A Survey on Model Context Protocol: Architecture, State-of-the-art, Challenges and Future Directions

Partha Pratim Ray, *Senior Member, IEEE*

***Abstract*—The rapid proliferation of large-language-model (LLM) services is constrained by three long-standing barriers: stateless integration interfaces, ad-hoc security controls, and the absence of a unified mechanism for injecting rich, multi-turn context. The Model Context Protocol (MCP) has been proposed to overcome these limitations through a session-oriented, JSON-RPC framework that allows LLMs to negotiate capabilities, invoke external tools, and retrieve contextual resources under fine-grained, OAuth 2.1–compliant access control. This article provides the first comprehensive treatment of MCP from a communications-systems perspective. We (i) dissect its layered host–client–server architecture, (ii) formalize its lifecycle and transport semantics, and (iii) benchmark official and community implementations spanning software-development, cloud-automation, healthcare, and cyber-security domains. Open challenges—including transport-layer latency,token-lifecycle overhead, cross-server privilege escalation, and persistent-context tampering—are analysed in depth, and mitigation strategies rooted in cryptographic hardening, edge caching, and capability-graph isolation are articulated. Finally, we outline a future research agenda that extends MCP toward multimodal payloads, Quick UDP Internet Connection (QUIC)-enabled streaming, and post-quantum credential exchange. By synthesizing protocol design with deployment evidence, the article positions MCP as a pivotal enabler of secure, adaptive, and scalable agentic workflows—marking a decisive forward step in the maturation of real-world LLM applications.**

***Index Terms*—Model context protocol, large-language models, context-aware communication, capability negotiation, secure tool invocation, agentic workflows**

I. Introduction

The remarkable advancements in artificial intelligence (AI)—and more specifically, LLMs—have ushered in a new paradigm of interactive, intelligent systems [1], [2], [3]. These models, built on transformer-based deep learning architectures, possess the ability to generate coherent and contextually rich responses, engage in multi-turn dialogue, and assist in a broad array of cognitive tasks [4], [5]. Yet, as their adoption expands across high-stakes domains such as healthcare, finance, legal analysis, and real-time decision-making, it becomes increasingly apparent that the infrastructure supporting their deployment must also evolve [6], [7]. A key bottleneck in this evolution lies in the inability of conventional communication protocols to handle the nuanced, context-rich, and stateful interactions required by modern LLM applications [8], [9].

Legacy systems—designed around stateless HTTP or fixed-schema remote procedure call (RPC)—are fundamentally limited in their ability to maintain conversational context, coordinate asynchronous operations, or dynamically negotiate capabilities between distributed agents. In contrast, LLM-based systems require more than mere data transmission; they require the orchestration of meaning across sessions, contextual adaptation based on user intent, and secure access to dynamic resources and tools [10], [11], [12]. The need for an interaction protocol that supports these requirements is both urgent and foundational. It is in response to this demand that the MCP has emerged—a context-exchange protocol that not only addresses the limitations of existing paradigms but also redefines the communication blueprint for LLM-centered systems [13], [14].

MCP introduces a robust and extensible mechanism for structured, stateful communication between clients, hosts, and servers—each of which may expose tools, resources, and prompts designed to augment the inferential capabilities of LLMs [15], [16]. Unlike traditional request-response mechanisms, MCP facilitates bi-directional, session-aware exchanges with support for dynamic capability negotiation, contextual sampling, tool invocation, and prompt completion. These interactions are governed by a well-defined JSON-RPC protocol specification, offering deterministic behavior, fine-grained authorization, and adaptive session lifecycle management [17], [18]. Its layered architecture fosters modular design and encourages extensibility, making it suitable for both lightweight edge devices and enterprise-grade deployments.

The motivation for MCP stems from the increasing complexity and modularity of AI-powered applications [19]. With LLMs acting as reasoning engines, the systems around them must support the real-time aggregation and transformation of data from diverse, and often heterogeneous, sources [20]. This includes secure interaction with filesystems, web services, databases, development pipelines, and even physical sensors. To orchestrate such interactions, a protocol must not only support declarative capability exchange and structured context injection but must do so while maintaining strict security boundaries, minimal latency, and maximal interoperability [21]. MCP fills this gap by embedding intelligence into the protocol layer itself—allowing agents to negotiate, interpret, and act on contextual information with precision and flexibility [22].

This work presents a comprehensive and technically grounded review of the MCP. It is motivated by the growing momentum around MCP and its pivotal role in enabling next-generation AI systems that are adaptive, context-sensitive, and semantically aware. The core objectives of this article are as follows:

This article provides a rigorous survey of MCP, with the following primary objectives:

- To provide a detailed overview of the MCP specifica-

P. P. Ray is with Department of Computer Applications, Sikkim University, India. Email: ppray@cus.ac.in.

tion, highlighting its architecture, session model, tool invocation framework, and its divergence from traditional stateless APIs.
- To evaluate and contrast current MCP implementations—including official, community-developed, and reference servers—through real-world case studies, highlighting their architectural decisions, trade-offs, and deployment insights.
- To identify and analyze the multifaceted challenges in MCP adoption, including capability negotiation complexity, protocol versioning, transport latency, and the integration of advanced security practices.
- To outline the future trajectory of MCP, focusing on areas such as context prioritization, multi-modal integration, quantum-safe cryptography, and adaptive, self-configuring communication strategies.

This article contributes to the discourse on MCP by integrating perspectives from protocol design, distributed systems engineering, LLM architecture, and secure communication. Key contributions include:

- A structured review of MCP literature and development history, tracing its origin and evolution in the context of LLM system requirements.
- A comparative analysis of state-of-the-art MCP frameworks, enriched with performance benchmarks, deployment experiences, and use-case-driven insights.
- A granular exploration of critical challenges such as capability negotiation, context overloading, latency mitigation, and interoperability across heterogeneous platforms.
- A forward-looking roadmap that discusses architectural innovations, cryptographic enhancements, context optimization techniques, and standardization efforts necessary to scale MCP adoption.

The MCP embodies the foundational infrastructure necessary for the future of AI-as-a-service (AIaaS). As the AI ecosystem matures, LLMs will increasingly rely on contextual information not merely for input conditioning but for tool usage, decision chaining, and external function execution [23]. MCP offers a solution grounded in protocol rigor, extensibility, and interoperability—key ingredients for building robust, secure, and scalable AI systems. For organizations seeking to operationalize AI in critical workflows, MCP provides a viable path toward standardized, explainable, and composable LLM interfaces [24], [25].

In addition to enhancing the technical efficiency of LLM deployments, MCP fosters a new level of human-computer collaboration. By aligning machine responses with contextual expectations and user intent, it significantly improves the quality, consistency, and accountability of AI interactions [26]. From powering intelligent customer care agents and automated development assistants to orchestrating context-aware data pipelines, MCP's design opens up possibilities that transcend traditional communication paradigms. List of abbreviations and full form thereto in this paper is mentioned in Table I.

TABLE I
LIST OF ABBREVIATIONS USED IN THE MANUSCRIPT

Abbreviation	Full form
A2A	Agent-to-Agent
ACL	Access Control List
AI	Artificial Intelligence
AIaaS	Artificial Intelligence-as-a-Service
API	Application Programming Interface
CLI	Command-Line Interface
CI/CD	Continuous Integration / Continuous Delivery or Deployment
CRM	Customer Relationship Management
DB	Database
DPAPI	Data Protection API
DPDK	Data Plane Development Kit
eBPF	Extended Berkeley Packet Filter
EHR	Electronic Health Record
ETL	Extract, Transform, Load
FaaS	Function-as-a-Service
FPGA	Field-Programmable Gate Array
GIS	Geographic Information System
gRPC	gRPC Remote Procedure Call
GPU	Graphics Processing Unit
HIPAA	Health Insurance Portability and Accountability Act
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
JSON-RPC	JavaScript Object Notation Remote Procedure Call
JWT	JSON Web Token
KPI	Key Performance Indicator
LLM	Large Language Model
LMS	Learning Management System
LUKS	Linux Unified Key Setup
MCP	Model Context Protocol
MES	Manufacturing Execution System
NER	Named Entity Recognition
NIC	Network Interface Card
OAuth	Open Authorization
PAT	Personal Access Token
PKCE	Proof Key for Code Exchange
PKI	Public Key Infrastructure
QUIC	Quick UDP Internet Connections
RAG	Retrieval-Augmented Generation
ReAct	Reasoning and Acting
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software-as-a-Service
SCADA	Supervisory Control and Data Acquisition
SDN	Software-Defined Networking
SSE	Server-Sent Events
SIEM	Security Information and Event Management
SQL	Structured Query Language
SSO	Single Sign-On
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
UI	User Interface
URI	Uniform Resource Identifier
VM	Virtual Machine

The remainder of this paper is structured as follows. Section II elaborates existing literature on MCP. Section III provides an in-depth overview of MCP, discussing its concept, motivation, and the evolution that has led to its current form. In Section IV, we present a comprehensive insight of state-of-the-art MCP implementations. Section V delves into the challenges faced by MCP deployments. Section VI outlines promising future directions. Section VII concludes the article.

II. Related Works

The rapid expansion of LLM capabilities has catalyzed a paradigm shift in how intelligent systems interface with external tools, services, and heterogeneous data sources [27]. This shift has highlighted the inadequacy of traditional stateless communication protocols—such as Representational State Transfer (REST)ful APIs or simplistic RPC models—for supporting the dynamic, context-rich interactions demanded by modern AI systems. Against this backdrop, the MCP has emerged as a novel architectural and semantic framework aimed at structuring, negotiating, and managing real-time contextual exchanges between LLMs and the environments in which they operate [28], [29]. A growing body of literature has begun to explore the conceptual, architectural, and operational dimensions of MCP, as well as its broader implications for agent orchestration, tool invocation, and secure reasoning. This section synthesizes key contributions from prior research, organizing them thematically across domains including secure execution, formal reasoning, agentic workflows, human-AI orchestration, and multimodal deployments. By critically examining these works, we contextualize the evolution of MCP and identify foundational ideas, architectural gaps, and open research challenges that motivate the present survey.

Initial investigations [30] have shed light on MCP's potential to streamline complex workflows by enabling LLMs to interact dynamically with toolchains, APIs, and datasets. However, security remains a critical concern. Radosevich and Halloran [30] demonstrated how adversaries can exploit tool invocations through MCP to compromise system integrity, underscoring the need for proactive safety audits such as McpSafetyScanner.

Several authors have attempted broader architectural surveys. Singh et al. [31] provided an early conceptual framework for MCP, situating it within the context of fragmented API designs while discussing its relevance in finance, healthcare, and customer service. Hou et al. [32] extended this by evaluating MCP's lifecycle—creation, operation, and update—offering security-centric insights for each phase. Both contributions emphasize the necessity of governance mechanisms and performance evaluation strategies to support real-world scalability.

Szeider [33] introduced MCP-Solver, bridging LLMs with formal reasoning tools like MiniZinc and PySAT, demonstrating that MCP can enhance LLM capabilities by delegating symbolic computation to specialized solvers. In networking domains, CAIP by Jiang et al. [34] presented an iterative prompting mechanism to detect router misconfigurations with LLMs, pointing to the utility of context-driven prompting for precision in infrastructure management.

A notable innovation in networking comes from Dall'Agata [35], who proposed intent-driven interfaces using LLMs to convert high-level policies into software-defined network configurations. This architecture aims to close the gap between human intent and machine-executable code using MCP-aligned principles. In wireless networks, Zhang et al. [36] highlighted the efficacy of LLMs in intrusion detection, validating in-context learning as a powerful paradigm with high accuracy using minimal examples.

The broader evolution of LLM ecosystems and agent infrastructure has also attracted scholarly attention. Hou et al. [37] proposed a layered software-hardware synergy model for scalable LLM deployment, advocating for open, modular ecosystems—conceptually aligned with MCP's design goals. Meanwhile, Cao et al. [38] explored multi-level clue-guided prompting for entity recognition in the coal mining industry, using MCP-like structured interactions to bridge pretraining and task-specific learning.

The role of LLM agents has been discussed extensively in the literature. Namiot and Ilyushin [39] drew parallels between agents and web mashups or software robots, positioning them as orchestrators of LLM workflows. In industrial automation, Restrepo Torres [40] explored the feasibility of transforming natural language requirements into PLCopen XML artifacts using open-source LLMs. This work supports MCP's premise of integrating structured outputs from unstructured inputs.

In the context of 6G communications, Jiang et al. [41] proposed a multi-agent framework where LLMs participate in retrieval, planning, and refinement—mirroring MCP's tool chaining and context aggregation functionalities. Luo et al. [42] offered a taxonomy of LLM agents, categorizing them by architectural and collaborative traits and calling attention to the growing sophistication of agent design in AI ecosystems.

Wu [43] introduced Autono, a ReAct-based agent framework that dynamically generates actions and abandons ineffective paths using a probabilistic strategy. It exemplifies how agentic robustness can be enhanced when execution pathways are guided by prior context and tool compatibility, an idea foundational to MCP's adaptability. Lo et al. [44] applied a multi-agent system for AI-powered resume screening, integrating Retrieval-Augmented Generation (RAG) to match applicants against customized criteria—a relevant example of MCP-style orchestrated interactions in HR.

Zimmermann et al. [45] captured the diverse application potential of LLMs in chemistry and materials science through a hackathon event, with applications spanning data curation, design, and hypothesis testing. This variety exemplifies how protocols like MCP can serve as a backbone for rapid, domain-agnostic LLM deployment. Lastly, Meijer [46] emphasized the value of symbolic abstraction via indirection in tool-calling frameworks. His proposal lays a theoretical foundation for building reusable, interpretable interactions, resonating with MCP's goals of modularity and interoperability.

Taken together, these works illustrate the growing interest in context-driven, agent-integrated LLM workflows, while also identifying critical gaps in formalization, security, and cross-domain applicability. The present survey builds on these foundations by offering a unified analysis of MCP's architecture, comparative implementations, deployment challenges, and emerging research frontiers. Table II presents the comparison of the reviewed literature.

Lessons Learned

From the reviewed studies, several key insights emerge that are instrumental in shaping the future of MCP server development and deployment. First, while MCP introduces a standardized and modular approach to tool integration for LLMs, its real-world implementation is still in its infancy,

TABLE II
COMPARISON OF RELATED WORKS ON MCP AND LLM INTEGRATION

Reference	Survey	MCP Workflow	Key Contributions	Limitations
[30]	No	Tool chaining, agentic workflows	Introduced McpSafetyScanner for auditing MCP server vulnerabilities	Limited focus on broader security standards or defense strategies
[31]	Yes	Client-server architecture, tool discovery	Surveyed MCP's architectural design and discussed its applications in various industries	Lack of empirical validation or long-term deployment results
[32]	No	Lifecycle phases (creation, operation, update)	Explored adoption scenarios and security risks across the MCP lifecycle	High-level discussion; lacks technical depth in mitigation measures
[33]	No	Symbolic solver integration via MCP	Bridged LLMs with MiniZinc, PySAT, and Z3 using structured MCP interfacing	Limited to symbolic computation; not extensible to non-logical domains
[34]	No	Context-aware prompt chaining	Introduced CAIP for automated misconfiguration detection in routers	Only evaluated on synthetic and small-scale datasets
[35]	No	Intent-to-API conversion	Mapped high-level intents into SDN configurations using LLMs	Pipeline performance depends heavily on prompt quality
[36]	No	In-context learning for intrusion detection	Demonstrated GPT-4 performance on wireless attacks without fine-tuning	Application limited to intrusion scenarios; lacks tool diversity
[37]	No	Modular AI architecture vision	Proposed a decoupled architecture for LLM ecosystem interoperability	Conceptual proposal without implementation or real-world results
[38]	No	Multi-level prompt processing	Developed MCP-like prompt structure for domain-specific NER in mining	Tailored to a niche domain; generalizability not tested
[39]	No	Agent orchestration using LLMs	Positioned agents as MCP-compatible modular components for AI tasks	Lacks empirical evaluation or implementation details
[40]	No	Requirement to XML generation	Used open-source LLMs to generate PLCopen XML from text requirements	Evaluation limited to synthetic datasets; scalability unclear
[41]	No	Multi-agent pipeline for communication tasks	Proposed CommLLM with retrieval, planning, and evaluation modules	Focused on 6G; not generalized to other verticals
[42]	Yes	Agent methodology taxonomy	Provided a structured classification of LLM agents and their roles	Does not delve into specific protocol implementations like MCP
[43]	No	ReAct-based agent workflows	Introduced dynamic path generation and memory sharing in agents	Probabilistic abandonment lacks fine-grained control models
[44]	No	RAG-enhanced resume screening	Deployed LLM agents with context-aware scoring using external knowledge	Focused only on HR; lacks comparison with traditional screening tools
[45]	No	Domain-specific tool orchestration	Highlighted LLM potential across scientific research applications	Hackathon format; implementations not production-tested
[46]	No	Indirect function abstraction	Theoretical basis for symbolic tool interfacing and parameterization	Lacks integration with LLM workflows or agent systems

requiring deeper scrutiny around security, performance, and interoperability. Researchers have shown that unregulated tool invocation can introduce significant vulnerabilities, underscoring the urgent need for built-in safety auditing mechanisms and access control policies to govern AI-tool interactions responsibly. Moreover, the utility of MCP across domains—from secure enterprise workflows to automation in industrial systems—demonstrates its adaptability, but also highlights the challenge of creating domain-agnostic yet configurable servers. The integration of LLMs with symbolic systems and multi-agent architectures further emphasizes the value of combining statistical reasoning with deterministic control, a balance that MCP servers must increasingly support. Lastly, the studies reveal that effective prompt engineering, contextual grounding, and iterative validation remain core components in ensuring that MCP-enhanced systems are not only functional but also interpretable, scalable, and trustworthy. As MCP continues to evolve, future work must prioritize robust tooling standards, open-source contributions, and comprehensive benchmarks to accelerate its safe and impactful adoption.

