

Article

Not peer-reviewed version

A Survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs)

[Aditi Singh](#)*, [Abul Ehtesham](#)*, Saket Kumar, [Tala Talaei Khoei](#)

Posted Date: 2 April 2025

doi: 10.20944/preprints202504.0245.v1

Keywords: Model Context Protocol (MCP); Large Language Models (LLMs); Agentic AI; Standard- ized LLM Integration; Contextual Integration; Tool Utilization; Data Exchange; Dynamic Tool Discovery; Workflow Orchestration; Sampling; Composability



Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Article

A Survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs)

Aditi Singh ^{1,†} , Abul Ehtesham ^{2,*,†} , Saket Kumar ^{3,†} and Tala Talaei Khoei ^{4,†}

¹ Cleveland State University, OH; a.singh22@csuohio.edu

² Kent State University, Kent, OH; aehtesha@kent.edu

³ Northeastern University, Boston, MA; kumar.sak@northeastern.edu

⁴ Khoury College of Computer Sciences, Roux Institute Northeastern University, Portland, MN; t.talaeikhoei@northeastern.edu

* Correspondence: a.singh22@csuohio.edu

† These authors contributed equally to this work.

Abstract: The Model Context Protocol (MCP), recently introduced by Anthropic, proposes a standardized framework for integrating large language models (LLMs) with dynamic external systems. This survey reviews the foundational architecture of MCP, including its client-server model, standardized messaging protocols, dynamic tool discovery, and security mechanisms, against the backdrop of current, fragmented API solutions. Although the protocol promises enhanced interoperability and scalability for Agentic AI systems, data supporting its long-term performance remains limited. MCP's design is critically evaluated, potential applications in domains such as finance, healthcare, and customer service are discussed, and the key challenges are outlined. This work aims to inform researchers and practitioners of MCP's potential benefits and its present limitations in the evolving AI integration landscape. The GitHub link for this survey is: Model-Context-Protocol-Survey (<https://github.com/asinghcsu/model-context-protocol-survey>).

Keywords: Model Context Protocol (MCP); Large Language Models (LLMs); agentic AI, standardized llm integration; contextual integration; tool utilization; data exchange; dynamic tool discovery; workflow orchestration; sampling; composability

1. Introduction

The emergence of Large Language Models (LLMs) [1] has fundamentally reshaped artificial intelligence, demonstrating extraordinary capabilities in natural language understanding and generation. Despite these successes, LLMs remain inherently constrained by their reliance on static, pre-existing training datasets, which limits their applicability to dynamic, real-world scenarios. Consequently, research has increasingly focused on integrating LLMs with external data sources and practical tools to enhance their responsiveness, relevance, and operational effectiveness.

Current integration approaches predominantly rely on fragmented, custom-built Application Programming Interfaces (APIs) [2], each tailored to individual tools or datasets. This lack of standardization introduces substantial challenges, including increased integration complexity, difficulties in scaling across multiple sources, interoperability issues, and inconsistent security practices. Such hurdles significantly impede the development and deployment of scalable and adaptive Agentic AI systems (i.e., systems that operate autonomously and adapt dynamically without continuous human intervention).

To address these limitations, Anthropic introduced the Model Context Protocol (MCP) [3], described metaphorically as a USB-C port for AI [4]. MCP is an open protocol that standardizes how applications provide context to LLMs, much like USB-C standardizes device connectivity. By doing

so, MCP offers a unified approach to connect AI models to diverse external resources, replacing the fragmented, custom-built API integrations that currently hinder scalability and interoperability.

Furthermore, MCP facilitates the development of complex, agent-based workflows. It provides a growing ecosystem of pre-built integrations, the flexibility to switch between LLM providers, and adherence to best practices for securing data within infrastructure. At its core, MCP follows a client-server architecture in which MCP hosts (such as Claude Desktop, IDEs, or other AI tools) connect to lightweight MCP servers that securely access both local and remote data sources, while dedicated protocol clients maintain one-to-one connections with these servers to ensure secure, reliable, and standardized communication.

By defining structured communication methods, dynamic tool discovery mechanisms, secure data handling processes, and flexible context management, MCP significantly enhances LLMs' capabilities to autonomously execute complex tasks and adapt to evolving data contexts.

This survey paper aims to:

- Review the architectural components and core concepts of MCP.
- Critically assess how MCP compares with conventional API integrations, as in Table 1.
- Discuss potential applications across various industries.
- Identify challenges and propose directions for future work, acknowledging that empirical validation is still emerging.

While the protocol is promising, the current literature and available data are limited. Hence, this survey serves as an early-stage evaluation intended to guide subsequent, more comprehensive studies.

The paper is organized as follows: Section 2 reviews the background and motivations for MCP in LLM integration; Section 4 describes its architecture; Section 4 presents its core concepts (resources, prompts, tools, sampling, and roots); Section 5 explains building effective agents with MCP; Section 6 discusses its applications and impact; Section 7 outlines the challenges; and Section 8 concludes the paper.

2. Background

LLMs have exhibited substantial progress over recent years, demonstrating exceptional proficiency in a variety of complex tasks such as natural language generation, machine translation, question answering, and code [1]. However, despite their remarkable capabilities [5], these models encounter fundamental limitations, primarily due to their reliance on static, pre-collected training datasets. This reliance results in significant operational constraints, particularly in dynamic and real-world application scenarios, manifesting as:

- **Knowledge Staleness:** LLMs frequently lack up-to-date information, causing inaccuracies or irrelevant outputs when responding to queries related to recent developments.
- **Contextual Limitations:** Static models are often unable to adapt dynamically to changing contexts or effectively integrate external real-time information, limiting their contextual responsiveness.

Addressing these limitations necessitates seamless integration between LLMs and external resources such as real-time data sources, databases, and specialized tools. Traditionally, this integration has been accomplished via Application Programming Interfaces (APIs), each customized for specific use-cases and requiring substantial configuration effort. The prevalent usage of such individualized integration solutions presents multiple significant challenges:

- **Integration Complexity:** Building diverse API integrations requires repeated effort, extending development cycles.
- **Scalability Issues:** Adding new data sources or tools is cumbersome, limiting scalability and adaptability [6].
- **Interoperability Barriers:** The absence of standardized protocols hinders reuse across AI models, leading to redundancy.

- **Security Risks:** Custom API integrations often lack consistent security measures, increasing the risk of data breaches [7].

The Table 1 below summarizes these challenges by comparing traditional API integrations with the unified approach offered by MCP.

In response to these challenges [8], MCP has emerged as a standardized open protocol specifically designed to facilitate the seamless integration of AI applications with external systems. Drawing inspiration from established standardization successes such as web APIs and the Language Server Protocol (LSP) [9], MCP establishes a unified framework that significantly simplifies interactions between AI applications and external resources.

