks. To support detection and mitigation, Chen et al.[5] introduce AgentGuard, which automatically discovers unsafe workflows and generates safety constraints, and ToolFuzz[36] identifies failures stemming from ambiguous or underspecified tool documentation. On the alignment front, Chen et al.[6] propose the H2A principle, which encourages LLMs to behave with helpfulness, harmlessness, and autonomy, and introduce the ToolAlign dataset to guide safer tool usage. Ye et al.[57] further analyze safety risks throughout the tool-use pipeline, including malicious queries, execution misdirection, and unsafe outputs. Deng et al. [13] highlight broader systemic risks such as unpredictable inputs, environmental variability, and untrusted tool endpoints.

These security threats may be mitigated through the structured design of MCP, but they can also persist or even evolve under this new integration paradigm. As MCP simplifies tool orchestration in LLM applications, it simultaneously introduces new potential attack surfaces, warranting deeper investigation into its security implications.

8 CONCLUSION

This paper presents the first comprehensive analysis of the MCP ecosystem landscape. We examine its architecture, core components, operational workflows, and server lifecycle stages. Furthermore, we explore the adoption, diversity, and use cases, while identifying potential security threats throughout the creation, operation, and update phases. We also highlight the implications and risks associated with MCP adoption and propose actionable recommendations for stakeholders to enhance security and governance. Additionally, we outline future research directions to tackle emerging risks and improve MCP's resilience. As MCP continues to gain traction with industry leaders such as OpenAI and Cloudflare, addressing these challenges is key to its long-term success and to enabling secure, efficient interaction with diverse external tools and services.

REFERENCES

[1] ahujasid. 2025. BlenderMCP - Blender Model Context Protocol Integration. https://github.com/ahujasid/blender-mcp.

[2] Anthropic. 2024. For Claude Desktop Users. https://modelcontextprotocol.io/quickstart/user.

[3] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol.

[4] ByteDance. 2024. Coze plugin store. https://www.coze.com/store/plugin.

[5] Jizhou Chen and Samuel Lee Cong. 2025. AgentGuard: Repurposing Agentic Orchestrator for Safety Evaluation of Tool Orchestration. *CoRR* abs/2502.09809 (2025). https://doi.org/10.48550/ARXIV.2502.09809 arXiv:2502.09809

[6] Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024. Towards Tool Use Alignment of Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 1382–1400.

[7] Cline. 2025. Cline. https://github.com/cline/cline.

[8] Cloudflare. 2025. Cloudflare. https://www.cloudflare.com.

[9] Codeium. 2025. Codeium. https://codeium.com.

[10] Sourcegraph Cody. 2025. Cody supports additional context through Anthropic's Model Context Protocol. https://sourcegraph.com/blog/cody-supports-anthropic-model-context-protocol.

[11] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The Power of Noise: Redefining Retrieval for RAG Systems. *CoRR* abs/2401.14887 (2024). https://doi.org/10.48550/ARXIV.2401.14887 arXiv:2401.14887

[12] Cursor. 2025. Learn how to add and use custom MCP tools within Cursor. https://docs.cursor.com/context/model-context-protocol.

[13] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2024. AI Agents Under Threat: A Survey of Key Security Challenges and Future Pathways. *CoRR* abs/2406.02630 (2024). https://doi.org/10.48550/ARXIV.2406.02630 arXiv:2406.02630

[14] Windsurf Editor. 2025. Windsurf Editor. https://windsurf.com.

[15] Antanavicius et al. 2025. PulseMCP. https://www.pulsemcp.com.