Novelty of the Work

Despite the growing interest in the MCP as a standardized interface for integrating large language models with external tools and data sources, there remains a notable absence of a consolidated, scholarly review that captures both its architectural foundations and emerging application landscape. This survey represents the first comprehensive and technically rigorous study dedicated to MCP in the context of LLM integration. Unlike prior literature that has primarily focused on individual implementations or domain-specific applications, our work unifies fragmented insights into a cohesive framework—encompassing architectural analysis, implementation strategies, comparative evaluation of real-world deployments, and an in-depth exploration of unresolved challenges. Furthermore, we extend the discussion beyond theoretical capabilities by contextualizing MCP's role within evolving trends in agentic AI systems, tool-call standardization, dynamic capability negotiation, and secure multi-agent workflows. This survey also distinguishes itself by examining MCP's implications across multiple domains, not just in software engineering or IT, but also in domains like healthcare, cybersecurity, industrial automation, and scientific research. By doing so, we not only position MCP as a transformative protocol for AI ecosystems but also lay the groundwork for future innovation by highlighting gaps and proposing clear research directions. Our contribution fills a crucial void in the literature and serves as a foundational reference for both academic researchers and industry practitioners aiming to harness MCP in real-world, high-impact AI systems.

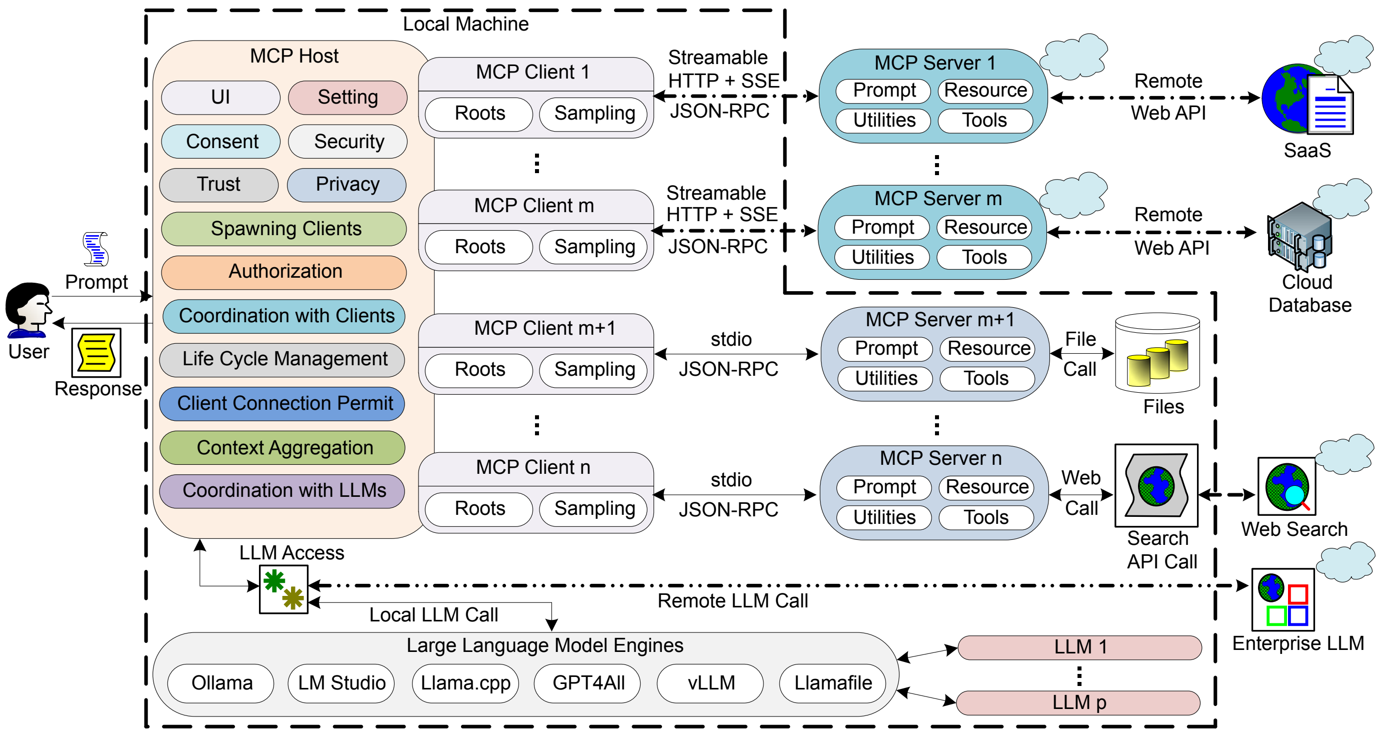

Fig. 1. MCP workflow.

III. Architectural Components of the Model Context Protocol

The evolution of AI integration protocols reflects the growing sophistication of AI systems—particularly LLMs [47], [48]. Early implementations depended on simple HTTP-based request-response schemes, which were sufficient for stateless and basic tasks but quickly proved inadequate as conversational AI systems grew more context-sensitive and interactive [49], [50]. Protocols like gRPC i.e. cross-platform with high performance RPC and REST emerged to address these limitations. gRPC enabled efficient, typed, and bi-directional communication, while REST simplified resource handling via standard HTTP verbs [51], [52]. However, both suffered in handling dynamic workflows, real-time interactions, and long-lived context—challenges increasingly central to LLM applications. REST lacked state management and standard schemas, while RPC frameworks introduced rigidity and maintenance burdens [53], [54]. With the rise of models like GPT-4 and Claude, the demand for context-rich, modular, and secure integration mechanisms intensified. Existing protocols couldn't meet requirements such as session persistence, complex capability negotiation, or service composability—highlighting the need for a purpose-built standard. The MCP emerged as a direct response. Built on JSON-RPC 2.0 [55], MCP is designed for extensibility, modularity, and security. Its architecture separates hosts, clients, and servers, enabling isolated concerns and incremental capability upgrades. Through capability negotiation, MCP facilitates backward-compatible extensions while maintaining strict security via OAuth 2.1 [56]. MCP thus fills a critical gap in the AI ecosystem: (i) it empowers developers to build advanced, (ii) context-aware applications with minimal friction, (iii) setting a foundation for scalable and (iv) secure AI innovation.

A. MCP Workflow

Figure 1 illustrates the comprehensive architectural workflow of the MCP, a standardized framework designed to seamlessly connect LLMs with contextual data, tools, and user interfaces across local and remote environments. The core of the system lies in the MCP Host, a trusted local process responsible for orchestrating the life cycle of multiple client instances. It manages critical concerns such as user interface controls, settings configuration, privacy enforcement, and trust policies. Moreover, it oversees security protocols, manages client permissions, and coordinates life cycle events, such as client spawning, shutdown, and resource cleanup. The host also plays a pivotal role in context aggregation, enabling coherent communication between multiple clients and language models, while enforcing strict boundaries to ensure that no server or client overreaches its designated access scope. Within this architecture, MCP Clients serve as isolated executors of specific sessions. Each client maintains a dedicated, point-to-point connection with a corresponding MCP Server through either Streamable HTTP[1] + Server-Sent Events (SSE)[2] which is usually used for asynchronous, event-driven communication or stdio-based JSON-RPC 2.0 which is normally used for lightweight, synchronous messaging. These clients advertise their capabilities—such as file system access (i.e. roots) and

[1]https://modelcontextprotocol.io/specification/2025-03-26/basic/transports#streamable-http

[2]https://en.wikipedia.org/wiki/Server-sent_events

language model querying (i.e. sampling)—during the initialization phase, and operate under a strict contract negotiated with both the host and server.

MCP Servers are modular, stateless or state-aware units that expose focused capabilities through well-defined primitives, including prompts (predefined input templates guiding model behavior), resources (structured or unstructured contextual data such as code files, logs, or documents), tools (e.g. executable functions or remote APIs), and utilities (supporting services like logging, completion, and notifications). These servers may run locally—interfacing directly with files and sensors—or connect to cloud endpoints such as Software-as-a-Service (SaaS) platforms, enterprise databases, and search APIs. Depending on the configuration, servers may support dynamic resource discovery, subscriptions to data changes, and asynchronous interactions, enriching the context available to the LLM without exposing the full conversation history or unrelated client data. One of the critical strengths of this architecture is its hybrid LLM invocation layer, which intelligently routes sampling requests either to local models via LLM engines or frameworks (e.g., Ollama[3], GPT4All[4], Llama.cpp[5], LM Studio[6], vLLM[7]) or to remote LLMs (e.g., via enterprise APIs or commercial endpoints) based on model preference hints, latency requirements, privacy sensitivity, or cost constraints. Model selection is performed based on client-defined capabilities using a preference structure that includes attributes like speed, cost, and intelligence priority, rather than hardcoded model identifiers, which ensures adaptability across heterogeneous environments.

The host retains control over all LLM interactions, acting as the intermediary that verifies prompt validity, aggregates multi-source context, and enforces user consent before any model invocation. Each client-server connection is sandboxed, preventing data leakage or unauthorized access across domains. The communication between components adheres to strict session and protocol rules, established during the initialization phase via JSON-RPC messaging. The Streamable HTTP transport supports rich server-sent interactions and session resumption using unique session identifiers and event cursors, while the stdio transport offers efficient local execution for edge deployments. Furthermore, the protocol supports advanced features such as cancellation, progress reporting, completion autocompletion, logging, and authorization via OAuth 2.1, making it robust and suitable for both high-trust environments and open multi-tenant platforms.

By decoupling context providers (i.e. servers), session managers (i.e. clients), and orchestration logic (i.e. host), the MCP architecture promotes modularity, scalability, and security. Its flexible transport mechanisms, strict capability negotiation, and extensible protocol design allow developers and researchers to construct AI-enhanced systems that are both powerful and privacy-preserving. This makes MCP a compelling foundation for building next-generation AI assistants, autonomous agents, IDE-integrated copilots, and interactive multi-modal systems that span local and distributed environments with confidence and control.

[3]https://github.com/ollama/ollama

[4]https://docs.gpt4all.io/index.html

[5]https://github.com/ggml-org/llama.cpp

[6]https://lmstudio.ai/

[7]https://docs.vllm.ai/

B. Host

The host component within the MCP architecture serves as the pivotal orchestrator and container, responsible for overseeing the interactions between clients and servers and managing their overall life cycle. It operates as the central entity that coordinates secure communication, session management, and integration of context across various servers and clients. By effectively abstracting the complexity of server-client interactions, the host simplifies the application developer's task, allowing developers to focus on their core business logic rather than getting entangled in infrastructural complexities.

Primarily, the host is tasked with creating and managing multiple client instances, each tailored to interact with specific servers. This approach ensures isolation, with clients serving as individual entities maintaining unique stateful sessions. The host carefully manages permissions and connectivity, granting or restricting access based on predefined security policies. This fine-grained control enables the host to enforce robust security measures effectively, mitigating risks and ensuring that sensitive data is handled appropriately.

Another fundamental responsibility of the host is managing the lifecycle of each client, from initialization to operation, through to graceful shutdown. During the initialization phase, the host coordinates capability negotiation, ensuring that both clients and servers are aware of the features they support, thus streamlining subsequent interactions. During normal operation, the host continuously monitors sessions for health and responsiveness, employing utilities such as ping and progress notifications to ensure optimal performance and responsiveness. In case of anomalies or inactivity, the host is equipped to gracefully terminate sessions, freeing up resources and maintaining overall system integrity. A critical aspect of the host's role is its ability to aggregate context from multiple clients and servers. Given the diverse range of capabilities offered by nt servers, the host centralizes this contextual information, making it accessible to clients in a coordinated and secure manner. This aggregation is crucial for facilitating rich, context-aware interactions with LLMs, significantly enhancing the quality and relevance of AI-driven outputs. The host also enforces user authorization and consent management. Employing OAuth 2.1, the host securely handles authentication and authorization flows, providing tokens that regulate access to resources and capabilities. By ensuring stringent authentication protocols and secure token management, the host plays a pivotal role in maintaining the security posture of the entire MCP system. Additionally, the host supports extensibility through its modular architecture, enabling incremental enhancements and easy integration of new features. As MCP evolves, the host can seamlessly adapt, incorporating new capabilities without disrupting existing functionalities, thus supporting continuous innovation and adaptation to emerging technological advancements.

By centralizing orchestration and security enforcement, the host significantly reduces the complexity faced by developers in managing multiple client-server interactions. Its role as a container and coordinator ensures robust, secure, and efficient management of context and interactions within the MCP framework, ultimately driving greater productivity and innovation in AI-driven application development.

C. Server

Servers constitute the critical infrastructure of the MCP, designed as specialized entities to provide contextual resources, tools, prompts, and various utilities essential for advanced interactions with LLMs. The architecture of MCP servers emphasizes modularity, composability, and extensibility, ensuring streamlined, secure, and effective interactions. Each server performs discrete roles, allowing developers and applications to leverage specialized functionalities seamlessly within an interoperable ecosystem.

MCP servers are designed with a highly modular, composable architecture that promotes incremental implementation and seamless integration of emerging capabilities. Each server can independently specialize in distinct functionalities, providing maintainable, reusable components. Capability negotiation within MCP ensures compatibility, allowing servers to advertise supported features explicitly and ensuring interoperability among diverse client implementations. Servers within MCP are built upon a modular, extensible architecture, allowing incremental enhancements and seamless integration of additional capabilities. Each server module remains highly composable, enabling developers to construct sophisticated applications from simpler, reusable units effectively. This modular design supports easy maintenance, promotes code reuse, and ensures robustness across various application scenarios. Servers adhere strictly to defined protocol standards, ensuring interoperability and seamless communication with MCP clients. Capability negotiation mechanisms ensure servers clearly advertise their supported features, maintaining compatibility and enabling continuous innovation and integration of emerging capabilities without disrupting existing systems.

Security is a foundational aspect of MCP servers, rigorously enforced through robust authorization and authentication mechanisms that adhere to OAuth 2.1 standards. These secure, token-based frameworks ensure controlled access to sensitive resources and capabilities, effectively mitigating unauthorized access risks and potential security breaches. Furthermore, isolation between servers is a core design principle, enforcing strict boundaries that prevent access to other servers' operational contexts or conversation histories. This ensures data privacy, prevents leakage, supports secure multi-tenant environments, and reinforces overall system integrity by allowing each server to function independently within its defined scope. Table III presents a comparison of local and remote MCP servers.