As illustrated in Figure 2, APIs historically standardized interactions between web applications and backend services like servers, databases, and software services. Similarly, LSP standardized interactions between Integrated Development Environments (IDEs) and programming language-specific tools, thereby improving code navigation, analysis, and intelligence. MCP extends this lineage of standardization explicitly into the AI domain, focusing on three core interfaces:

- **Prompts:** [10] Standardizing the provision and formatting of input context to AI models.
- **Tools:** Establishing consistent methodologies for dynamic tool [11] discovery, integration, and usage by AI agents.
- **Resources:** Defining standardized access to and utilization of external data and auxiliary resources for contextual enrichment.

The central idea behind MCP is the acknowledgement that a model's quality depends entirely upon the context it is provided. Historically, context management relied heavily on manual or fragmented solutions such as copying and pasting context data into AI interfaces. MCP eliminates these inefficient methods by enabling automated, structured, and secure integration of context, significantly enhancing AI responsiveness, personalization, and effectiveness. Figure 1 presents an overview of the MCP, illustrating its core components—such as the client-server architecture, standardized message formats, dynamic tool discovery, and context streaming mechanisms—that enable seamless and secure model interaction with external systems.

Consequently, MCP not only addresses the practical limitations faced by current LLM applications but also provides a robust foundation for the development of advanced Agentic AI systems capable of autonomous perception, reasoning, decision-making, and real-world interaction. Its adoption promotes scalability, interoperability, and security, positioning MCP as a critical component in the future development trajectory of intelligent AI systems.

Table 1. Comparison between Traditional APIs and Model Context Protocol (MCP).

Feature	API	MCP
Primary Focus	Application-to-application communication, specific function execution.	LLM-to-external resource communication, contextual data delivery, dynamic tool use, and standardized resource access.
Integration Approach	Fragmented, custom-built integrations.	Standardized, unified protocol for diverse resources and tools.
Context Handling	Limited, requires explicit context passing.	Built-in mechanisms for contextual data delivery, metadata, and prompt-based context communication.
Tool/Data Discovery	Manual endpoint discovery, requires prior knowledge.	Dynamic discovery through server queries, metadata, and tool introspection.
Security	Varies widely, depending on API design.	Standardized security mechanisms, including authentication, authorization, and data encryption.
LLM Specificity	General-purpose, used across various applications.	Designed specifically for LLM integration, Agentic capabilities, and resource/tool management.
Tool Usage	Requires custom-built coding for tool access.	Standardized tool usage via tool metadata, dynamic invocation, and prompt-based interaction.
Resource Access	Limited, requires specific API endpoints.	Standardized resource access through addressable entities and defined protocols.
Prompt Interaction	Typically simple text-based requests.	Structured prompts for detailed data and function requests, supporting complex interactions.
Transports	Varies, often HTTP/HTTPS.	Standardized transports for reliable communication, supporting diverse environments.
Sampling	Implementation varies, often manual.	Standardized sampling mechanisms for efficient data retrieval.

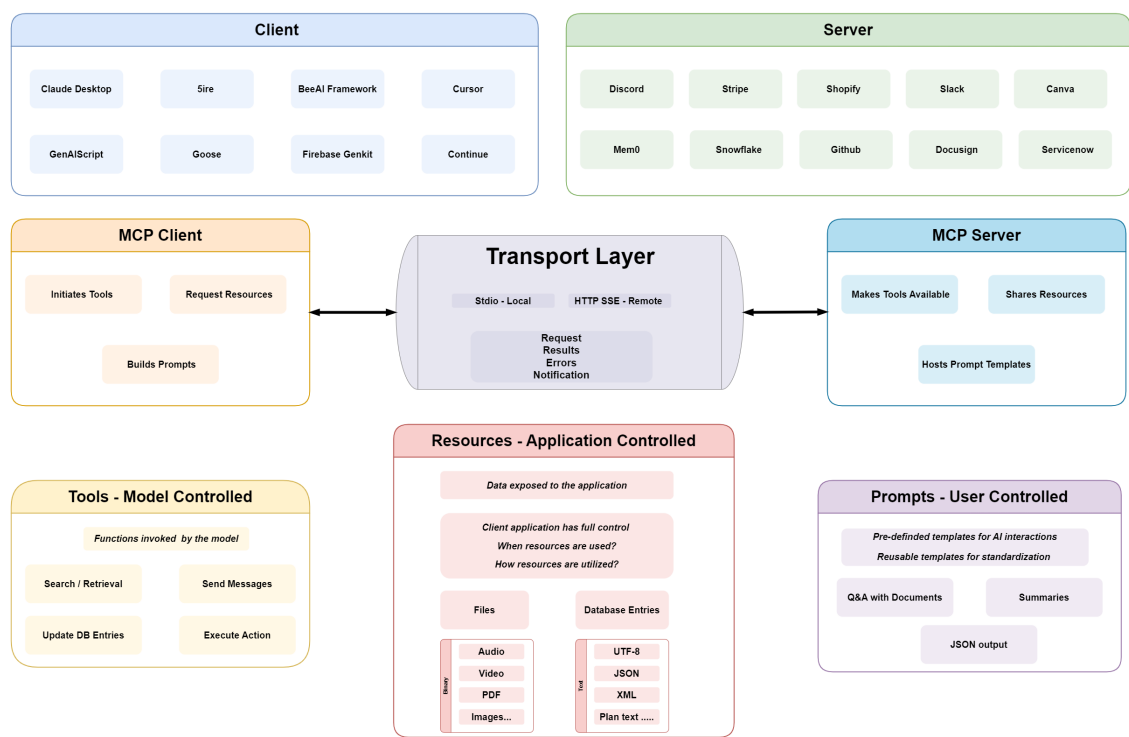


Figure 1. Overview of Model Context Protocol (MCP).