[16] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, Ricardo Baeza-Yates and Francesco Bonchi (Eds.). ACM, 6491–6501. https://doi.org/10.1145/3637528.3671470

[17] Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K. Gupta, Taylor Berg-Kirkpatrick, and Earlence Fernandes. 2024. Imprompter: Tricking LLM Agents into Improper Tool Use. *CoRR* abs/2410.14923 (2024). https://doi.org/10.48550/ARXIV.2410.14923 arXiv:2410.14923

[18] Yuyou Gan, Yong Yang, Zhe Ma, Ping He, Rui Zeng, Yiming Wang, Qingming Li, Chunyi Zhou, Songze Li, Ting Wang, Yunjun Gao, Yingcai Wu, and Shouling Ji. 2024. Navigating the Risks: A Survey of Security, Privacy, and Ethics Threats in LLM-Based Agents. *CoRR* abs/2411.09523 (2024). https://doi.org/10.48550/ARXIV.2411.09523 arXiv:2411.09523

[19] gching. 2025. Toolbase. https://gettoolbase.ai.

[20] glama.ai. 2025. Glama MCP Servers. https://glama.ai/mcp/servers.

[21] Goose. 2025. Goose. https://goose.ai.

[22] Nadeeshaan Gunasinghe and Nipuna Marcus. 2021. *Language Server Protocol and Implementation.* Springer.

[23] Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, Creation, Usage: Benchmarking LLMs for Comprehensive Tool Utilization in Real-World Complex Scenarios. *CoRR* abs/2401.17167 (2024). https://doi.org/10.48550/ARXIV.2401.17167 arXiv:2401.17167

[24] JetBrains. 2025. JetBrains MCP Server. https://plugins.jetbrains.com/plugin/26071-mcp-server.
[25] Sotiropoulos John, Rosario Ron F Del, Kokuykin Evgeniy, Oakley Helen, Habler Idan, Underkoffler Kayla, Huang Ken, Steffensen Peter, Aralimatti Rakshith, Bitton Ron, et al. 2025. *OWASP Top 10 for LLM Apps & Gen AI Agentic Security Initiative.* Ph. D. Dissertation. OWASP.
[26] LangChain. 2022. LangChain: Framework for developing applications powered by language models. https://github.com/langchain-ai/langchain.
[27] Xinzhe Li. 2025. A Review of Prominent Paradigms for LLM-Based Agents: Tool Use, Planning (Including RAG), and Feedback Learning. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, 9760–9779. https://aclanthology.org/2025.coling-main.652/
[28] LibreChat. 2025. LibreChat. https://librechat.ai.
[29] Jerry Liu. 2022. LlamaIndex: A data framework for LLM applications. https://github.com/run-llama/llama_index.
[30] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2024. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. *CoRR* abs/2408.04682 (2024). https://doi.org/10.48550/ARXIV.2408.04682 arXiv:2408.04682
[31] Baidu Maps. 2025. Baidu Maps MCP Servers. https://lbs.baidu.com/faq/api?title=mcpserver/base.
[32] Emacs MCP. 2025. Emacs MCP. https://github.com/lizqwerscott/mcp.el.
[33] Tripo3D MCP. 2025. Tripo3D MCP. https://blender-mcp.com/.
[34] mcp dockmater. 2025. Dockmaster. https://mcp-dockmaster.com.
[35] mcp.so. 2025. MCP.so. https://mcp.so/.
[36] Ivan Milev, Mislav Balunović, Maximilian Baader, and Martin Vechev. 2025. ToolFuzz–Automated Agent Tool Testing. *arXiv preprint arXiv:2503.04479* (2025).
[37] OpenAI. 2023. ChatGPT plugins. https://openai.com/index/chatgpt-plugins/.
[38] OpenAI. 2023. Funcation Calling. https://platform.openai.com/docs/guides/function-calling?api-mode=responses.
[39] OpenAI. 2025. OpenAI Agents SDK - Model context protocol (MCP). https://openai.github.io/openai-agents-python/mcp/.
[40] OpenSumi. 2025. OpenSumi. https://github.com/opensumi/core.
[41] Model Context Protocol. 2024. GitHub MCP Server. https://github.com/modelcontextprotocol/servers/tree/main/src/github.
[42] Model Context Protocol. 2024. Slack MCP Server. https://github.com/modelcontextprotocol/servers/tree/main/src/slack.
[43] Replit. 2025. Replit. https://replit.com.
[44] Zhuocheng Shen. 2024. LLM With Tools: A Survey. *CoRR* abs/2409.18807 (2024). https://doi.org/10.48550/ARXIV.2409.18807 arXiv:2409.18807
[45] Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Zhumin Chen, Suzan Verberne, and Zhaochun Ren. 2024. Chain of Tools: Large Language Model is an Automatic Multi-tool Learner. *CoRR* abs/2405.16533 (2024). https://doi.org/10.48550/ARXIV.2405.16533 arXiv:2405.16533
[46] Zhengliang Shi, Shen Gao, Lingyong Yan, Yue Feng, Xiuyi Chen, Zhumin Chen, Dawei Yin, Suzan Verberne, and Zhaochun Ren. 2025. Tool Learning in the Wild: Empowering Language Models as Automatic Tool Agents. In *THE WEB CONFERENCE 2025*. https://openreview.net/forum?id=T4wMdeFEjX
[47] Block (Square). 2025. Block (Square). https://glama.ai/mcp/servers/atblock/square-mcp/tools/team.
[48] Stripe. 2025. Stripe. https://stripe.com.
[49] Microsoft Copilot Studio. 2025. Introducing Model Context Protocol (MCP) in Copilot Studio: Simplified Integration with AI Apps and Agents. https://www.microsoft.com/en-us/microsoft-copilot/blog/copilot-studio/introducing-model-context-protocol-mcp-in-copilot-studio-simplified-integration-with-ai-apps-and-agents/.
[50] Tencent. 2024. Tencent plugin shop. https://yuanqi.tencent.com/plugin-shop.
[51] Apify MCP Tester. 2025. Apify MCP Tester. https://apify.com/jiri.spilka/tester-mcp-client.
[52] TheiaAI/TheiaIDE. 2025. TheiaAI/TheiaIDE. https://theia-ide.org/docs/user_ai/.
[53] Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. 2025. LLM Agents Making Agent Tools. *CoRR* abs/2502.11705 (2025). https://doi.org/10.48550/ARXIV.2502.11705 arXiv:2502.11705
[54] Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Y. Zou. 2024. AvaTaR: Optimizing LLM Agents for Tool Usage via Contrastive Reasoning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/