1) Prompts: Prompts serve as structured instructions or templates guiding interactions with LLMs, significantly enhancing consistency, efficiency, and clarity in user-driven engagements. They are intentionally user-controlled primitives, explicitly triggered through client applications based on user actions, such as context menu selections or slash commands

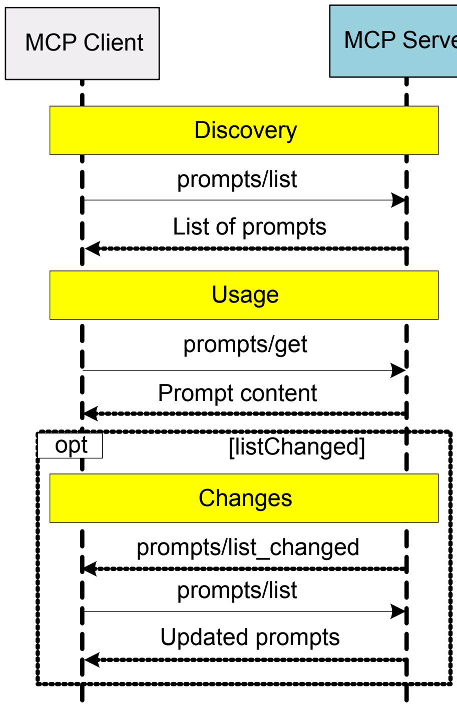

Fig. 2. MCP prompts workflow.

[57], [58]. Each prompt comprises clearly defined messages that provide standardized contexts, supporting predictable and repeatable LLM outputs. Prompts can be dynamically customized via arguments, allowing client applications to inject contextual parameters effectively, tailoring interactions to the current user context or operational needs [59], [60]. Servers expose these prompts through well-defined API endpoints, enabling client applications to discover, retrieve, and invoke them seamlessly. The life cycle of prompt management includes comprehensive discovery mechanisms, standardized JSON-RPC API communication, and structured prompt argument schemas. This ensures easy integration, maintainability, and adaptability across diverse MCP-enabled client applications. Figure 2 presents the workflow of prompts.

2) Resources: Resources are fundamental server components that deliver structured and unstructured data crucial for enriching LLM interactions. These resources are uniquely identified by Uniform Resource Identifiers (URIs)[8] and can include varied content types such as plain text, binary files, images, audio, video, and structured documents. Resources are application-controlled primitives, managed by client applications, allowing diverse strategies for context incorporation—from explicit user-driven resource selection to automatic contextual enrichment based on internal application heuristics. Servers facilitate resource discovery and retrieval via standardized methods like resources/list, resources/read, and dynamic resource templates, ensuring efficient access and handling. Figure 3 presents the workflow of resources. Advanced MCP server implementations support subscription mechanisms (resources/subscribe), enabling real-time notifications of resource changes. This proactive management ensures

[8]https://www.w3.org/Addressing/URL/uri-spec.html

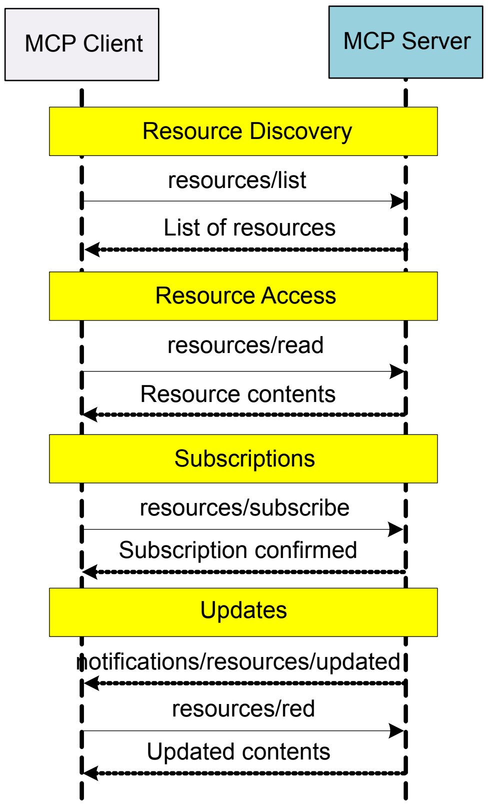

Fig. 3. MCP resources workflow.

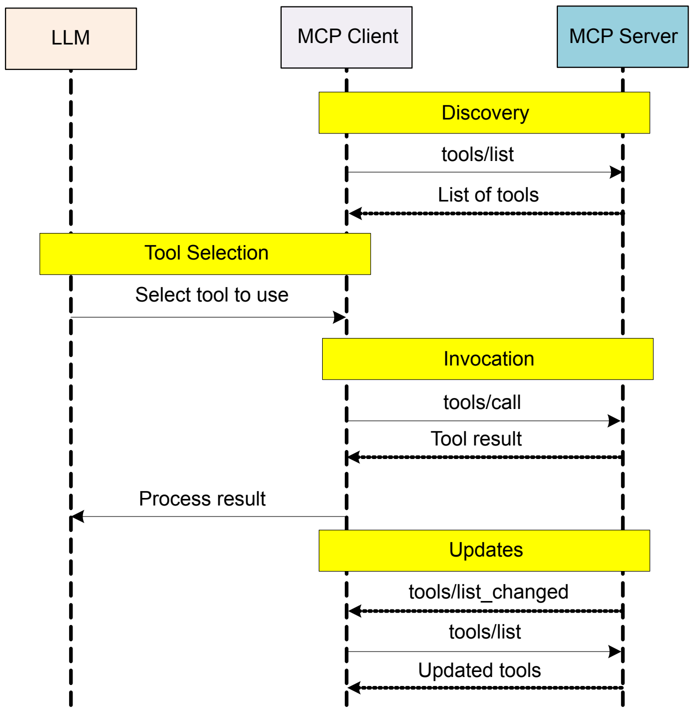

Fig. 4. MCP tools workflow.

up-to-date contextual synchronization between servers and clients, maintaining the relevance and accuracy of the provided contexts.

3) Tools: Tools provide LLMs with direct, model-driven capabilities to interact and execute actions within external systems or data environments [61]. These are defined as callable functions on the server side, each equipped with clear JSON schema-based input parameters and explicit documentation. By empowering LLMs to autonomously invoke these functionalities, tools significantly enhance the practical application and decision-making capabilities of AI systems [62]. Despite their autonomous nature, tool invocation always includes stringent security checks, enforced through human-in-the-loop mechanisms embedded within client applications. This model-controlled primitive structure promotes transparency and security while offering advanced operational capabilities [63], [64]. Servers actively advertise available tools through standardized API methods (tools/list and tools/call), providing comprehensive discovery and invocation capabilities for LLMs. This transparency enables LLMs to utilize these capabilities contextually, optimizing interaction quality and task automation in complex operational scenarios. Figure 4 presents the workflow of tools in MCP server.

4) Utilities: Utilities form essential cross-cutting concerns enhancing server reliability, operational transparency, and maintainability. These include structured logging, auto-completion suggestions, connection health monitoring through pagination operations.

- Logging
  Structured system for logging, allowing servers to send detailed log messages to clients in a standardized format. Clients can configure the minimum log level to control the volume of messages they receive, while servers send logs tagged with severity levels, optional logger names, and additional contextual data in JSON format. Log levels follow the widely adopted syslog standard (RFC 5424)[9], ranging from debugging info to critical and emergency alerts. Clients may adjust these levels dynamically using a logging/setLevel request, and servers respond with log messages using notification protocols. The system is flexible—implementations can choose how to display logs, such as visually in the UI or via persistent logs. Security is very essential in this scenario. Sensitive data like credentials or personal details must never appear in logs. Best practices include rate limiting, consistent logger naming, and validating data before transmission to ensure privacy and system integrity.
- Completion
  The Completion feature in the MCP allows servers to offer intelligent, real-time suggestions for filling in arguments in prompts or resource URIs, similar to code completion in IDEs[10]. As users type, applications can display dynamic suggestions—like dropdowns or pop-ups—based on context, although the specific UI implementation is flexible. To enable this, servers must declare the completions capability and respond to client requests containing reference types (i.e. prompt names or resource URIs) and partial argument values. Responses include a list of suggested completions, optionally indicating

[9]https://datatracker.ietf.org/doc/html/rfc5424
[10]https://github.com/JetBrains/mcp-jetbrains

TABLE III
COMPARISON OF LOCAL AND REMOTE MCP SERVERS

Aspect	Local MCP Server	Remote MCP Server
Process startup and transport	Spawned by the Host (e.g., `npx mcp-server-filesystem ...`). Begins reading newline-delimited JSON-RPC from stdin and writing to stdout immediately.	Runs as an independent web service (container, VM, FaaS). Exposes a single `/mcp` endpoint that accepts HTTP POST for every client request and may hold an SSE stream open for server-push.
Capability focus	• Surface local context: files, Git repos, databases on the same machine, OS APIs, desktop apps, CLI tools. • Low-latency, high-bandwidth access to the filesystem and other local resources. • Can emit frequent notifications/resources/updates as files change.	• Surface cloud or SaaS context: web search, hosted DBs, SaaS APIs, organization knowledge bases. • Often multiplexes many tenants or datasets; must enforce per-tenant isolation. • Streams long results or progress via SSE.
Security model	• Runs under the Host's user account; inherits file permissions. • Host constrains it with an allow-list of root paths or CLI arguments. • No OAuth—credentials (if any) come from env vars or CLI flags.	• Implements OAuth 2.1 (PKCE) and may support Dynamic Client Registration. • Issues/validates `Mcp-Session-Id` headers to keep stateful sessions. • Must rate-limit, log, and possibly bill per request.
State and sessions	State is often in-memory or on local disk; dies with the process. No session header needed—one stdio pipe == one session.	Can keep durable state keyed by session ID (e.g., user cart, RAG cache). Must resume SSE streams after network drops when `Last-Event-ID` is provided.
Concurrency expectations	Typically single-client (the spawning Host). Simple concurrency model—may rely on the Host to serialize requests.	Must handle many concurrent Clients; HTTP requests may overlap, SSE pushes are asynchronous.
Error and recovery	On crash, Host restarts the subprocess. Uses exit code + stderr for diagnostics.	Needs robust 5xx handling, exponential back-off, and health probes. Should expose `/healthz` or similar for orchestration systems.
Update cadence	Can ship bleeding-edge features—users upgrade by updating the CLI package.	Must maintain backward compatibility; may serve multiple protocol versions side-by-side.
Typical examples	• `server-filesystem` – secure file access. • `server-git` – interact with a local repo. • `server-puppeteer` – drive a headless browser installed on the machine.	• `server-brave-search` – web search via Brave API. • `server-postgres` (managed DB) – read-only SQL over HTTPS. • `server-slack` – operate on a Slack workspace using OAuth scopes.

the total matches and whether more suggestions are available. The protocol ensures structured communication via JSON-RPC, supports fuzzy matching and relevance ranking, and enforces security through input validation and rate limiting. Clients are encouraged to debounce rapid requests, cache results, and handle errors or partial data smoothly.

- Pagination
  Pagination in the MCP helps manage large data sets by breaking responses into smaller, more manageable chunks using opaque cursor tokens. This cursor-based method is more flexible than numbered pages and is especially useful for both internet-based services and local setups to maintain performance. Servers decide the page size and return a *nextCursor* if more data exists, allowing clients to continue the request sequence. Clients must treat cursors as opaque, avoiding assumptions about their format or persistence.

D. Client

MCP Clients serve as pivotal intermediaries within the MCP ecosystem, facilitating stateful, secure, and structured interactions between host applications and specialized servers. Each client instance represents a singular and isolated session with a corresponding MCP server, designed explicitly to handle context exchanges and capability negotiations seamlessly. Clients function to uphold rigorous protocol standards, ensuring that interactions remain robust, secure, and efficient across all architectural layers.

Clients within the MCP architecture establish and manage persistent, stateful sessions with their corresponding servers. This approach is fundamental to maintaining a coherent and continuous exchange of information and context over the session life cycle. Each session is explicitly initiated with a clearly defined handshake through an initialization phase, which confirms protocol versions and negotiates capabilities. This stateful management approach allows clients to dynamically handle context, updates, and interaction continuity effectively, significantly enhancing reliability and responsiveness during long-running or multi-step interactions. Session statefulness further implies that clients maintain robust session identifiers, track ongoing interactions, and handle server responses contextually. Clients are responsible for session resilience, including the handling of interruptions, timeouts, reconnections, and explicit session closures. This management is crucial to prevent resource leaks, enhance security, and sustain optimal application performance.

Capability exchange is a core aspect of MCP clients, enabling dynamic, runtime negotiation of available features and supported interactions between clients and servers. Through capability declarations, clients articulate their supported functionalities—such as sampling, notification handling, and specific data exchange formats—clearly defining their scope and constraints upfront. This mechanism ensures compatibility, reduces miscommunication risks, and enables progressive enhancement as both clients and servers evolve independently. Capability negotiation involves clearly defined JSON-RPC interactions, ensuring structured, transparent, and machine-readable communication of client capabilities during session initialization. Clients explicitly inform servers about the features they support, allowing servers to tailor their interactions and requests accordingly. This structured capability negotiation mechanism is central to MCP's extensibility, accommodating new and emerging functionalities seamlessly without disrupting existing interactions.

MCP clients rigorously uphold security and isolation principles through stringent enforcement of session boundaries, capability constraints, and context exchanges. Clients mediate and route protocol messages bidirectionally, ensuring secure interactions and preventing cross-contamination between isolated server sessions. By enforcing these boundaries explicitly, clients ensure the security and privacy of interactions, limiting server access strictly within agreed-upon contexts and capabilities. Client-side enforcement of security policies and context isolation guarantees that sensitive data remains protected, minimizing exposure risks. Additionally, the explicit, structured nature of capability negotiations and context exchanges ensures robust accountability and auditability of all interactions, enhancing operational security further.

1) Roots: Clients provide structured representations of filesystem or data hierarchies through the concept of roots, defining explicit operational boundaries within which MCP servers can securely operate. Roots are essentially designated areas of the client's data environment, explicitly exposed to servers for controlled access. Clients manage root lists dynamically, updating server-side contexts upon changes through notifications. This structured boundary management ensures secure, predictable interactions, preventing unauthorized data access or modification beyond established root directories. Clients must robustly enforce these boundaries, validating all access requests against predefined roots to safeguard data integrity and security effectively. Figure 5 demonstrates the MCP client's roots workflow.

2) Sampling: Sampling is another essential feature facilitated by clients, which empowers MCP servers to request AI-generated content, such as text, audio, or visual completions from integrated LLMs. Clients act as gatekeepers for sampling requests, maintaining strict control over the selection and utilization of LLM resources. Sampling requests are processed via structured JSON-RPC methods, clearly specifying model preferences, priorities (e.g. speed, intelligence, and cost), and other parameters necessary for precise model selection. Clients ensure transparency and user agency by embedding human-in-the-loop approvals for sampling interactions. This process mitigates risks associated with unauthorized or excessive model usage, maintaining secure, ethical, and user-aligned operations.

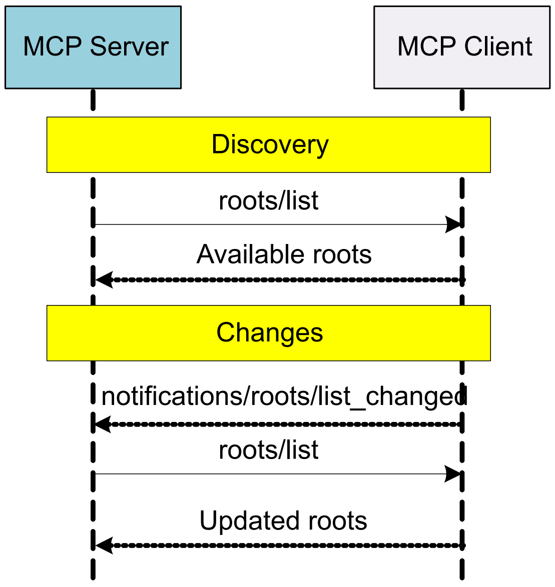

Fig. 5. MCP client roots message flow.

Additionally, clients handle response parsing, delivery, and error handling, ensuring robust, contextually appropriate interactions. Figure 6 represents the sampling workflow.

E. Base MCP Protocol

The Base Protocol of the MCP defines foundational elements necessary for reliable, structured, and secure communication between MCP clients and servers. It includes core components such as (i) transports, (ii) authorization, (iii) life cycle management, and (iv) utilities, which together establish a robust communication framework ensuring interoperability and extensibility across applications.