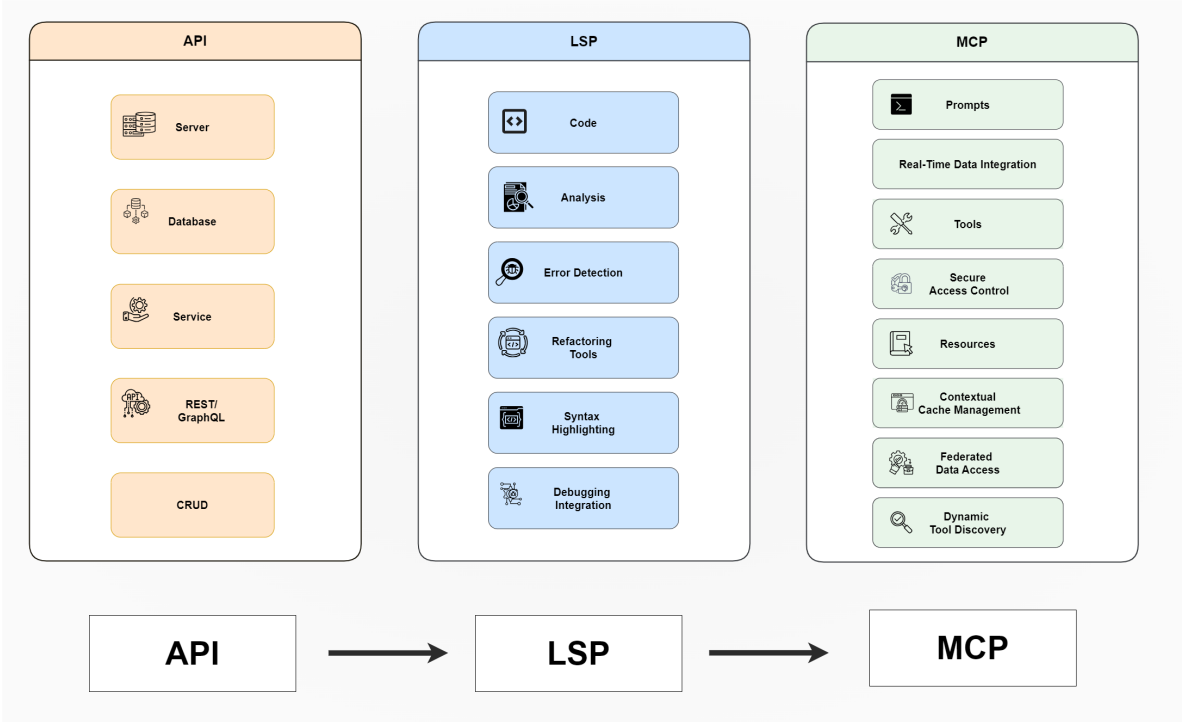


Figure 2. Comparative illustration showing how MCP standardizes AI application interactions with external systems, inspired by the success of APIs and LSP in their respective domains.

3. Architecture of MCP

This section describes the fundamental architectural components and lifecycle of MCP, highlighting its structured approach to client-server interactions, error handling, and security considerations.

3.1. Client-Server Architecture

MCP is designed around a structured client-server [12] model that facilitates efficient interaction between LLMs, external tools, and supporting resources, as illustrated in Figure 3. Within this architecture:

- **Hosts** refer to LLM applications, such as IDEs or platforms like Claude Desktop, that initiate communication.
- **Clients** establish dedicated one-to-one connections with servers within these host applications.
- **Servers** provide essential contextual data, tools, and prompts to enhance the client's functionality.

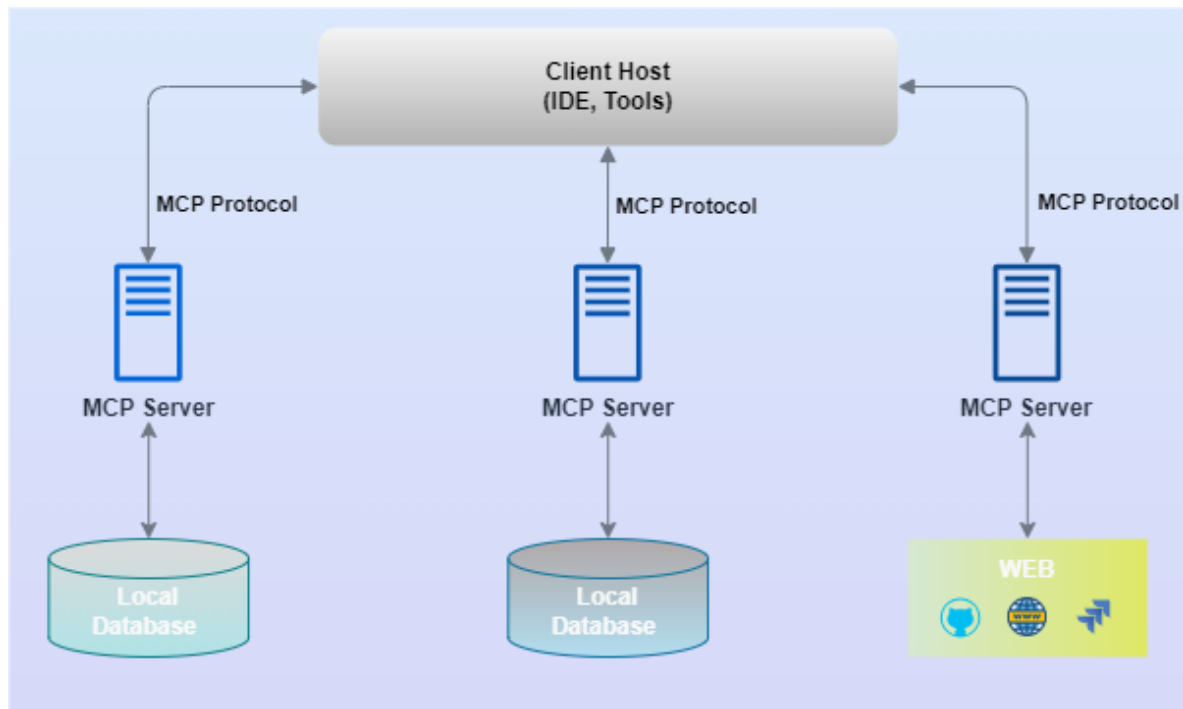


Figure 3. MCP's Structured Client-Server Architecture.

This design promotes modularity, scalability, and seamless interoperability, ensuring AI systems can efficiently integrate and utilize external resources while maintaining security and efficiency.

3.2. Protocol Layer

The protocol layer [12] governs structured communication by managing message framing, request-response handling, and notifications. MCP enforces a standardized message format, ensuring consistency across diverse implementations. It supports asynchronous communication, enabling non-blocking operations for efficient data exchange. The protocol also incorporates error handling mechanisms, ensuring robustness by validating responses and managing failures gracefully. This approach enhances interoperability, simplifies integration, and reduces overall system complexity.

3.3. Transport Layer

The transport layer [12] underpins message delivery between MCP components. Currently, MCP supports:

- **Standard Input/Output (Stdio):** Ideal for local and command-line based interactions, efficient in same-machine scenarios.
- **HTTP with Server-Sent Events (SSE):** Suitable for remote interactions and streaming scenarios, leveraging widely-adopted web standards.

Selection of appropriate transport mechanisms depends on application requirements such as locality, scalability, security constraints, and network configurations.