paper/2024/hash/2db8ce969b000fe0b3fb172490c33ce8-Abstract-Conference.html

[55] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. *CoRR* abs/2309.07864 (2023). https://doi.org/10.48550/ARXIV.2309.07864 arXiv:2309.07864

[56] Konstantin Yakovlev, Sergey I. Nikolenko, and Andrey Bout. 2024. Toolken+: Improving LLM Tool Usage with Reranking and a Reject Option. *CoRR* abs/2410.12004 (2024). https://doi.org/10.48550/ARXIV.2410.12004 arXiv:2410.12004

[57] Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. ToolSword: Unveiling Safety Issues of Large Language Models in Tool Learning Across Three Stages. *CoRR* abs/2402.10753 (2024). https://doi.org/10.48550/ARXIV.2402.10753 arXiv:2402.10753

[58] Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pang, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, et al. 2025. A Survey on Trustworthy LLM Agents: Threats and Countermeasures. *arXiv preprint arXiv:2503.09648* (2025).

[59] Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2024. EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction. *CoRR* abs/2401.06201 (2024). https://doi.org/10.48550/ARXIV.2401.06201 arXiv:2401.06201

[60] Zed. 2025. Zed - Model Context Protocol. https://zed.dev/docs/assistant/model-context-protocol.

[61] Mingming Zha, Jice Wang, Yuhong Nan, Xiaofeng Wang, Yuqing Zhang, and Zelin Yang. 2022. Hazard Integrated: Understanding Security Risks in App Extensions to Team Chat Systems. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/auto-draft-262/

[62] Yanjie Zhao, Xinyi Hou, Shenao Wang, and Haoyu Wang. 2024. LLM App Store Analysis: A Vision and Roadmap. *CoRR* abs/2404.12737 (2024). https://doi.org/10.48550/ARXIV.2404.12737 arXiv:2404.12737