1) Transports: In the MCP, transports define the mechanism by which clients and servers exchange JSON-RPC 2.0 messages. These messages are UTF-8 encoded and represent all protocol interactions—from initialization to capability negotiation, sampling requests, and resource management. The MCP standard supports two primary transport layers: stdio and Streamable HTTP, each serving different deployment contexts and design preferences. Both approaches maintain compatibility with MCP's core design goal—modularity, isolation, and seamless integration into diverse application environments. MCP's dual-transport architecture provides flexibility without compromising the protocol's core semantics. stdio is ideal for lightweight, embedded use cases where performance and simplicity are prioritized. Streamable HTTP, on the other hand, offers the scalability and resilience needed for modern cloud-native or multi-client deployments. Implementers may also define custom transports so long as they preserve the JSON-RPC messaging format and adhere to MCP's protocol life cycle—ensuring that even beyond these two, the protocol remains adaptable to future communication paradigms.

Before delving in such two standards, we will look into JSON-RPC in context of MCP. We then dive into these two standard transports such as (i) stdio and (ii) Streamable

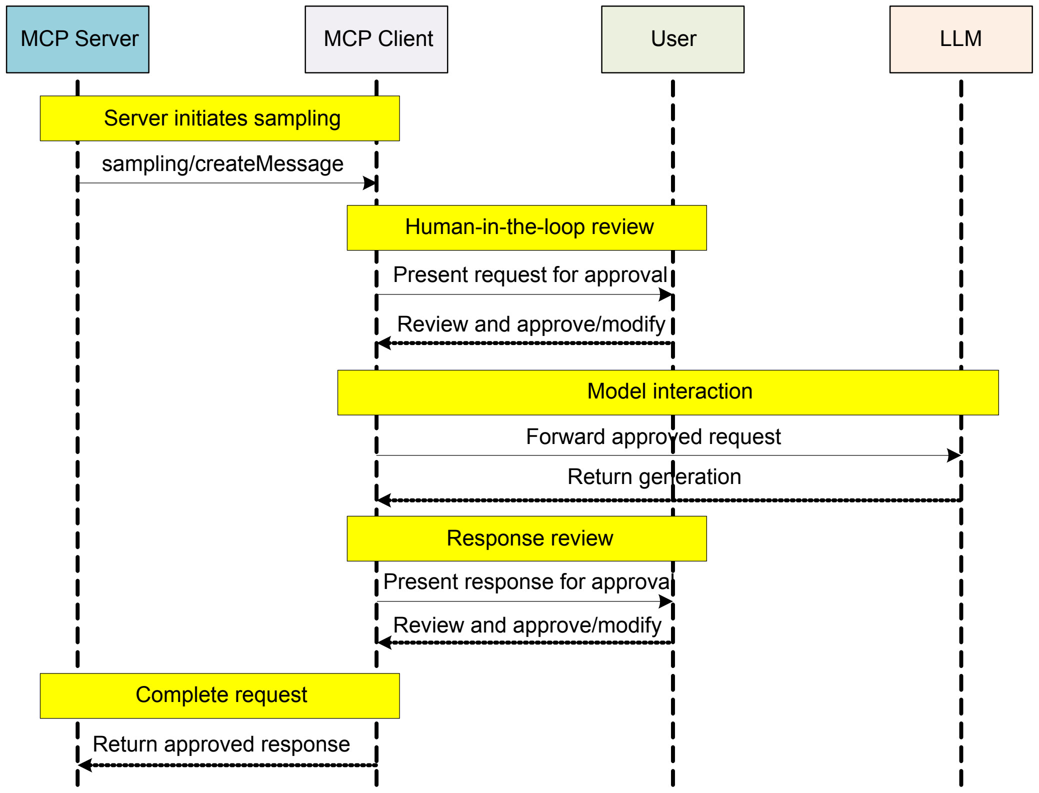

Fig. 6. MCP client sampling message flow.

HTTTP, outlining their mechanisms, usage, advantages, and operational considerations.

- JSON-RPC 2.0
  At the core of the MCP lies JSON-RPC 2.0, a minimalist, stateless remote procedure call protocol designed to enable structured communication between clients and servers using JSON. Its simplicity, language-agnostic format, and transport independence make it an ideal foundation for asynchronous, bi-directional message passing in systems like MCP, which involve frequent contextual exchanges between agents, tools, and large language models. In the context of MCP, JSON-RPC messages act as the carrier format for all meaningful protocol interactions—from invoking tools and reading resources to subscribing to updates and handling logging or sampling events. By adhering to the JSON-RPC 2.0 specification, MCP ensures predictable communication patterns, interoperability across different implementations, and extensibility for future protocol evolutions.
  JSON-RPC 2.0 standardizes three main message types: requests, responses, and notifications. A typical request message initiates a remote method call and contains four essential fields: "*jsonrpc*" (which is always "*2.0*" to denote protocol version), "*method*" (i.e. the name of the function to invoke), optional "*params*" (where arguments to be passed), and an "*id*" (a unique identifier to correlate with the response). This structure allows for clearly defined, traceable interactions. When a request is valid and successfully processed, the server returns a response containing the same "id" and either a "*result*" (in case of success) or an "*error*" object (on failure). This duality enforces strict consistency and makes debugging and tracking operations straightforward.
  An important variant of the request message is the notification—a request that omits the "*id*" field, signaling that no response is expected. Notifications are useful for fire-and-forget operations, where the client does not require any acknowledgment. However, since no error or confirmation is returned, their use must be reserved for non-critical operations where reliability isn't paramount.
  The params field in JSON-RPC 2.0 supports two formats: positional parameters using arrays, and named parameters using JSON objects. This flexibility allows developers to structure method inputs in a way that best matches their API design, whether ordered or key-based.
  In cases of failure, error responses are returned with a standardized error object that includes a numeric error "*code*", a short descriptive "*message*", and optional "data" providing additional context. Common error codes include -32600 for invalid requests, -32601 for unknown methods, and -32700 for malformed JSON input. These predefined codes support rapid identification and resolution of issues during runtime.
  One of JSON-RPC 2.0's powerful features is its batching mechanism. This allows clients to send multiple requests (or notifications) in a single array, enabling parallelism and reducing the overhead of repeated connections. Servers can process these batched requests in any order

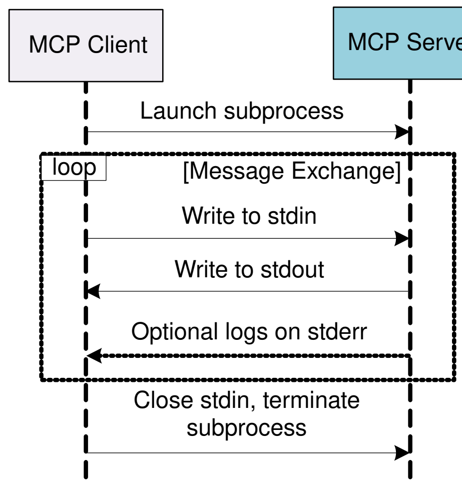

Fig. 7. Transport stdio workflow.

and return corresponding responses as an array, preserving efficiency without compromising the independence of each call.

- stdio
  The stdio transport is particularly well-suited for scenarios where the server process is tightly integrated with the host application—common in local desktop environments or command-line tools[11]. In this mode, the client launches the server as a subprocess and communicates with it through the operating system's standard input and output streams. This interaction is strictly confined to valid MCP messages encoded as JSON-RPC, with newline-delimited formatting ensuring reliable parsing. The server reads input from *stdin*, processes the message, and sends responses or notifications via *stdout*. Any non-protocol logs or diagnostics may be written to *stderr*, which the client may choose to capture or ignore.
  This transport method offers several benefits. It introduces minimal complexity, requiring no network configuration or port management, which is ideal for tooling like IDE extensions or language servers. It also enables fast, low-latency communication, as data flows directly between processes on the same machine. Security is inherently improved through process-level isolation, and since the server is short-lived and spawned on demand, resource management is straightforward.
  However, the simplicity of stdio comes with a trade-off. Because the server is tightly bound to the client's process life cycle, handling process crashes, unexpected termination, or high concurrency requires deliberate orchestration. If the client closes its *stdin*, the server must interpret this as a signal to shut down cleanly, preserving system stability. Despite this, stdio remains the transport of choice for lightweight, embedded, or plugin-driven MCP integrations where simplicity and speed are paramount. Figure 7 illustrates the stdio workflow.
- Streamable HTTP
  In contrast to the tight coupling of stdio, Streamable HTTP offers a decoupled, network-ready alternative ideal for remote or multi-client deployments[12]. Here, the server operates as a standalone service accessible over HTTP, supporting both HTTP POST and HTTP GET methods on a single MCP endpoint (e.g. https://example.com/mcp). Clients interact with this endpoint to send JSON-RPC messages, initialize sessions, receive real-time updates, and process bidirectional communication—thanks to its optional SSE stream capability.
  The interaction begins when the client sends a POST request to initialize the session. The server responds with an *Mcp-Session-Id*, which must be included in all subsequent requests. This session management layer introduces logical isolation and ensures message continuity across long-lived interactions. After initialization, clients may continue issuing POST requests for prompts, tools, or sampling, while also opening a persistent GET connection to listen for SSE messages from the server—such as progress updates, notifications, or server-initiated requests. Figure 8 illustrates the Streamable HTTP workflow.
  What sets Streamable HTTP apart is its support for asynchronous and streaming communication. Servers can emit JSON-RPC messages in real time over an SSE stream, even before responding to a client's POST request. This is especially powerful in use cases involving streaming completions, background computation, or progress reporting. The server manages message delivery, ensures responses align with session context, and may even resume interrupted streams using SSE's *Last-Event-ID* mechanism—improving reliability across unstable networks.
  Despite its power, this transport is more complex to implement than stdio. It demands rigorous session handling, graceful management of disconnections, and proper handling of streaming logic. Yet for distributed applications, web platforms, and AI agents operating over a network, Streamable HTTP provides a scalable, secure, and fully-featured pathway for MCP communication.

2) Authorization: Authorization in the MCP enables secure, permissioned access between clients and servers, especially in environments where protected data or sensitive actions are involved. While MCP's core functions like prompt delivery and resource handling can operate in open configurations, any real-world application typically requires access control, which is addressed via OAuth 2.1-based mechanisms. MCP's authorization layer is crafted to align with modern security standards while maintaining compatibility with various server architectures and user flows. Let's explore the key components of MCP's authorization stack in detail. It serves as a gatekeeping mechanism. It ensures that only authenticated and explicitly permitted clients can interact with secured resources on MCP servers. These servers may host APIs, tools, or prompts that interface with private data or initiate impactful

[11]https://modelcontextprotocol.io/specification/2025-03-26/basic/transports#stdio

[12]https://modelcontextprotocol.io/specification/2025-03-26/basic/transports#streamable-http

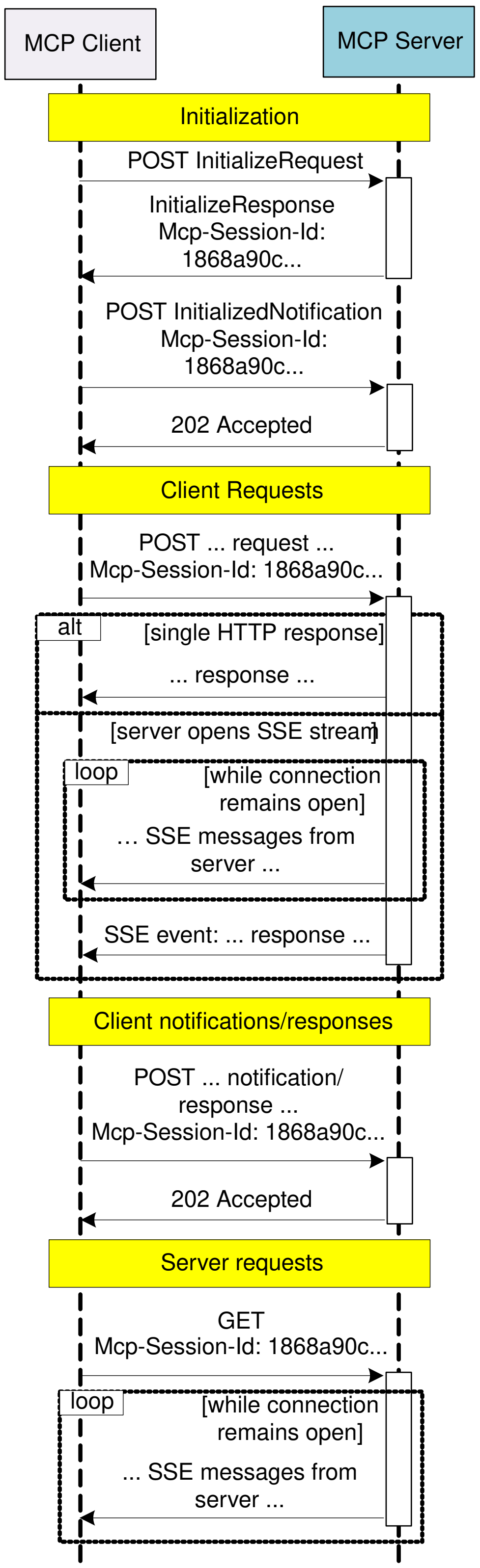
MCP Client
MCP Server
Initialization
POST InitializeRequest
InitializeResponse Mcp-Session-Id: 1868a90c...
POST InitializedNotification Mcp-Session-Id: 1868a90c...
202 Accepted
Client Requests
POST ... request ... Mcp-Session-Id: 1868a90c...
alt
[single HTTP response]
... response ...
[server opens SSE stream]
loop
[while connection remains open]
... SSE messages from server ...
SSE event: ... response ...
Client notifications/responses
POST ... notification/ response ... Mcp-Session-Id: 1868a90c...
202 Accepted
Server requests
GET Mcp-Session-Id: 1868a90c...
loop
[while connection remains open]
... SSE messages from server ...

Fig. 8. Transport streamable HTTP workflow.

actions similar to like writing to files or accessing cloud storage.

In practice, when a client sends a request without valid credentials, the MCP server responds with an HTTP 401 Unauthorized status. This response signals the need for OAuth authorization. The client must then initiate the flow to retrieve an access token, which it will use in subsequent communications. The key benefit of MCP's transport-layer authorization is that it decouples authentication logic from server implementation, allowing flexibility in how tokens are issued and validated. MCP supports optional integration of this flow for stdio-based servers but requires it for HTTP-based transports. When omitted, local configuration or user-side approval must act as a substitute security mechanism. MCP's authorization framework is both powerful and adaptable. By integrating modern OAuth 2.1 standards—including dynamic registration and server metadata discovery—it supports both traditional and federated login systems. These mechanisms allow developers to build AI-driven applications with strong security guarantees, streamlined user flows, and flexible deployment models. Whether for private tools or enterprise-grade LLM orchestration, MCP's authorization ensures trust, accountability, and safe data exchange at every step. The complete authorization workflow is shown in Figure 9.

Its layered structure, grounded in open standards and extensible via optional flows, positions MCP to support both minimal setups and highly regulated environments. Through a balanced design of automation, configurability, and compliance, it offers a best-practice model for modern AI service authentication. Authorization is complex process and incurs several aspects thereto as explained below.

- Public Clients using OAuth 2.1 and PKCE
  For public clients like local desktop applications or thin clients running within user environments, MCP adopts the OAuth 2.1 Authorization Code Flow with Proof Key for Code Exchange (PKCE)[13]. This modern approach mitigates risks associated with authorization code interception and client impersonation. Upon detecting that the user is not yet authorized (via an HTTP 401), the client generates a *code_verifier*—a securely random string—and derives a *code_challenge* by hashing it. The client opens the browser with an authorization URL that includes the code_challenge, prompting the user to log in and grant access. After authorization, the server redirects the user to a predefined callback URL with an authorization code. The client then submits a token request with the authorization code and the original *code_verifier* to the token endpoint. This step finalizes the handshake and returns an access token (and optionally a refresh token). This token is then included in subsequent requests using the Authorization: Bearer header. This mechanism keeps the user's credentials secure while enabling seamless LLM workflows that depend on remote context or server-driven logic. This flow allows MCP to leverage industry-grade security without embedding sensitive secrets in the client,

[13]https://oauth.net/2/pkce/

making it especially appropriate for distributed, untrusted, or open-source client deployments.