3.4. Message Types

MCP employs four primary message types:

- **Requests:** Initiated by a client expecting a response.
- **Results:** Successful responses to requests.
- **Errors:** Responses signaling failures in request processing.
- **Notifications:** One-way informational messages that do not require responses.

4. Fundamental MCP Concepts

MCP introduces several fundamental concepts designed to facilitate structured interactions between LLMs and external resources or tools. These concepts include resources, prompts, tools, sampling, and roots. Understanding these components is essential for effectively utilizing MCP in diverse application scenarios.

4.1. Resources

Resources [13] in MCP refer to structured data or content that servers expose to clients, which can then be utilized by LLMs to gain contextual insights during interactions. Resources can include a variety of data types such as textual documents, database entries, file contents, images, system logs, and real-time data streams. Each resource is uniquely identifiable using Uniform Resource Identifiers (URIs), conforming to a standardized schema to ensure consistency and ease of discovery. MCP resources are categorized broadly into two types:

- **Text Resources:** These resources consist of UTF-8 encoded textual data, making them ideal for sharing source code, configuration files, JSON or XML data, and plain text documents. Textual resources facilitate straightforward integration with LLM workflows for tasks like summarization, data extraction, and code analysis.
- **Binary Resources:** Binary resources comprise data encoded in base64 format, suitable for multimedia content such as images, audio, video, PDF documents, and other non-textual formats. Binary resources extend the capability of LLMs to interact with richer, multimedia-based contexts, supporting advanced tasks such as image recognition, document analysis, or multimedia summarization.

Resources are discoverable via two primary mechanisms:

1. **Direct Resource Listing:** Servers explicitly list available resources through the `resources/list` endpoint, providing metadata including the resource URI, descriptive name, optional detailed description, and MIME type.
2. **Resource Templates:** Servers can define dynamic resources via URI templates following RFC 6570 standards. Templates enable clients to construct specific resource URIs dynamically based on contextual requirements or parameters.

Clients interact with resources through structured requests, specifically via the `resources/read` endpoint. Responses include detailed metadata, MIME type, and resource content (either textual or binary). Additionally, MCP supports real-time resource updates using subscription-based notifications, enabling clients to remain synchronized with the latest resource state changes.

Effective resource management requires clear naming, robust error handling, thorough URI validation, and strong security measures (access control, encryption, and input sanitization) to mitigate vulnerabilities.

4.2. Prompts

Prompts [14] within MCP are structured templates provided by servers to standardize and streamline interactions between users, clients, and LLMs. These templates enable consistent, reusable workflows and interactions, enhancing both user productivity and model efficiency by clearly defining how interactions should be presented and processed.

MCP prompts serve several important functions:

- **Standardization of Interactions:** Prompts define structured formats that can include dynamic arguments, allowing consistent interactions regardless of the specific application context or LLM used.
- **Dynamic Context Integration:** Prompts can integrate content from external resources dynamically, enriching the interaction context available to LLMs, which facilitates more informed and relevant model responses.
- **Workflow Automation and Composability:** Prompts can encapsulate multi-step interaction workflows, enabling automated or semi-automated processes that guide users and models through complex tasks systematically.

Each prompt is characterized by the following structure:

- **Name and Description:** Clearly identifies the prompt, providing descriptive context for easy discovery and selection.
- **Arguments:** Optionally includes parameters defined by schema, specifying required or optional inputs from users or automated processes.

Clients discover available prompts through a dedicated endpoint (`prompts/list`), retrieving metadata including prompt names, descriptions, and defined argument schemas. Prompt usage occurs through requests made to the `prompts/get` endpoint, allowing the retrieval of fully structured prompt content ready for immediate use or customization.

Advanced implementations of prompts support features such as embedded resource context, enabling the combination of external resource data directly into prompt-driven interactions, and multi-step workflows that facilitate conversational and iterative interaction patterns between users and LLMs.

Best practices in prompt management include clear naming conventions, descriptive metadata, validation of inputs, handling errors gracefully, and careful management of security aspects such as input sanitization and controlled data exposure.

4.3. Tools

Tools [15] in MCP represent executable capabilities exposed by servers, empowering LLMs to perform actions, interact with external systems, and execute dynamic operations. Distinct from passive resources or static prompts, tools are designed for active invocation by models typically with human approval thereby significantly expanding the Agentic functionalities of LLM-driven systems. Tools Figure 4:

- **Dynamic Action Invocation:** Tools allow models to trigger operations dynamically, performing tasks such as computations, system commands, or API interactions without manual preconfiguration at runtime.
- **Structured Tool Definition:** Each tool is explicitly defined through JSON schema specifying inputs, ensuring clarity in interactions, precise validation, and secure invocation by clearly outlining required parameters and expected data formats.
- **Discoverability and Integration:** Clients discover tools through the `tools/list` endpoint, retrieving descriptive metadata and schemas. Invocation occurs via the structured `tools/call` mechanism, which facilitates clear, controlled tool execution and output handling.

MCP tool schemas typically define:

- **Name and Description:** Clearly identifying each tool and detailing its functionality.
- **Input Schema:** Precisely describing required inputs, parameter types, and validation constraints to prevent erroneous or unsafe executions.
- **Structured Results and Error Reporting:** Tools explicitly communicate execution outcomes, including successful results or structured errors, thereby enabling model-driven error handling or corrective actions.