- Server Metadata Discovery
  To streamline integration and minimize client-side configuration, MCP implements OAuth 2.0 authorization server metadata discovery as outlined in RFC 8414[14]. Instead of hardcoding OAuth endpoints (e.g., for authorization, token, or registration), clients dynamically retrieve these values using a GET request to the server's well-known metadata endpoint: *GET /.well-known/oauth-authorization-server* When this endpoint is available, the server returns a metadata document containing URLs for */authorize*, */token*, and */register*. If discovery fails—typically via a 404 Not Found—clients fall back to default endpoint paths derived from the base URL. This design ensures forward compatibility with evolving server deployments. Clients are encouraged to include the MCP-Protocol-Version header in discovery requests, enabling the server to return protocol-aware responses. Metadata discovery eliminates guesswork, simplifies *onboarding*, and minimizes the chances of client *misconfiguration*, which is crucial for automated systems or large-scale deployments.
- Dynamic Client Registration
  In many deployments, MCP clients connect with previously unknown or dynamically spawned servers. To simplify onboarding and avoid manual configuration, MCP supports OAuth 2.0 Dynamic Client Registration (RFC 7591[15]). This allows clients to programmatically register and obtain a client ID (and optionally a secret) for use in the authorization code flow. If supported, the client sends a registration request including metadata like redirect URIs and desired capabilities. The server responds with a client identifier that is stored and reused in future sessions. When registration is unsupported, MCP clients may prompt users to enter their credentials or embed static credentials in configuration files. While less flexible, this fallback ensures operability in static or restricted environments. Dynamic registration is a key enabler for scalable AI deployments. In scenarios where clients need to dynamically connect to microservices or per-user instances, the ability to register and authenticate programmatically removes friction and enhances automation.
- Authorization Flow Decision Tree
  The full MCP authorization flow is orchestrated as a layered decision tree, ensuring robustness across a wide range of server capabilities. The flow begins with metadata discovery. If that succeeds, the client extracts endpoint URLs from the metadata. If not, it defaults to standard endpoints like */authorize* and */token*. Next, the client checks whether the server supports dynamic registration. If available, registration proceeds programmatically. If not, the client either uses hardcoded credentials or prompts the user for manual registration. Once the client is registered and the authorization endpoints are known, it generates PKCE parameters, initiates the authorization request via browser, captures the authorization code, and exchanges it for a token. This token becomes the key credential for subsequent requests. The structured flow ensures that clients can adapt to various levels of server support while providing a unified user experience.
- Third-Party Authorization Flow
  MCP's flexibility extends to federated identity models. In enterprise environments or public integrations, the MCP server can delegate authorization to a third-party identity provider like Google OAuth[16], Microsoft Azure AD[17], or Okta[18]. In this delegated flow: (i) the client initiates an OAuth request with the MCP server, (ii) the MCP server redirects the user to the third-party provider for authentication, (iii) upon user authorization, the provider returns an authorization code to the MCP server, (iv) the MCP server exchanges the code for a third-party token, and (v) the MCP server then issues its own bound MCP token for client use. This hybrid model preserves the external identity source as the system of record while maintaining session security and auditability within MCP. To be secure, the MCP server must tightly bind third-party tokens to local sessions and enforce token revocation policies. Third-party authorization support enables integration with Single Sign-On (SSO) systems, enterprise user directories, and centralized identity providers, all of which are critical in regulated or large-scale organizational environments.
- Token Management and Security Best Practices
  Token security is paramount in any OAuth-based system. In MCP, access tokens must only be transmitted in HTTP headers—not in query strings or stored in plaintext logs. Tokens should be treated as sensitive assets. Clients should store tokens securely—ideally in encrypted local storage or hardware-backed keychains. Refresh tokens, if issued, must be managed carefully to avoid long-lived credentials falling into the wrong hands. Servers should issue short-lived access tokens, enforce rotation policies, and use HTTPS exclusively for all endpoints. Redirect URIs must be pre-validated to guard against redirection-based attacks. Moreover, all token issuers must validate scopes, client identities, and request integrity. To further strengthen security, applications should implement token expiration monitoring, proactive refresh logic, and revoke unused tokens. Proper logging of access attempts and anomalies helps in forensic analysis and compliance.
- Error Handling in Authorization
  To ensure robust interaction and user feedback, MCP authorization endpoints return precise HTTP status codes for various failure scenarios:
  - 401 Unauthorized: The request lacks a valid access token or the token is expired.

[14]https://datatracker.ietf.org/doc/html/rfc8414
[15]https://datatracker.ietf.org/doc/html/rfc7591
[16]https://developers.google.com/identity/protocols/oauth2
[17]https://www.microsoft.com/en-in/security/business/identity-access/microsoft-entra-id
[18]https://www.okta.com/

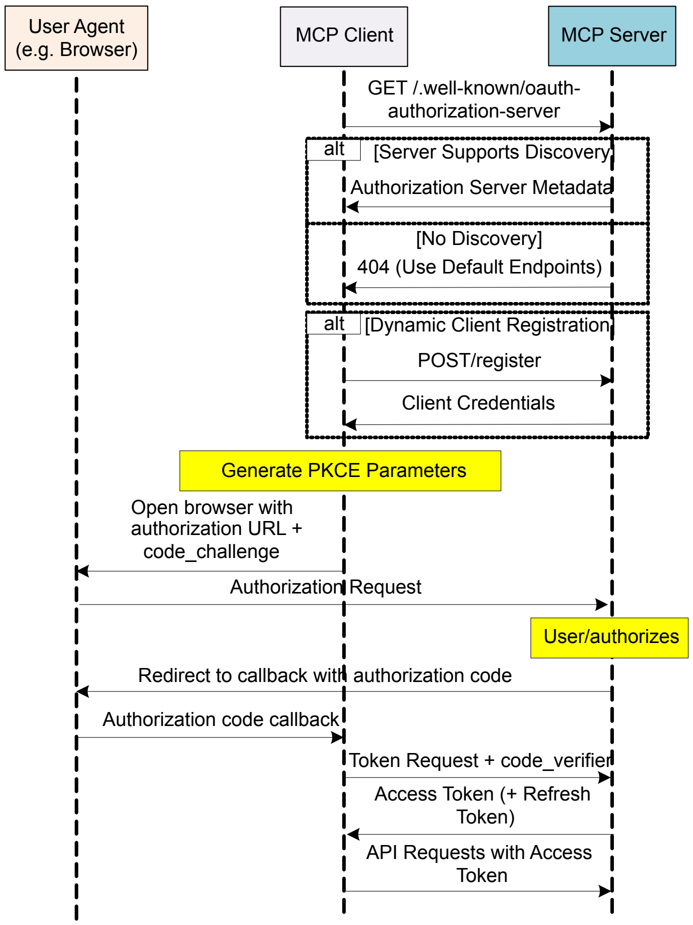
User Agent (e.g. Browser)
MCP Client
MCP Server
GET /.well-known/oauth-authorization-server
alt [Server Supports Discovery]
Authorization Server Metadata
[No Discovery]
404 (Use Default Endpoints)
alt [Dynamic Client Registration]
POST/register
Client Credentials
Generate PKCE Parameters
Open browser with authorization URL + code_challenge
Authorization Request
User/authorizes
Redirect to callback with authorization code
Authorization code callback
Token Request + code_verifier
Access Token (+ Refresh Token)
API Requests with Access Token

Fig. 9. MCP authorization workflow.

– 403 Forbidden: The token is valid but does not grant the required scope.
– 400 Bad Request: The OAuth request is malformed or missing fields.

Clients should interpret these codes contextually. A 401 often indicates the need to restart the authorization flow, while a 403 may require prompting the user to adjust permissions or scopes. The error logging and structured error responses which follow JSON-RPC error object conventions to help clients adapt their behavior, enhance reliability, and provide actionable messages to users or administrators.

3) Life Cycle: Life cycle management within the MCP provides a structured framework for initializing, maintaining, and gracefully terminating client-server sessions. This ensures that both ends of a connection operate with a mutual understanding of capabilities, protocol versions, and active features. The life cycle is divided into three primary stages: (i) initialization, (ii) operation, and (iii) shutdown as explained below. Figure 10 demonstrates the MCP life cycle process.

- Initialization Phase
  The life cycle begins with the initialization phase, which establishes the foundation for all future communications. At this stage, the client initiates the interaction by sending an initialize request to the server. This request includes the supported protocol version, a description of client capabilities (e.g. sampling, roots), and implementation metadata such as the client name and version. This request must not be sent as part of a JSON-RPC batch, ensuring it receives prioritized processing before any subsequent communication. Upon receiving the request, the server responds with its own set of capabilities and metadata. This handshake validates that both client and server can engage in a mutually intelligible session. The version negotiation process ensures backward and forward compatibility. If the server does not support the requested version, it will reply with the latest version it does support. Clients that cannot accommodate the server's version should terminate the connection. Once both ends have exchanged and acknowledged initialization data, the client sends a *notifications/initialized* message to confirm readiness for standard operations. This exchange ensures that neither party proceeds with core protocol actions until both initialization and mutual compatibility are confirmed.
- Operation Phase
  The operation phase begins after successful initialization and lasts for the duration of the session. During this phase, clients and servers engage in routine protocol communications based on the capabilities negotiated earlier. These may include exchanging prompts, tools, resources, sampling requests, or logging events. Each party is expected to adhere strictly to the capabilities agreed upon during initialization. For instance, if the server did not declare support for prompts, the client must not attempt prompt-related requests. Similarly, if the client declared support for sampling, it must handle incoming sampling requests gracefully. To ensure performance and reliability, implementations should implement timeout mechanisms for requests. If a request is not acknowledged within a predefined period, the sender may send a cancellation notification and halt any further processing. These timeouts prevent resource exhaustion and keep the session resilient to delays or failures. Middleware and SDKs supporting MCP should offer configurable timeout settings to accommodate varying network and compute environments. Progress notifications are also supported during long-running operations. By including a progress token with a request, clients can track and display real-time updates to users or systems, improving transparency and user experience during heavy tasks.
- Shutdown Phase
  The shutdown phase marks the end of the life cycle, when either the client or server decides to close the connection. Unlike initialization, this stage does not involve formal shutdown messages. Instead, it relies on the underlying transport mechanism to terminate communication. For MCP implementations using stdio, the client should initiate shutdown by closing its input stream to the server and waiting for the process to exit. If the server does not terminate gracefully, the client may escalate to sending SIGTERM, followed by SIGKILL as a last resort. Servers may also initiate shutdown by closing their output streams

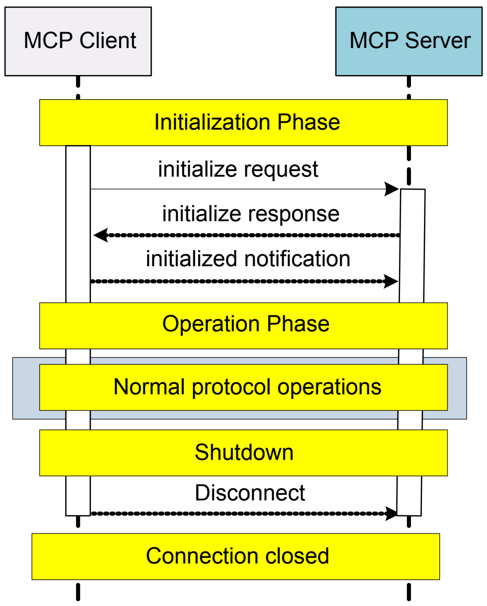

Fig. 10. MCP life cycle.

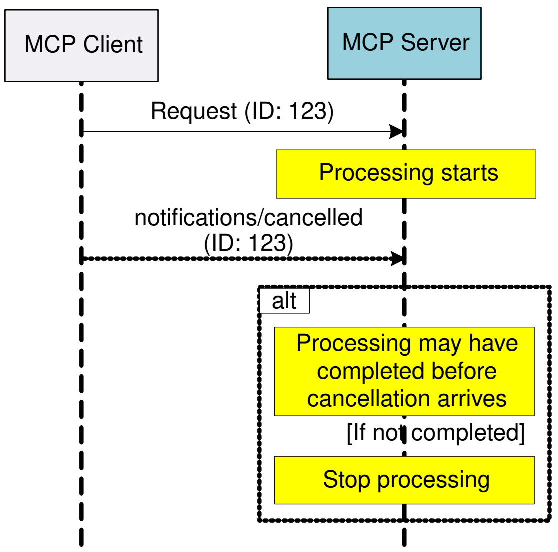
MCP Client
MCP Server
Request (ID: 123)
Processing starts
notifications/cancelled (ID: 123)
alt
Processing may have completed before cancellation arrives
[If not completed]
Stop processing

Fig. 11. MCP cancellation workflow.

and exiting. In HTTP-based implementations, shutdown is generally indicated by terminating the HTTP session or closing SSE (Server-Sent Events) streams. Servers may respond with an HTTP 404 status code to indicate a terminated session. Clients that detect such responses are expected to initiate a fresh session using a new initialize request. The absence of explicit shutdown messaging reflects MCP's modular and lightweight nature. However, developers should implement logging and error-handling mechanisms to track the closure of sessions for audit and debugging purposes.

4) Utilities: The Model Context Protocol provides several utility features designed to improve reliability, responsiveness, and user experience during client-server interactions. These utilities are optional but strongly recommended for robust implementations. The three most notable utilities include: (i) ping, (ii) cancellation, and (iii) progress.

- Ping
  The ping utility is a foundational health-check mechanism built into MCP to ensure active connectivity between the client and server. In long-lived or persistent sessions, where periods of silence are common, ping offers a proactive way to validate that the communication channel remains open and functional. The implementation uses a minimalistic JSON-RPC ping request—containing no parameters and only a unique identifier. The receiving party must promptly respond with an empty result, signifying that the connection is alive. If no response is received within a configurable timeout, the initiating party may assume the connection is stale and begin recovery procedures such as reconnection, re-initialization, or alerting the user. Effective ping strategies include adaptive timing, backoff mechanisms, and logging. Frequent pings may introduce unnecessary load, while infrequent checks may fail to detect issues early. Thus, implementations should allow customization of ping intervals based on network reliability and user expectations. Integrating ping failures into a broader monitoring framework also supports fault diagnosis and system stability.
- Cancellation
  CThe cancellation utility enhances user experience and system efficiency by allowing either party to terminate a request in progress. This is particularly useful in interactive environments where user actions can change quickly, rendering some operations obsolete before completion. Cancellation is triggered through the *notifications/cancelled* message, which references the unique ID of the request to cancel. The notification may also include a human-readable reason, aiding in debugging or logging. Upon receiving such a message, the recipient must stop processing the request, free up any resources it was using, and refrain from returning a response. It is important to recognize the unidirectional nature of cancellations—clients may cancel their outgoing requests, and servers may cancel theirs, but each party can only reference its own outbound traffic. This model ensures clarity and avoids unintended disruptions. Race conditions are gracefully handled by treating cancellations as advisory: if a request is already complete, the notification is ignored. This design avoids unnecessary complexity while still supporting responsive behavior. UI layers can reflect cancellation status visually, giving users control over long operations. Figure 11 shows the procedure behind cancellation.
- Progress
  Progress tracking introduces transparency and predictability into long-running operations within MCP. Instead of leaving users or systems in the dark while a task executes, the progress utility allows the recipient of a request to emit periodic updates keyed to a *progressToken* specified by the initiator. Each progress notification includes the original token, a numeric progress metric

(e.g. percentage completed), and optionally a total goal or textual message. This enables detailed and user-friendly progress visualization in UIs or terminal logs. It is particularly useful in compute-heavy workflows such as model sampling, file scanning, or tool chaining. Progress tokens must be unique across active sessions and may follow any alphanumeric format. Their association with a specific request ensures context is preserved throughout execution. While entirely optional, supporting progress helps systems become more observable and trustworthy. It empowers automation scripts to handle long waits intelligently and enables end-users to make informed choices about canceling or waiting. Rate-limiting and message deduplication should be considered to prevent flooding clients with redundant updates.

IV. State-of-the-Art Implementations

With the foundational architecture and core design of the MCP now well established, attention naturally shifts toward its practical realization across a growing spectrum of real-world systems. This section explores the breadth of existing MCP implementations—both official and community-developed—and highlights how the protocol is being operationalized in diverse domains such as software development, automation, data science, communication, and enterprise tooling. From lightweight reference servers built to demonstrate protocol compliance, to full-scale enterprise deployments backed by major technology providers, the MCP ecosystem has matured rapidly, supported by a variety of SDKs and frameworks that abstract protocol complexities and accelerate integration. These implementations not only validate MCP's modular design and transport-agnostic messaging layer but also illustrate its adaptability in orchestrating large language model interactions with secure, context-rich, and tool-driven workflows. The following subsections provide a categorized overview of server types, client modules, community contributions, and SDKs—underscoring the protocol's evolution from a theoretical construct to a vibrant and production-ready interface layer for intelligent systems.

A. Ecosystem of MCP Servers and Clients

The MCP ecosystem has rapidly evolved into a diverse and expansive landscape, with hundreds of publicly available tools tailored to augment LLM capabilities. As curated on https://mcp.so[19], the largest existing registry of MCP endpoints, the breadth of application domains supported by MCP servers reflects the dynamic intersection of AI, automation, and digital infrastructure.

MCP servers span across multiple domains, each contributing unique functionalities to AI workflows and intelligent automation. At the forefront are the *Official Servers* (3 total), representing foundational implementations supported or endorsed by core MCP maintainers. These typically offer baseline capabilities and serve as references for best practices.

The *Research and Data* category dominates the registry with 1915 entries. This collection includes scientific computation tools, open data providers, NLP evaluators, and scholarly access platforms—catering to academic and exploratory AI use cases. Following closely are *Developer Tools* (4258), underscoring MCP's utility as a bridge for AI-to-code interactions, from CI/CD utilities to API testing environments.

TABLE IV
Categories of MCP Servers and Clients (as of April 8, 2025)

MCP Category	**MCP Count**
Servers	
Official Servers	3
Research And Data	1915
Cloud Platforms	149
Browser Automation	48
Databases	29
AI Chatbot	4
File Systems	19
OS Automation	30
Finance	33
Communication	45
Developer Tools	4258
Knowledge And Memory	27
Entertainment And Media	21
Calendar Management	11
Database	1
Location Services	15
Customer Data Platforms	2
Security	22
Monitoring	28
Virtualization	2
Cloud Storage	35
Clients	
MCP Clients	185

Cloud Platforms (149) and *Cloud Storage* (35) servers bring cloud-native operations into the fold, allowing LLMs to perform provisioning, querying, or management tasks across major providers. Similarly, *Browser Automation* (48) and *OS Automation* (30) empower agents to interact with software environments, executing scripts or mimicking user actions.

Other valuable contributions arise from *Databases* (29), *Database* (1), and *Monitoring* (28), where agents may search, log, or query data systems. In contrast, *File Systems* (19) and *Virtualization* (2) offer tools to manage local or virtual file hierarchies and containers.

Communication (45), *Calendar Management* (11), and *Location Services* (15) illustrate use cases grounded in human coordination, whether through messaging, scheduling, or geospatial awareness. A smaller but critical set focuses on *Security* (22) and *Customer Data Platforms* (2), where LLMs engage with privacy-sensitive environments.

Entertainment and creativity are also supported via *Entertainment and Media* (21), while financial domains see presence through *Finance* (33). *Knowledge and Memory* (27) services aid in persistent contextualization, memory chaining, or semantic search—fostering long-term interactions. Table IV presents MCP server and client counts as of April 8, 2025 where the total reaches to 7109.

B. Community MCP Servers

A lively ecosystem of community-developed MCP servers illustrates how the protocol seamlessly adapts to diverse domains. Although untested and lacking official endorsement,

[19]https://mcp.so/

they represent an expanding frontier of creativity, automation, and data integration. In music production, for instance, the ability to control Ableton Live[20] in real time or employ Blender[21] for 3D scene creation demonstrates how content-generation pipelines gain an AI-driven boost, while Arduino-based servers[22] enable robotics and hardware automation with minimal overhead. Travel and hospitality services become significantly more accessible through dedicated integrations for Airbnb[23], NS Travel Information[24], and FlightRadar24[25], simplifying itinerary planning and live trip monitoring.

Specialized data-driven servers also abound. Airtable[26], BigQuery[27][28], and ArangoDB[29] highlight flexible database interactions, enabling tasks such as schema inspection, query execution, and real-time analytics. Meanwhile, advanced financial and blockchain operations appear in Algorand[30], Bsc-mcp[31], or the EVM MCP Server[32], offering on-chain transaction management, DeFi analytics, or token security checks. Where large-scale development and operations are involved, Airflow[33], AWS[34], Azure DevOps[35], and Docker[36] expansions reinforce MCP's capacity to orchestrate cloud resources, automate CI/CD pipelines, and streamline resource management.

Collaboration and knowledge-sharing similarly find expression through servers for Atlassian[37] (Confluence/Jira), Discord[38], Gmail[39], or Microsoft Teams[40], which allow LLMs to read, post, and triage communications. Productivity gains emerge from integrated scheduling servers—Apple Calendar[41], Google Calendar[42], or Todoist[43]—offering direct ways to manage events and tasks via natural-language commands. At the same time, OS-level servers like Windows CLI[44], Terminal-Control[45], or iOS Simulator[46] illustrate how fundamental computing resources become scriptable through AI, broadening the range of possible automations.

Developers exploring AI-based browser automations and data extraction can leverage FireCrawl[47] or Rquest[48] for dynamic web scraping and external content retrieval, whereas code-assistant and code-executor expand into local or containerized code editing and execution. More specialized integrations, such as Pandoc[49] for document conversion, Xcodebuild[50] for iOS app compilation, or quick access to high-level GUIs in Reaper, show how any domain-specific workflow can be harnessed within an MCP-driven environment.

These community servers ultimately underscore the open, modular essence of MCP: from controlling creative tools to provisioning entire cloud environments, from bridging blockchains to orchestrating multi-database queries, they expand AI capabilities far beyond basic prompting. While their readiness and stability may vary, each highlights how enthusiastic contributors are merging domain-specific expertise with the universal interface of MCP. Individuals seeking to integrate these servers are urged to exercise caution, test them thoroughly, and tailor implementations to their own security requirements and operational contexts. Table IX shows existing list of community MCP servers.

C. *Official MCP Servers*

Officially maintained integrations illustrate the maturity and widespread reach of MCP within enterprise ecosystems. These are built and supported by the companies providing each platform, ensuring stability and rigorous production-level standards. Official integrations represent a steady evolution of MCP into enterprise-grade contexts, delivering specialized capabilities—from data indexing and user authentication to real-time analytics and financial transactions. Deployed and maintained by the respective organizations, these servers provide robust, tested functionalities that reflect the strategic significance of AI-driven automation in each platform's ecosystem.

In the design and user-interface sphere, 21st.dev Magic[51] offers a curated approach to crafting UI elements, while JetBrains[52] and IBM wxflows[53] empower development teams to integrate AI assistance seamlessly into IDEs and cross-platform workflows. Meanwhile, Apify[54], Firecrawl[55], and Hyperbrowser[56] highlight robust tools for large-scale web automation and data extraction, providing AI-driven insights from social media, e-commerce, and search engines.

In the realm of data, vector search, and observability, Chroma[57], ClickHouse[58], Meilisearch[59], Milvus[60], Mother-

[20]https://github.com/Simon-Kansara/ableton-live-mcp-server
[21]https://github.com/ahujasid/blender-mcp
[22]https://github.com/vishalmysore/choturobo
[23]https://github.com/openbnb-org/mcp-server-airbnb
[24]https://github.com/r-huijts/ns-mcp-server
[25]https://github.com/sunsetcoder/flightradar24-mcp-server
[26]https://github.com/domdomegg/airtable-mcp-server
[27]https://github.com/LucasHild/mcp-server-bigquery
[28]https://github.com/ergut/mcp-bigquery-server
[29]https://github.com/ravenwits/mcp-server-arangodb
[30]https://github.com/GoPlausible/algorand-mcp
[31]https://github.com/TermiX-official/bsc-mcp
[32]https://github.com/mcpdotdirect/evm-mcp-server
[33]https://github.com/yangkyeongmo/mcp-server-apache-airflow
[34]https://github.com/rishikavikondala/mcp-server-aws
[35]https://github.com/Vortiago/mcp-azure-devops
[36]https://github.com/ckreiling/mcp-server-docker
[37]https://github.com/sooperset/mcp-atlassian
[38]https://github.com/v-3/discordmcp
[39]https://github.com/GongRzhe/Gmail-MCP-Server
[40]https://github.com/InditexTech/mcp-teams-server
[41]https://github.com/Omar-v2/mcp-ical
[42]https://github.com/v-3/google-calendar
[43]https://github.com/abhiz123/todoist-mcp-server
[44]https://github.com/SimonB97/win-cli-mcp-server
[45]https://github.com/GongRzhe/terminal-controller-mcp
[46]https://github.com/InditexTech/mcp-server-simulator-ios-idb
[47]https://github.com/mendableai/firecrawl-mcp-server
[48]https://github.com/xxxbrian/mcp-rquest
[49]https://github.com/vivekVells/mcp-pandoc
[50]https://github.com/ShenghaiWang/xcodebuild
[51]https://github.com/21st-dev/magic-mcp
[52]https://github.com/JetBrains/mcp-jetbrains
[53]https://github.com/IBM/wxflows/tree/main/examples/mcp/javascript
[54]https://github.com/apify/actors-mcp-server
[55]https://github.com/mendableai/firecrawl-mcp-server
[56]https://github.com/hyperbrowserai/mcp
[57]https://github.com/privetin/chroma
[58]https://github.com/ClickHouse/mcp-clickhouse
[59]https://github.com/meilisearch/meilisearch-mcp
[60]https://github.com/zilliztech/mcp-server-milvus

Duck[61], Qdrant[62], Greptime[63], and StarRocks[64] stand out for their focus on querying, indexing, and processing vast datasets. Complementary capabilities in logs and event analysis are made possible by Axiom[65], Logfire[66], Comet Opik[67], and Raygun[68], offering real-time monitoring or historical forensics. Platforms like Graphlit, Unstructured, and Integration App facilitate either ingestion, transformation, or cross-application bridging of content from multiple sources, bolstering the underlying workflows managed by AI.

Cloud-based dev and infrastructure services also see official MCP support. Aiven[69], Cloudflare[70], Metoro[71], or Keboola[72] let agents manage provisioning, configurations, or data pipelines in a programmatic and standardized manner, while code execution sandboxes such as E2B[73] and ForeverVM[74] emphasize secure environment provisioning. Database specialists Apache IoTDB[75], Neo4j[76], SingleStore[77], OceanBase[78], and Neon[79] illustrate the consistent push for structured data access, offering official solutions that unify AI-based queries with well-governed data management. Storage solutions from Box[80] or Zapier's[81] broad integration network further expand the practical scope.

E-commerce and finance solutions play a central role in many enterprise workflows. Xero and Stripe deliver invoice, transaction, and refund management for business accounting. Tools like Adfin[82], Bankless[83], BICScan[84], Fewsats[85], Financial Datasets[86], and Octagon[87] target payment processing, asset insight, and general financial analysis. Meanwhile, domain-specific marketing and business optimization arrives through Audiense Insights[88], Fibery[89], and Riza[90]. Beyond that, solutions like eSignatures[91], Tavily[92], and PayPal[93] illustrate the breadth of official expansions, covering contract management, search, and payment processing.

An equally extensive group of platforms facilitate advanced data or document transformations. Agents can interact with APIMatic[94] for OpenAPI validation, CodeLogic[95] for software architecture, and Search1API[96] or ScreenshotOne[97] for wide-reaching search and web captures. Additional specialized capabilities arise in Lingo.dev[98] (multilingual localization), Lara Translate[99] (contextual translations), and Semgrep[100] (secure coding). By combining AI with official providers such as Mailgun[101] or Make[102], users orchestrate email dispatch and scenario automation in a thoroughly integrated environment. Table X shows existing list of official MCP servers.

D. Reference MCP Servers

Much like official and community implementations, a variety of reference servers illustrate how MCP works in practice. These examples focus on demonstrating specific protocol features or showcasing TypeScript and Python SDK capabilities, often emphasizing secure or read-only operations. Table V presents the comparison of selected reference MCP servers.

At the data-access level, the AWS KB Retrieval[103] server highlights retrieval-augmented generation by fetching AWS Knowledge Base content using the Bedrock Agent Runtime, whereas Brave Search extends web and local queries through advanced filtering and fallback logic. For creative tasks, EverArt[104] offers AI-driven image generation via multiple underlying models, and the minimalistic Everything server enumerates nearly all MCP features—from prompts to resource subscriptions—for thorough client testing. Similarly, Fetch enables chunked web content fetching and markdown conversion, reflecting a practical approach to large-page reading.

In the realm of system automation, Filesystem[105] demonstrates robust file operations, including editing and directory management with controlled access. A parallel focus on version control appears in Git[106], bridging LLMs with Git commands for staging, committing, and viewing diffs. More specialized repository interactions emerge in GitHub[107] and GitLab[108], each featuring nuanced tools for file commits, branch management, and PR or MR creation.

Developers prioritizing data queries have multiple read-only or partially restricted options, such as ostgreSQL[109] for

[61]https://github.com/motherduckdb/mcp-server-motherduck
[62]https://github.com/qdrant/mcp-server-qdrant/
[63]https://github.com/GreptimeTeam/greptimedb-mcp-server
[64]https://github.com/StarRocks/mcp-server-starrocks
[65]https://github.com/axiomhq/mcp-server-axiom
[66]https://github.com/pydantic/logfire-mcp
[67]https://github.com/comet-ml/opik-mcp
[68]https://github.com/MindscapeHQ/mcp-server-raygun
[69]https://github.com/Aiven-Open/mcp-aiven
[70]https://github.com/cloudflare/mcp-server-cloudflare
[71]https://github.com/metoro-io/metoro-mcp-server
[72]https://github.com/keboola/keboola-mcp-server
[73]https://github.com/e2b-dev/mcp-server
[74]https://github.com/jamsocket/forevervm/tree/main/javascript/mcp-server
[75]https://github.com/apache/iotdb-mcp-server
[76]https://github.com/neo4j-contrib/mcp-neo4j/
[77]https://github.com/singlestore-labs/mcp-server-singlestore
[78]https://github.com/yuanoOo/oceanbase_mcp_server
[79]https://github.com/neondatabase/mcp-server-neon
[80]https://github.com/box-community/mcp-server-box
[81]https://zapier.com/mcp
[82]https://github.com/Adfin-Engineering/mcp-server-adfin
[83]https://github.com/bankless/onchain-mcp
[84]https://github.com/ahnlabio/bicscan-mcp
[85]https://github.com/Fewsats/fewsats-mcp
[86]https://github.com/financial-datasets/mcp-server
[87]https://github.com/OctagonAI/octagon-mcp-server
[88]https://github.com/AudienseCo/mcp-audiense-insights
[89]https://github.com/Fibery-inc/fibery-mcp-server
[90]https://github.com/riza-io/riza-mcp
[91]https://github.com/esignaturescom/mcp-server-esignatures
[92]https://github.com/tavily-ai/tavily-mcp
[93]https://mcp.paypal.com/
[94]https://github.com/apimatic/apimatic-validator-mcp
[95]https://github.com/CodeLogicIncEngineering/codelogic-mcp-server
[96]https://github.com/fatwang2/search1api-mcp
[97]https://github.com/screenshotone/mcp/
[98]https://github.com/lingodotdev/lingo.dev/blob/main/mcp.md
[99]https://github.com/translated/lara-mcp
[100]https://github.com/semgrep/mcp
[101]https://github.com/mailgun/mailgun-mcp-server
[102]https://github.com/integromat/make-mcp-server
[103]https://github.com/modelcontextprotocol/servers/blob/main/src/aws-kb-retrieval-server
[104]https://github.com/modelcontextprotocol/servers/blob/main/src/everart
[105]https://github.com/modelcontextprotocol/servers/blob/main/src/filesystem
[106]https://github.com/modelcontextprotocol/servers/blob/main/src/git
[107]https://github.com/modelcontextprotocol/servers/blob/main/src/github
[108]https://github.com/modelcontextprotocol/servers/blob/main/src/gitlab
[109]https://github.com/modelcontextprotocol/servers/blob/main/src/postgres

TABLE V
COMPARISON OF REFERENCE MCP SERVERS

Name	Language Used	Key Features	License
AWS KB Retrieval	TypeScript	Retrieval-augmented generation from AWS Knowledge Base, flexible query results	MIT
Brave Search	TypeScript	Web/local search with filtering, pagination, fallback logic	MIT
EverArt	TypeScript	AI-driven image generation, multiple model options	Not mentioned
Everything	TypeScript	Demonstrates nearly all MCP features, includes resources, prompts, tools	Not mentioned
Fetch	Python	Chunked web content fetching, HTML-to-Markdown conversion, partial reading	MIT
Filesystem	TypeScript	Secure file operations, directory management, advanced editing	MIT
Git	TypeScript	Staging, committing, diff viewing, branch management	MIT
GitHub	TypeScript	Repo/file operations, auto branch creation, code/issue search	MIT
GitLab	TypeScript	Project management, file commits, merges, robust error handling	MIT
Google Drive	TypeScript	Google Drive file listing, reading, searching with auto-export	MIT
Google Maps	TypeScript	Location services, directions, place details, distance matrix	MIT
Memory	TypeScript	Knowledge graph-based memory, entity/relationship tracking	MIT
PostgreSQL	TypeScript	Read-only DB access, schema inspection, secure SQL queries	MIT
Puppeteer	TypeScript	Browser automation, screenshot capture, console logs	MIT
Redis	TypeScript	Key-value store operations, flexible pattern-based listing	MIT
Sentry	Python	Issue retrieval from Sentry.io, debug info	MIT
Sequential Thinking	TypeScript	Stepwise reflection, branching problem-solving approach	MIT
Slack	TypeScript	Channel management, messaging, reactions, workspace user retrieval	MIT
Sqlite	TypeScript	On-the-fly table creation, data analysis, insights memo	MIT
Time	Python	Timezone conversions, system time detection, IANA naming	MIT

structured SQL exploration and Sqlite[110] for on-the-fly table creation and business insights. A further subset addresses dynamic analysis and memory retention: Memory[111], with its knowledge-graph approach to entity and relation tracking, and Sequential Thinking[112], which structures reflective problem-solving through iterative steps. For orchestration tasks, Puppeteer[113] handles headless browser interactions, while the Redis server enables key-value store operations.