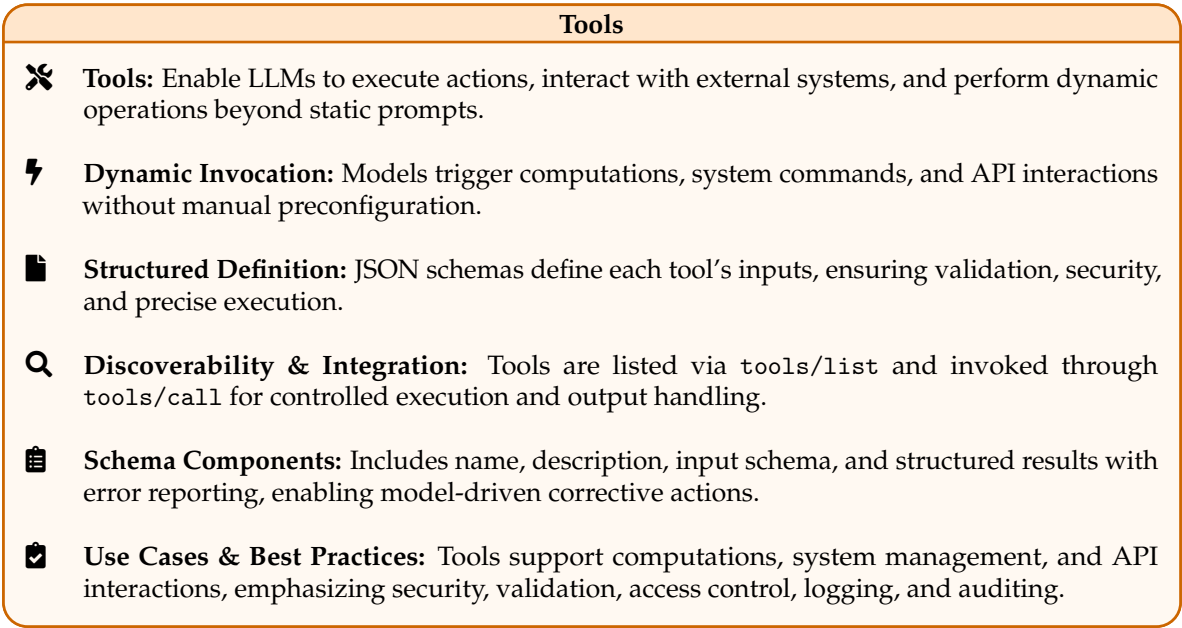


Figure 4. Overview of MCP Tools: Dynamic Invocation, Structured Definition, Discoverability, and Integration

Common use cases for tools range from simple local computations and file manipulations to sophisticated operations such as system management, database queries, and API interactions. Best practices emphasize clear definitions, comprehensive validation and error handling, security through rigorous input sanitization, proper access control, and robust logging and auditing to ensure accountability and transparency during tool execution.

4.4. Sampling

Sampling [16] within MCP enables servers to initiate inference requests directly to LLMs through clients, allowing dynamic retrieval of model-generated completions. This capability is crucial for developing sophisticated, interactive, and context-aware Agentic workflows that integrate human oversight and adaptive model interactions.

The MCP sampling process encompasses the following structured flow as illustrated in Figure 5:

1. **Server Initiates Request:** The server initiates a sampling request through the `sampling/create` Message endpoint, providing structured input, including the message context, optional model preferences (e.g., desired cost, latency, or capabilities), and sampling parameters such as temperature, token limits, and stop sequences.
2. **Client Review and Approval:** Before invoking the LLM, clients typically allow users to review, adjust, or approve the proposed message, thereby maintaining human-in-the-loop oversight and security.
3. **Model Execution and Completion Generation:** Upon user approval, the client interacts with the selected LLM to generate a completion, considering the provided sampling parameters to control response variability and specificity.
4. **Return and Handling of Completions:** The generated completion is returned to the initiating server through a structured response, clearly identifying the model used, the reason for completion termination, and the resultant output (textual or multimodal).

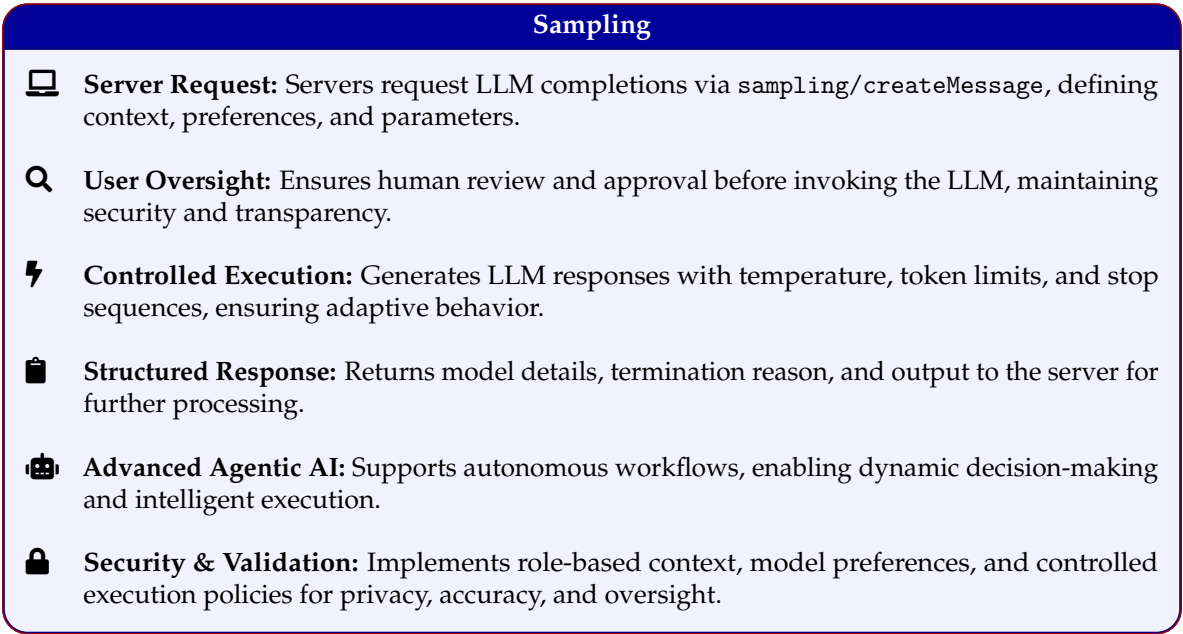


Figure 5. Overview of Sampling

Sampling requests use clearly defined JSON-based structures to ensure interoperability and transparency, specifying:

- **Contextual Messages:** Input messages structured with roles (e.g., user or assistant) and content type (text or images), guiding model responses clearly and effectively.
- **Model Preferences:** Providing hints or priorities for cost-efficiency, response speed, and model intelligence, assisting clients in optimal model selection and use.
- **Context Inclusion Policies:** Defining the scope of additional context to be included in the sampling request, such as data from the current or connected servers, enhancing the relevance and accuracy of model completions.

This mechanism enables adaptive interactions while ensuring security through human approval, input validation, and controlled execution.

4.5. Roots

Roots [17] define logical boundaries indicating the scope or context within which servers are expected to operate.They serve as structured guides provided by clients to inform servers about relevant resources or operational contexts, thus streamlining interactions and clarifying operational scope as illustrated in Figure 6.