More domain-specific references exist for debugging and collaboration. Sentry[114] offers issue retrieval and analysis from error-monitoring pipelines, Slack[115] integrates channel-based messaging and reaction functionality, and the read-only nature of Time supplies robust time-zone calculations. Google Drive[116] and Google Maps[117] add specialized resource management, search, location, and route planning, echoing the diversity of real-world contexts addressed by MCP.

E. Existing MCP Frameworks

MCP frameworks enable seamless interaction between language models and external tools, resources, or prompts by standardizing communication over server and client infrastructures. These frameworks abstract protocol intricacies and offer diverse language implementations, empowering developers to build AI-integrated systems with minimal overhead. From lightweight TypeScript libraries to gateway-hosted Go solutions, MCP ecosystems support both rapid prototyping and scalable deployments. Table VI presents comparison of these frameworks.

[110]https://github.com/modelcontextprotocol/servers/blob/main/src/sqlite
[111]https://github.com/modelcontextprotocol/servers/blob/main/src/memory
[112]https://github.com/modelcontextprotocol/servers/blob/main/src/sequentialthinking
[113]https://github.com/modelcontextprotocol/servers/blob/main/src/puppeteer
[114]https://github.com/modelcontextprotocol/servers/blob/main/src/sentry
[115]https://github.com/modelcontextprotocol/servers/blob/main/src/slack
[116]https://github.com/modelcontextprotocol/servers/blob/main/src/gdrive
[117]https://github.com/modelcontextprotocol/servers/blob/main/src/google-maps

1) EasyMCP: EasyMCP exemplifies a minimalist yet expressive approach to MCP server development [65]. Written in TypeScript, it abstracts the complexities of the Model Context Protocol, allowing developers to focus on defining application logic rather than managing protocol intricacies. Its architecture emphasizes simplicity: the core API draws from the familiarity of ExpressJS, enabling a gentle learning curve for JavaScript and Node.js developers. For more advanced scenarios, an experimental decorators-based API auto-generates tool and prompt schemas, further reducing boilerplate and enabling high productivity. The framework also introduces a context object that unifies logging, progress reporting, and tool execution context under a single interface. While EasyMCP lacks advanced features such as SSE or real-time updates, its focus on developer experience and type safety makes it a suitable starting point for lightweight MCP servers. Its beta status implies ongoing evolution, making it ideal for early adopters seeking a straightforward yet type-safe entry into MCP development.

2) FastAPI to MCP: FastAPI to MCP offers a radically different proposition: it automatically translates FastAPI[118] endpoints into MCP-compliant tools with zero manual configuration. Implemented in Python, this auto-generator framework targets developers who already rely on FastAPI for building RESTful APIs. By enabling direct mounting of an MCP server within a FastAPI application, it eliminates the need for duplicate logic and accelerates the adoption of MCP in production-ready environments [66]. Schema and documentation preservation ensure that existing OpenAPI[119] models are retained, allowing seamless integration with tooling such as Swagger UI. This dynamic translation approach unlocks the possibility of reusing enterprise APIs as tools for large language models with minimal effort. Moreover, compatibility with both SSE and STDIO transports ensures broad interop-

[118]https://fastapi.tiangolo.com/
[119]https://www.openapis.org/

TABLE VI
COMPARISON OF MCP FRAMEWORKS

Name	Language Used	Category	Key Features	License
EasyMCP	TypeScript	Server	Express-like API, Decorators API, Context object, Type safety	MIT
FastAPI to MCP	Python	Server	Auto-generation, Schema preservation, Zero-config integration	MIT
FastMCP	TypeScript	Server	Session management, SSE, Authentication, Typed events	MIT
Foxy Contexts	Go	Server	Declarative DI, Transports (SSE, stdio), Dynamic resources	MIT
Higress MCP	Go (WASM)	Server	API Gateway integration, Security, Observability, WASM plugins	Apache-2.0
MCP-Framework	TypeScript	Server	CLI scaffolding, Auto-discovery, JWT auth, STDIO/SSE	Not mentioned
Quarkus MCP Server SDK	Java	Server	CDI Beans, SSE transport, Declarative annotations	Apache-2.0
Template MCP Server	TypeScript	Server	Dual transport support, TypeScript, Extensible structure	MIT
codemirror-mcp	TypeScript	Client	Autocomplete for @resources and /prompt, decorations, prompt building	Apache-2.0

erability with clients like Cursor[120] and Claude Desktop[121]. FastAPI to MCP is particularly valuable in enterprise contexts where integration speed, maintainability, and documentation fidelity are non-negotiable.

3) FastMCP: FastMCP targets developers who need more than just basic tooling—it is a robust, full-featured framework purpose-built for MCP server development in TypeScript. It supports key features such as session management, authentication, typed server events, and server-sent events (SSE), which together make it suitable for applications requiring stateful interactions and real-time capabilities. Tools and resources are defined using strong type schemas, and FastMCP provides auto-completion for prompt and resource arguments, improving developer ergonomics. A built-in CLI enables fast prototyping and debugging, while support for image content and progress reporting extends its applicability to multimedia workflows and user-driven interactions [67]. Unlike lightweight alternatives, FastMCP is well-suited for scalable AI integrations, where control, precision, and real-time feedback are essential. This framework positions itself as a foundational piece for building sophisticated AI-powered services.

4) Foxy Contexts: For developers who prefer static typing, concurrency, and dependency injection, Foxy Contexts presents a compelling option. Built in Go[122], this framework adopts a declarative programming model, utilizing Uber's FX[123] for life cycle and dependency injection (DI) management. Tools, resources, and prompts are registered declaratively, allowing clean separation of definitions and logic. The framework's design encourages modularity, enabling better testability and reuse of shared services like database connectors or cloud clients. Foxy Contexts supports multiple transport layers, including SSE and stdio, as well as experimental support for Streamable HTTP. A dedicated package for functional testing (*foxytest*) ensures developers can verify end-to-end behavior with clarity [68]. The framework's structured, DI-based approach makes it particularly effective in team environments where reproducibility, maintainability, and testability are critical requirements. It also lays the foundation for more advanced context-oriented server patterns by treating tools and prompts as injectable components.

[120]https://www.cursor.com/
[121]https://claude.ai/download
[122]https://go.dev/
[123]https://github.com/uber-go/fx

5) Higress MCP: Higress MCP merges infrastructure and protocol by embedding MCP server support within an Envoy-based API[124] gateway using WASM[125] plugins. This makes it more than just a development framework—it is a gateway-hosted runtime environment for MCP servers. Implemented primarily in Go, with WASM modules serving as extension points, Higress offers a unique deployment model: multiple MCP servers can be bundled into a single binary and exposed via HTTP interfaces [69]. The integration enables fine-grained control over access, observability, and performance metrics through gateway-level primitives like rate limiting, audit logs, and unified authentication. One standout feature is its ability to translate existing REST APIs into MCP tools using a declarative templating system powered by GJSON[126] and Sprig[127] functions. This allows rapid onboarding of legacy services into the LLM ecosystem with minimal custom code. Higress MCP is particularly suited for organizations seeking to combine cloud-native service meshes with AI orchestration via MCP.

6) MCP-Framework: MCP-Framework embodies the philosophy of convention over configuration. Built in TypeScript, it offers developers a structured scaffolding system through a command-line interface (*mcp create app*), enabling projects to be bootstrapped in minutes [70]. Its directory-based auto-discovery simplifies resource management, while built-in support for multiple transports (stdio and SSE) ensures flexible deployment paths. Security features such as JWT and API key authentication are available out-of-the-box, and tool/resource definitions can be extended with minimal overhead. While it does not currently disclose its license, the framework's opinionated structure promotes consistency, which is especially useful for teams managing multiple MCP services. MCP-Framework's elegance lies in its ability to abstract complexity while preserving extensibility—an ideal balance for teams building custom AI assistants or enterprise MCP services.

7) Quarkus MCP Server SDK: For Java developers entrenched in enterprise software, the Quarkus MCP Server SDK offers a natural entry point into MCP development [71]. Built atop the Quarkus runtime, it allows developers to define tools,

[124]https://gateway.envoyproxy.io/
[125]https://istio.io/latest/docs/reference/config/proxy_extensions/wasm-plugin/
[126]https://gjson.dev/
[127]http://masterminds.github.io/sprig/

prompts, and resources using annotated CDI beans[128]. This model provides both declarative and programmatic pathways for building MCP servers. Integration with LangChain4j[129] extends its utility to applications that require fine-grained AI orchestration. The SDK supports SSE transport by default and also accommodates stdio for local or embedded use cases. Java's maturity as a language, combined with Quarkus's optimized runtime and dependency injection capabilities, makes this SDK highly suited to regulated or high-availability environments, where type safety, modularity, and enterprise integration are paramount.

8) Template MCP Server: The Template MCP Server is a developer onboarding tool built around the FastMCP framework [72]. It provides a ready-to-run boilerplate that includes dual transport support (e.g. HTTP and stdio), TypeScript configuration, and a modular folder structure for organizing tools, prompts, and resources. This template reduces the initial setup burden by abstracting runtime logic and promoting rapid iteration. Its alignment with FastMCP ensures compatibility with the broader MCP ecosystem, while built-in scripts support Bun[130], npm[131], yarn[132], and pnpm[133] for flexible development workflows. It serves as a starter kit that can evolve into a full-fledged production server, bridging the gap between experimentation and deployment-readiness.

9) codemirror-mcp: codemirror-mcp is the only MCP client framework in this collection, designed to bring MCP capabilities into the browser-based development experience [73]. Implemented as an extension for CodeMirror, it enables features such as inline prompt completion, @*resource* mention resolution, and clickable decorations. These enhancements are especially powerful for AI-enhanced IDEs or code sandboxes where context and interaction are critical. The extension supports WebSocket[134]-based communication through the MCP SDK, providing real-time linkage to MCP servers. Its theme support and interaction handlers make it highly customizable, allowing seamless integration with existing editor environments. codemirror-mcp effectively bridges human and machine collaboration, turning the editing interface into a first-class participant in AI workflows.

F. Software Development Kits

The Software Development Kits (SDKs) for the MCP act as developer-friendly toolkits that simplify building and integrating MCP-compliant systems. Designed in various languages such as TypeScript, Python,Kotlin, C#, and Java, these SDKs reduce the burden of directly implementing the MCP specification by offering pre-built modules for core features like life cycle handling, protocol messaging, and capability negotiation. Whether you're building a client or a server, the SDK provides a structured foundation to help you focus on your application logic rather than the low-level details of JSON-RPC communication or session management.

On the server side, SDKs make it easy to define and expose functionalities like tools, prompts, and contextual resources—components that MCP servers use to interact with language models. They take care of routing messages, validating arguments, and responding to client requests, all while maintaining the strict isolation and permission boundaries MCP requires. Some SDKs even include project templates and command-line interfaces that can spin up a basic server setup within minutes, allowing quick experimentation and deployment. Client SDKs, on the other hand, are geared towards managing secure, session-based connections to one or more servers. They handle initial setup, exchange capabilities during negotiation, and support ongoing features like streaming updates, LLM sampling requests, and structured logging. These clients follow a one-to-one communication model with each server, ensuring clear boundaries and protocol compatibility. Advanced utilities like progress tracking, request cancellation, and ping mechanisms are also typically built-in. Another strength of MCP SDKs is their modular design. Developers can easily add or remove components without affecting the entire setup, allowing for flexible, minimal, and maintainable architectures. Moreover, SDKs accommodate different transport layers—such as standard input/output or Streamable HTTP—while preserving the integrity of the MCP session and message exchange.

Figure 12 presents the layered software architecture of the MCP SDK stack, illustrating how different components interact within a typical client-server setup. The diagram reflects the modular and transport-agnostic design principles of the Model Context Protocol, emphasizing reusability and compatibility across platforms. At the top layer, the MCP Client and MCP Server represent the application-facing interfaces responsible for handling LLM-related operations—such as prompt handling, tool execution, or context retrieval. These components maintain independent MCP Sessions, which encapsulate stateful interactions between the communicating parties, including capability negotiation and session continuity. Beneath these, the MCP Transport layers abstract the communication mechanisms, separating the transport concerns of both the client and the server. This modular design allows the underlying transport medium to be swapped or extended without altering session logic or application behavior. SSE, enabling streamable messaging over web-friendly protocols. On the Java SDK side, the diagram further delineates HTTP transport options via specific frameworks such as Java HttpClient [74], Spring WebFlux [75], and Spring WebMVC [76]—each supporting different concurrency models and deployment styles within Java ecosystems.

To support developers in implementing the MCP across diverse environments, several official SDKs have been released. These SDKs abstract the low-level operations of MCP, making it easier to create servers and clients that provide structured context to language models while managing life cycle events, messaging, and communication over standard transports like stdio and Streamable HTTP. Below is an overview of the key language-specific SDKs currently available.

[128]https://docs.oracle.com/javaee/6/tutorial/doc/gjfzi.html
[129]https://docs.langchain4j.dev/
[130]https://bun.sh/
[131]https://www.npmjs.com/
[132]https://yarnpkg.com/
[133]https://pnpm.io/
[134]https://websocket.org/

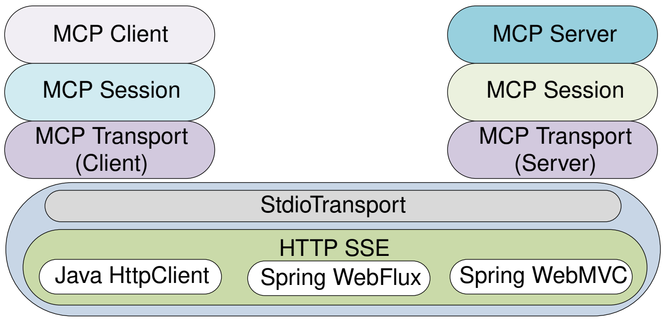

Fig. 12. MCP stack for SDK.