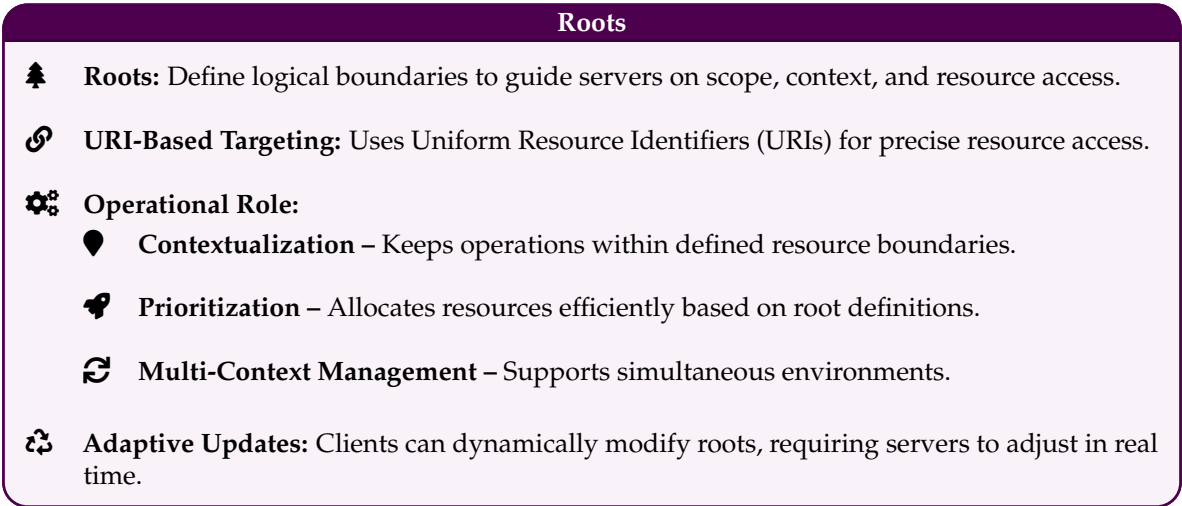


Figure 6. Roots and their Role in MCP Operations

Roots are primarily represented by Uniform Resource Identifiers (URIs), enabling precise resource targeting. While commonly used to denote filesystem paths or URLs, their application is highly flexible and can represent diverse resources, including:

- Local directories, for example: `file:///home/user/projects/app`
- API endpoints, for instance: `https://api.example.com/v1`
- Repository locations, configuration paths, or resource identifiers within distinct operational domains.

When establishing connections, clients specify supported roots, allowing servers to:

1. **Contextualize Operations:** Focus their data retrieval or tool invocation strictly within relevant resource boundaries.
2. **Prioritize Activities:** Allocate resources and execution priority based on the specified roots, improving operational efficiency.
3. **Manage Multi-Context Environments:** Seamlessly support scenarios involving multiple simultaneous operational contexts, such as multiple project directories or diverse remote services.

Although roots provide guidance rather than enforcement, servers typically implement best practices to respect the provided boundaries, optimizing their operations for clarity, security, and organizational alignment. Additionally, clients can dynamically update these roots during runtime, requiring servers to handle such updates gracefully and maintain operational robustness.

5. Building Effective Agents with MCP

MCP offers powerful features to develop sophisticated Agentic AI systems, primarily through its capabilities of *Sampling* and *Composability*.

5.1. Sampling: Federated Intelligence Requests

Sampling within MCP enables servers to request completions directly from the MCP client. This shifts control over LLM interactions entirely to the client side, enhancing security, privacy, and cost-efficiency.

5.2. Composability: Logical Separation and Agent Chaining

Composability in MCP refers to each node’s ability to function as both client and server as illustrated in Figures 7 and 8. This facilitates chaining, enabling complex, hierarchical, and specialized agent-based architectures.

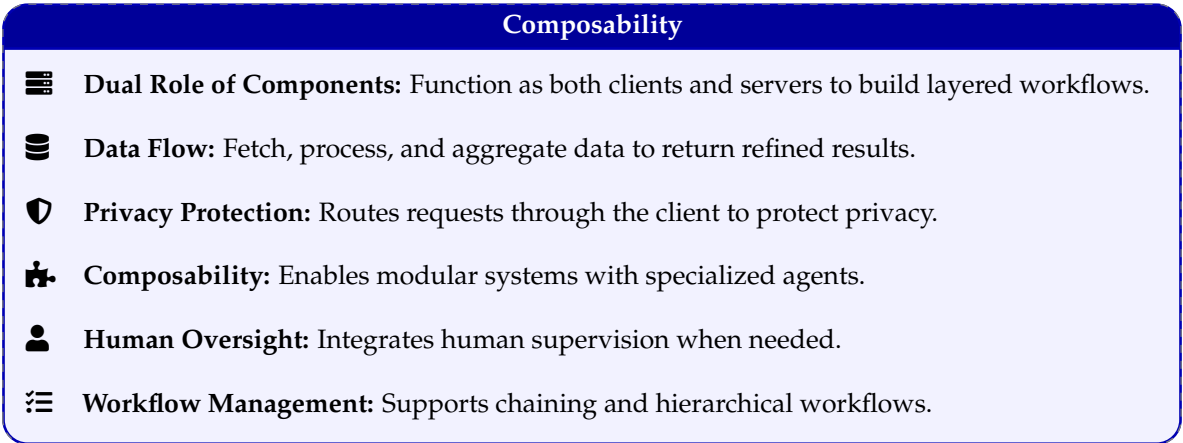


Figure 7. Overview of Composability.

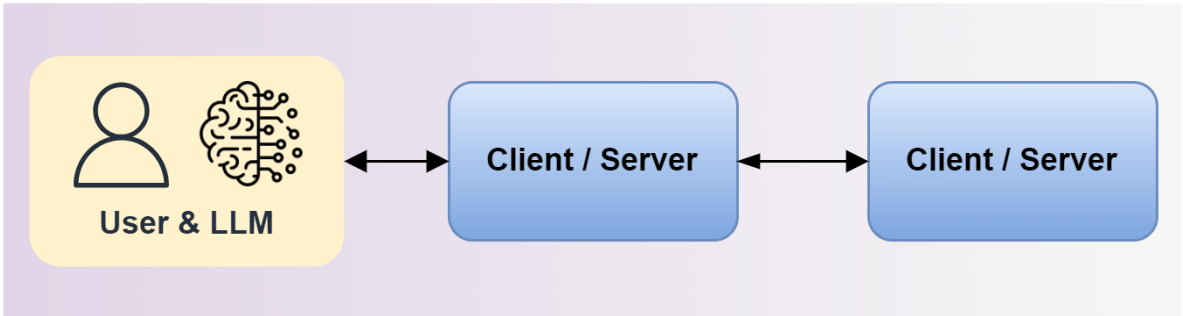


Figure 8. MCP Composability illustrating logical separation, allowing MCP client to simultaneously act as clients and servers.

- Significant advantages:
- Modular design of complex agent systems
 - Clear task specialization among agents
 - Scalability in multi-agent systems

5.3. Combined Sampling and Composability

Integration of sampling and composability provides a robust foundation for federated, hierarchical agent networks. Figures 8 and 9 illustrate these combined features.

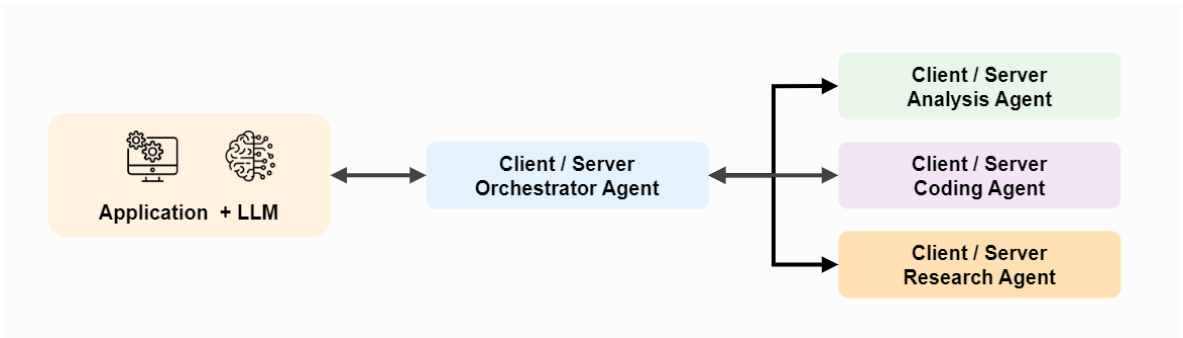


Figure 9. Combined MCP Features: A central orchestrator manages multiple specialized agents using sampling and composability.

- The combined approach allows:
- Centralized control and orchestration
 - Secure and flexible agent interactions
 - Enhanced capabilities for complex task management

6. Applications and Impact




























MCP revolutionizes LLM interactions by standardizing integration, enabling robust, agentic AI systems to perform complex tasks and access real-time information. Table 3 summarizes examples of MCP servers across various domains.

6.1. Real-World Applications

MCP enables a diverse range of real-world applications by standardizing the integration of LLMs with external resources. Specific examples include the MCP Use Case for a Customer Support Chatbot, which illustrates a comprehensive workflow—from constructing structured prompts to delivering a customer-friendly response (see Figure 10).

- **Finance:** Automated financial analysis, fraud detection, and personalized financial advice become more efficient through real-time data integration. For instance, the *Stripe* or *PayPal* MCP servers (see Table 2) allow LLMs to handle secure transactions and financial workflows, mitigating the fragmentation of traditional financial APIs.
- **Healthcare:** By integrating diverse healthcare data via specialized MCP servers, automated diagnosis, personalized treatments, and drug discovery become feasible. Although FHIR-based integrations are not yet available, they offer a promising path to standardize patient data access and bolster security.
- **Education:** Through personalized learning experiences, automated grading, and research assistance, MCP can enhance educational outcomes. Integrations like *Canvas* and *Blackboard* servers (see Table 2) streamline content delivery and user management, fostering adaptive learning environments.

Table 2. MCP Providers.

Category	Providers
Productivity & Project Management	 Airtable,  Google Tasks,  Monday,  ClickUp,  Trello
Design & Creative Tools	 Figma,  Canva
Marketing & Social Media	 LinkedIn,  X,  Mailchimp,  Ahrefs,  Reddit
Productivity	 Asana,  Jira,  Bolna
Entertainment & Media	 YouTube
CRM	 Salesforce,  HubSpot,  Pipedrive,  Apollo
Education & LMS	 Canvas,  Blackboard,  D2L Brightspace
Other	 Browserbase,  WeatherMap,  Google Maps,  HackerNews
Explore the Complete List of MCP Providers at:	awesome-mcp-servers (GitHub) mcp.composio.dev mcp.so github.com/modelcontextprotocol/servers

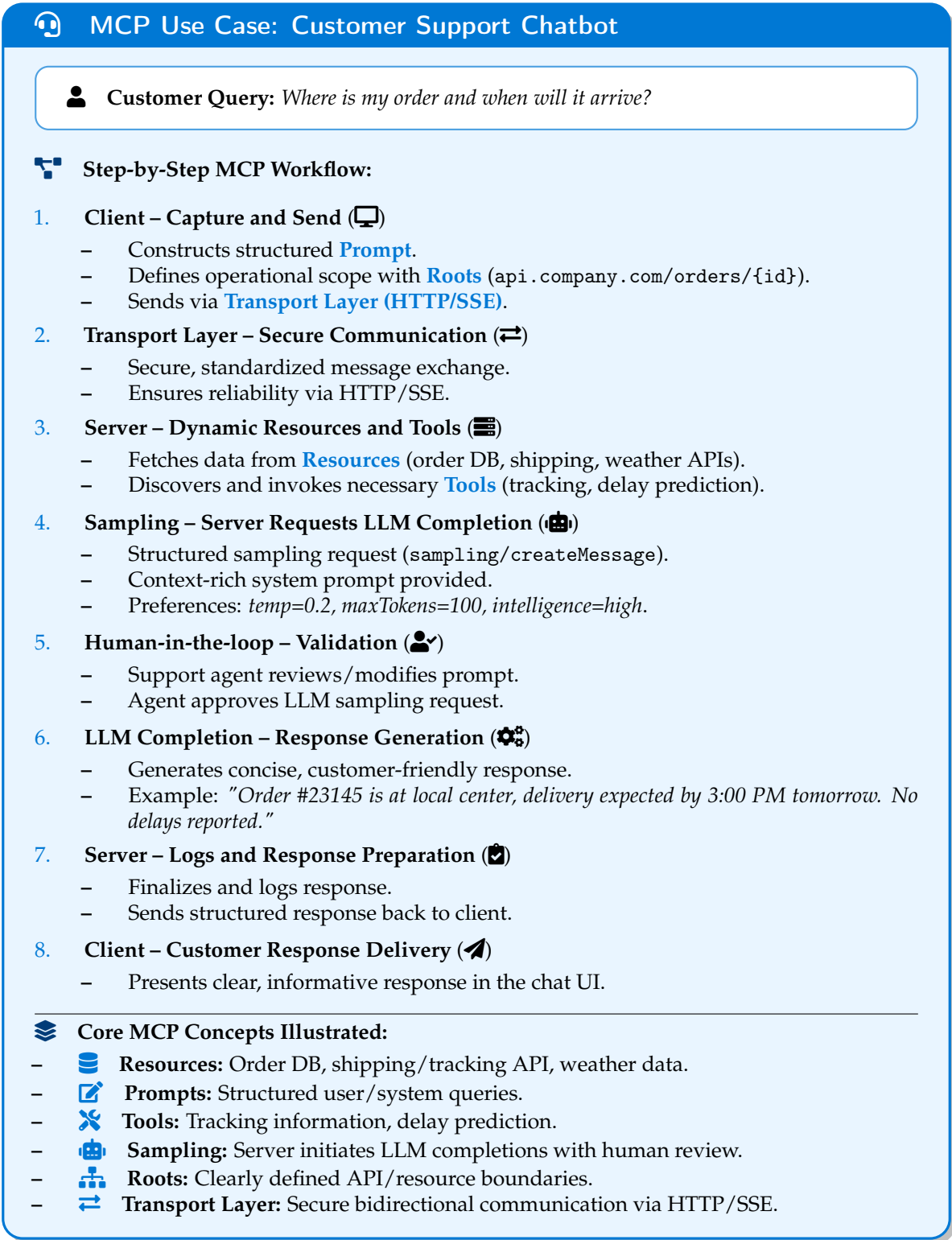


Table 3. MCP Impact on Agentic AI Development.

Aspect	Impact on Agentic AI
Contextual Awareness	Improves LLM understanding by incorporating contextual data for informed decisions.
Dynamic Tool Utilization	Allows LLMs to dynamically discover and use tools, enabling adaptation and complex task performance.
Scalability & Interoperability	Promotes standardized frameworks, enhancing robustness, versatility, scalability, and interoperability.
Secure Data Exchange	Ensures secure interaction between LLMs and external resources, enhancing trust and supporting sensitive applications.

- **Scientific Research:** By automating data analysis, literature review, and hypothesis generation, MCP supports faster, more thorough research processes. Tools such as *PostHog* or *Supabase* provide real-time analytics and manage large datasets efficiently, reducing time-to-insight in research workflows.
- **Customer Service:** MCP facilitates the development of intelligent chatbots and recommendation systems that deliver automated, personalized support and issue resolution. For example, *Salesforce* or *Zendesk* servers (see Table 2) enable LLMs to handle customer tickets, escalate complex queries, and securely manage user data, resulting in improved response times and customer satisfaction.
- **Marketing and Social Media:** Platforms like *Mailchimp*, *LinkedIn*, or *Reddit* can be integrated via MCP to automate campaign management, lead generation, and real-time audience engagement, thereby reducing manual overhead and ensuring up-to-date marketing efforts.
- **E-commerce:** Using *Shopify* or *WooCommerce* servers, LLMs can process orders, track inventory, and manage customer queries in a unified environment, making e-commerce operations more adaptive and efficient.
- **Collaboration and Project Management:** MCP servers for *Jira*, *Asana*, or *Trello* simplify workflow automation, task delegation, and real-time progress tracking, ensuring that teams can leverage AI to coordinate projects more effectively.

Overall, these applications demonstrate MCP’s capability to standardize communication across multiple platforms, reduce the complexity of custom-built API integrations, and empower LLMs to deliver truly Agentic AI experiences.

7. Challenges

While MCP offers a transformative approach to LLM integration, challenges remain in adoption, standardization, security, scalability, and ecosystem development.

- **Adoption and Standardization:** Establishing MCP as the industry standard requires active community involvement.
- **Security and Privacy:** Implementing comprehensive security mechanisms is crucial.
- **Scalability and Performance:** Ensuring low-latency communication as the system scales is essential.
- **Ecosystem Development:** Building a robust ecosystem of compatible tools and services is necessary.

8. Conclusions

The practical applications discussed throughout this survey underscore MCP's transformative potential across diverse domains, including finance, healthcare, education, scientific research, and customer service. However, the widespread adoption of MCP hinges on overcoming several challenges such as implementing robust security measures, scaling effectively to manage extensive data sources and interactions, and fostering a vibrant ecosystem through developer-friendly tools and comprehensive documentation.

In summary, while MCP paves the way for more autonomous, responsive, and intelligent AI capabilities, addressing the aforementioned challenges is critical for its long-term success. This survey highlights the current state of MCP and serves as a foundation for future evaluations and empirical studies.

References

1. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. A Survey of Large Language Models, 2025, [arXiv:cs.CL/2303.18223].
2. Perron, B.E.; Luan, H.; Qi, Z.; Victor, B.G.; Goyal, K. Demystifying Application Programming Interfaces (APIs): Unlocking the Power of Large Language Models and Other Web-based AI Services in Social Work Research, 2024, [arXiv:cs.SE/2410.20211].
3. Anthropic. Model Context Protocol. <https://modelcontextprotocol.io/introduction>, 2024. Accessed: 2025-03-18.
4. Model Context Protocol. <https://www.anthropic.com/news/model-context-protocol>, 2024. Accessed: 2025-03-18.
5. Kaddour, J.; Harris, J.; Mozes, M.; Bradley, H.; Raileanu, R.; McHardy, R. Challenges and Applications of Large Language Models, 2023, [arXiv:cs.CL/2307.10169].
6. Macvean, A. API Usability at Scale. <https://ppig.org/files/2016-PPIG-27th-Macvean.pdf>, 2016. Accessed: 2025-03-18.
7. Software Engineering Institute, C.M.U. API Vulnerabilities and Risks. <https://insights.sei.cmu.edu/documents/5908/api-vulnerabilities-and-risks-2024sr004-1.pdf>. Accessed: 2025-03-18.
8. Chen, X.; Gao, C.; Chen, C.; Zhang, G.; Liu, Y. An Empirical Study on Challenges for LLM Application Developers, 2025, [arXiv:cs.SE/2408.05002].
9. Kjær Rask, J.; Palludan Madsen, F.; Battle, N.; Daniel Macedo, H.; Gorm Larsen, P. The Specification Language Server Protocol: A Proposal for Standardised LSP Extensions. *Electronic Proceedings in Theoretical Computer Science* **2021**, 338, 3–18. <https://doi.org/10.4204/eptcs.338.3>.
10. White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; Schmidt, D.C. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT, 2023, [arXiv:cs.SE/2302.11382].
11. Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs, 2023, [arXiv:cs.AI/2307.16789].
12. Layer, P. Protocol Layer - Model Context Protocol. <https://modelcontextprotocol.io/docs/concepts/architecture#protocol-layer>, 2025. Accessed: 2025-03-18.
13. Resources. Resources - Model Context Protocol. <https://modelcontextprotocol.io/docs/concepts/resources>, 2025. Accessed: 2025-03-18.
14. Prompts. Prompts - Model Context Protocol. <https://modelcontextprotocol.io/docs/concepts/prompts>, 2025. Accessed: 2025-03-18.
15. Tools. Tools - Model Context Protocol. <https://modelcontextprotocol.io/docs/concepts/tools>, 2025. Accessed: 2025-03-18.
16. Sampling. Sampling - Model Context Protocol. <https://modelcontextprotocol.io/docs/concepts/sampling>, 2025. Accessed: 2025-03-18.
17. Roots. Roots - Model Context Protocol. <https://modelcontextprotocol.io/docs/concepts/roots>, 2025. Accessed: 2025-03-18.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.