- Python SDK
  The Python SDK enables developers to quickly build both MCP clients and servers using Python [77]. It offers full support for the protocol, including session management, capability negotiation, and message routing. With this SDK, developers can expose prompts, tools, and contextual resources, and interact with any compliant MCP server. The SDK is easy to install using pip and comes bundled with working examples like an echo server and SQLite-based context exploration. It is ideal for research prototypes and production-ready AI applications.
- TypeScript SDK
  The TypeScript SDK brings MCP capabilities to JavaScript and TypeScript environments, supporting both browser-based and Node.js applications. It includes utilities to create servers that offer tools, resources, and prompt templates, as well as clients that can connect to any MCP server [78]. The SDK supports transport layers such as stdio and SSE out-of-the-box and provides a clean, modular codebase with examples for quick adoption. It is particularly suited for front-end extensions and cloud-native LLM integrations.
- Java SDK
  This SDK is tailored for developers working in Java-based ecosystems, including Spring Boot environments [79]. It supports building fully MCP-compliant clients and servers with integration hooks for REST and AI services. Designed with strong typing and asynchronous communication in mind, the Java SDK is a good fit for enterprise-grade applications that require stability and interoperability with existing backend systems.
- Kotlin SDK
  Optimized for Kotlin's expressive syntax and modern architecture, this SDK allows the development of lightweight MCP clients and servers in Kotlin [80]. It offers flexible support for transports such as stdio, SSE, and WebSocket, and is compatible with multi-platform projects. The Kotlin SDK is suitable for developers looking to implement MCP in mobile, desktop, or cross-platform applications while maintaining compliance with the core protocol.
- C# SDK
  The C# SDK brings MCP functionality to the .NET ecosystem, enabling integration with desktop, web, and service-based applications [81]. It supports full lifecycle handling, message parsing, and standard transport layers. Although still in an early release stage, it offers a foundation for building robust MCP applications within Microsoft technologies. The SDK can be easily installed via NuGet[135], and developers are encouraged to track future updates as the SDK evolves. Table VII presents comparison of SDKs.

G. *Prospective Cross-Domain Use Case Deployments*

The MCP introduces a transformative middleware approach that supports scalable, secure, and contextually adaptive interactions between LLMs and domain-specific systems [82], [83]. With its structured architecture and support for tool invocation, resource access, and session-based communication, MCP offers a generalized solution for operationalizing AI agents across multiple sectors [84], [85]. This section delineates a spectrum of potential deployment scenarios across vertical domains where MCP can play a pivotal role in enabling intelligent automation and task orchestration.

1) Conversational Database Interfaces : MCP can enable LLMs to serve as natural language interfaces to structured data systems, thereby reducing dependency on technical query languages [86], [87]. When integrated with database servers via MCP-compliant endpoints, LLMs can interpret user instructions, translate them into SQL queries, and return results in human-readable summaries or visualizations. This configuration facilitates real-time, read-only access to enterprise datasets, supporting use cases such as operational reporting, decision analytics, and key performance indicator (KPI) dashboards without requiring database access privileges for end-users. Through template-based prompt execution and schema-aware query generation, such agents reduce friction in business intelligence workflows and democratize data access across organizational roles.

2) AI-Augmented Software Development: The integration of MCP within software development tools may introduce a new paradigm in developer productivity [88]. By connecting LLM agents with IDEs, version control systems, and build automation tools via MCP-compliant interfaces, contextual tasks—such as refactoring suggestions, dependency graph analysis, or pre-commit validations—can be delegated to intelligent agents [89]. This approach enables LLMs to function as collaborative programming assistants that leverage prompt templates to analyze source code, inspect project structure, and respond to code quality issues [90], [91]. Extensions toward continuous integration and continuous delivery/continuous deployment (CI/CD) workflows further allow these agents to monitor build health, interpret logs, and recommend recovery actions, thereby enhancing pipeline resilience and debugging efficiency.

3) Enterprise Workflow Automation: In modern enterprise ecosystems characterized by distributed software-as-a-service (SaaS) tools, MCP may offer a unifying communication layer that allows LLMs to coordinate operations across disparate

[135]https://www.nuget.org/

TABLE VII
COMPARISON OF OFFICIAL MCP SDKS ACROSS LANGUAGES

SDK	Language	Key Capabilities	Target Use Case	Current Limitations
Python SDK	Python	Full MCP spec support; build servers and clients; expose resources, prompts, tools; stdio and SSE transport	Ideal for prototyping, academic research, and quick server deployment with minimal setup	Early-stage projects may need additional maturity and optimization for large-scale production
TypeScript SDK	TypeScript/ JavaScript	Supports both client and server roles; transport via stdio and SSE; CLI tools and examples included	Suitable for web-based LLM integrations, browser extensions, and Node.js applications	May require familiarity with async programming and transport event handling
Java SDK	Java	Spring Boot integration; strong typing and async support; suitable for enterprise services	Enterprise backend services, microservices, and AI pipeline integrations	Limited high-level abstractions; still evolving for wider REST interop
Kotlin SDK	Kotlin	Lightweight SDK with multiplatform support; stdio, SSE, WebSocket transport; idiomatic Kotlin APIs	Mobile, desktop, or cross-platform apps that use LLM contexts	Ecosystem less mature than Java counterpart; smaller community support
C# SDK	C# (.NET)	Preliminary support for MCP lifecycle and messaging; integrates with .NET applications; stdio transport available	.NET desktop apps, service-layer integration, or web tools	Preview release; breaking changes expected; limited documentation as of now

platforms [92]. By defining standardized tool schemas and action prompts, MCP agents can synchronize operations such as file sharing, calendar updates, CRM record modification, and internal messaging. This functionality supports high-level automation where user instructions result in chained API calls across services like Google Workspace[136], Slack[137], Notion[138], or Jira[139]. The outcome is a cohesive, agent-driven workflow that eliminates the need for manual switching between interfaces, enhances traceability, and ensures consistent operational logic across digital touchpoints.

4) Agentic IT Support: The MCP framework can provide an effective mechanism for interfacing LLMs with cloud infrastructure, system diagnostic tools, and container orchestration platforms. By exposing command-line utilities, monitoring dashboards, and admin scripts as callable MCP tools, IT operations can be augmented with agents capable of performing system introspection, log analysis, and automated service management. Agents may retrieve telemetry data, diagnose anomalies, or perform remediation actions such as restarting services or adjusting resource allocation [93]. These capabilities enable autonomous system maintenance, event-driven responses, and structured reporting, reducing operational latency and supporting more resilient IT service delivery [94].

5) Context-Aware Technical Compliance: Maintaining synchronized technical documentation across dynamic software systems poses a persistent challenge, especially in regulated industries. MCP can enable LLMs to access source files, configuration repositories, and documentation endpoints through a uniform protocol, enabling context-sensitive documentation synthesis [95], [96]. Agents can extract function signatures, parse changelogs, compare version history, and produce structured output for integration into compliance artifacts or developer portals. By coupling documentation generation with version control and structured prompt workflows, MCP facilitates continuous documentation updates that reflect real-time system state, contributing to traceability, auditability, and regulatory conformity [97].

6) AI-Assisted ETL Pipelines: MCP-equipped LLMs can streamline ETL pipeline design and execution by interfacing with file storage, schema registries, transformation engines, and orchestration platforms. By querying metadata, validating data types, and composing transformation logic, agents can assist in constructing robust data ingestion and processing workflows [98]. MCP tools for schema inspection, file parsing, and batch processing allow LLMs to act as data wrangling assistants that validate integrity constraints, automate joins and mappings, and simulate data flows [99]. This approach reduces the technical entry barrier for data engineers while enhancing reproducibility and pipeline documentation [100].

7) Healthcare: In healthcare settings, MCP may enable intelligent agents to interact with heterogeneous clinical systems such as EHRs, lab management platforms, imaging repositories, and scheduling interfaces [101]. Agents configured via MCP tools can retrieve structured patient data, identify diagnostic trends, and present clinical summaries that support triage, diagnosis, and treatment planning. Context-aware prompt templates allow agents to query temporal patient records or detect anomalies in lab reports, enabling continuous support in inpatient care and outpatient workflows. By maintaining strict access control and data isolation, MCP supports secure deployment in health IT environments requiring Health Insurance Portability and Accountability Act (HIPAA) or General Data Protection Regulation (GDPR) compliance.

8) Financial Services: The financial domain presents extensive opportunities for MCP-based agents to automate analytical, compliance, and advisory tasks. Through structured connections to transaction logs, anti-fraud engines, and market data providers, agents can conduct anomaly detection, generate regulatory reports, or provide risk assessment summaries [102]. MCP servers can expose critical functionalities such as fund flow tracing, compliance checklist matching, or rule-based portfolio audits, all accessible through prompt-guided LLM workflows. Such integrations streamline operational audits, support financial forecasting, and enable real-time com-

[136]https://workspace.google.com/
[137]https://slack.com/
[138]https://www.notion.com/
[139]https://jira.atlassian.com/

pliance verification with minimal human oversight.

9) Education: MCP can serve as the infrastructural backbone for intelligent tutoring systems, integrating LLMs with LMS platforms, assessment databases, and content repositories. Through structured tools and session-based prompts, agents can retrieve academic records, recommend remedial content, and generate adaptive learning modules [103]. Instructors may utilize these agents to automate attendance, generate grading rubrics, or design quizzes aligned with student performance metrics. By enabling consistent educational interventions based on real-time analytics, MCP-based deployments can facilitate scalable personalized learning environments and reduce manual administrative workload for educators.

10) Manufacturing: MCP-based agents in manufacturing can interact with Industrial internet of things (IoT) devices, supervisory control and data acquisition (SCADA) systems, maintenance logs, and scheduling tools to monitor and control production environments. Agents can retrieve sensor readings, correlate them with operational thresholds, and initiate mitigation protocols in case of anomalies. Integration with MES platforms through MCP facilitates workload optimization, quality control, and supply chain visibility. The modular nature of MCP supports scalable deployment in smart factory architectures where contextual task delegation, safety enforcement, and traceable interactions are critical for efficient and safe manufacturing [104].

11) Retail and E-Commerce: Retail ecosystems encompass CRM platforms, inventory systems, point-of-sale terminals, and logistics engines [105]. By deploying MCP agents that interface with these modules, LLMs can manage customer queries, monitor supply chain fluctuations, and generate predictive analytics for sales optimization. MCP tools for recommendation generation, product metadata analysis, and order status retrieval allow intelligent assistants to deliver personalized shopping experiences. Concurrently, backend agents can automate inventory forecasting, restocking, and fulfillment coordination, fostering a tightly integrated retail automation stack.

12) Legal Sector: Legal practice involves extensive document review, clause comparison, and compliance tracking, all of which benefit from structured AI assistance. MCP-based agents can access legal repositories, version-controlled contract archives, and regulatory documents, allowing prompt-guided analysis and recommendation generation. By defining tools for clause extraction, terminology validation, and precedent search, agents can assist lawyers in contract drafting, due diligence, and legal research with contextual accuracy [106]. These deployments improve throughput and consistency while preserving confidentiality and control over sensitive information.

13) Governance: In the public sector, MCP offers a framework for deploying citizen-facing AI agents that connect to forms portals, document archives, spatial services, and notification systems [107]. Agents may guide users through administrative procedures, verify eligibility criteria, or retrieve legal documents from regional databases. Government workflows—such as benefit disbursement, license renewal, or civic feedback processing—can be streamlined by MCP agents managing standardized interactions, facilitating accessibility and transparency. This contributes to the realization of digital governance goals centered on efficiency, accountability, and user-centric service delivery.

14) Cybersecurity: In the realm of cybersecurity, MCP-enabled agents can integrate logs from security information and event management (SIEM) systems, firewalls, intrusion detection systems, and threat intelligence feeds to provide a unified view of enterprise security [108], [109]. These agents not only identify anomalies like privilege escalation or lateral movement across the network, but can also cross-reference active threats with real-time threat databases. Over time, they can build behavioral baselines and automatically suggest or initiate countermeasures, dramatically accelerating incident response cycles and reducing dependency on manual security audits.

15) Smart Cities: City planners and civic authorities stand to benefit significantly from MCP agents capable of interfacing with GIS layers, infrastructure sensors, public transport APIs, and zoning databases. These agents can generate insights on traffic congestion patterns, identify areas lacking amenities, or flag hazardous construction zones. More holistically, they could simulate urban growth scenarios, advise on equitable distribution of green spaces, or even support policy planning with data-backed urban equity indices [110].

16) Tourism and Hospitality: Travel agents augmented by MCP can tap into flight APIs, hotel booking systems, local activity aggregators, and even visa requirement databases [111], [112]. They can provide end-to-end itinerary suggestions personalized by traveler preferences, time constraints, and budget, adjusting plans dynamically based on delays, health advisories, or natural calamities. At hospitality venues, such agents could manage check-in procedures, predict peak dining hours, or adapt room services in real time, elevating the overall guest experience.

17) Agriculture: In agricultural domains, MCP can connect LLMs to field sensors, weather APIs, crop health imaging platforms, and agronomic databases. Agents may perform soil quality assessment, irrigation schedule optimization, and yield forecasting by synthesizing multimodal inputs. The MCP protocol facilitates interoperability between diverse rural edge devices and centralized advisory platforms, allowing real-time support in resource-constrained environments. These capabilities support the transition toward precision agriculture, improving sustainability, productivity, and climate resilience for smallholder and industrial-scale farms alike.

18) Recruitment Automation: Within HR departments, MCP can support intelligent automation across recruitment, evaluation, and employee engagement workflows. Agents can scan large volumes of resumes to shortlist candidates based on nuanced criteria, such as inferred cultural fit or learning agility, not just keyword matches. Moreover, by accessing performance dashboards, leave records, and internal surveys, these agents can help HR managers preempt attrition, improve workplace morale, and personalize training recommendations, thereby streamlining both strategic and operational tasks.

19) Transportation: In transportation networks and supply chain ecosystems, MCP agents can facilitate seamless coordination between vehicle telematics, warehouse inventory systems, and delivery platforms. They can autonomously reroute shipments based on weather conditions, fuel prices, or delivery urgency, while proactively alerting logistics teams about delays or bottlenecks. Over time, such agents may help build predictive maintenance models for fleet vehicles or suggest optimal load balancing strategies based on historical and real-time data.

20) Media and Publishing: MCP-integrated agents in publishing environments can help editorial teams automate content curation, headline generation, and formatting for different channels. These agents can scan through large corpora of draft content to identify repetition, flag bias, or suggest stylistic enhancements aligned with house tone. For platforms dealing with user-generated content, MCP agents can enforce community guidelines through real-time moderation, multilingual filtering, and even sentiment analysis to promote respectful discourse.

21) Disaster Response: During crisis events such as floods, earthquakes, or wildfires, MCP can coordinate agents that unify aerial imagery, local emergency call data, supply inventories, and GPS tracking from rescue teams. These agents can offer near-instantaneous impact assessments, prioritize high-risk zones, and assist command centers in allocating medical aid, food, or shelter. Furthermore, they can generate dynamic situational reports that evolve with the event, helping stakeholders make informed decisions faster and more accurately.

H. Differentiating MCP from Traditional APIs and A2A

Traditional APIs are grounded in a stateless client-server communication model where each request is executed independently, lacking intrinsic memory of previous exchanges [114], [115]. These APIs, typically RESTful and reliant on HTTP/HTTPS protocols, are effective for structured, one-shot data retrieval or submission. However, they falter in agentic contexts that demand contextual awareness, multi-turn reasoning, or tool orchestration [116], [117]. To simulate continuity, developers often rely on workarounds like cookies or repeated token-based authentication, which increases design complexity. As agent-based systems evolve with LLMs at their core, this absence of native context retention and runtime adaptability proves a limiting factor [118]. Traditional APIs do not support on-the-fly negotiation of capabilities or seamless coordination of tool access across modular services, restricting the flexibility needed in complex reasoning pipelines.

The Model Context Protocol addresses these constraints by implementing a structured, session-oriented communication mechanism between AI agents, external tools, and contextual data sources. Built upon a host-client-server paradigm, MCP enables persistent state tracking through session identifiers, allowing continuity across LLM invocations and tool interactions. The Host acts as an orchestrator, aggregating tool definitions, user inputs, conversation history, and resource metadata, then packaging this information into standardized JSON-RPC 2.0 messages for the model. Tool invocation, sampling requests, and result integration are all governed within the same message protocol, ensuring a coherent conversational flow. Furthermore, MCP supports dynamic capability registration and negotiation at session initialization, making each interaction context-aware and extensible. As such, MCP transforms the application-model interface into an intelligent middleware layer capable of maintaining multi-step dialogue, invoking tools with precision, and preserving context for future operations. It also enforces strict access control boundaries and integrates with OAuth 2.1-based authentication, improving auditability and security compliance.

In contrast, Google's Agent-to-Agent (A2A) protocol is designed to enable communication among independent agents, often developed by different vendors or deployed in distributed environments [119], [120]. A2A introduces a standardized mechanism for capability discovery, secure inter-agent messaging, and cooperative task management [121], [122]. Through structured constructs such as Agent Cards, Tasks, Parts, and Artifacts, A2A formalizes how agents advertise their services, coordinate actions, and share outcomes [123], [124]. Built on familiar internet standards like JSON-RPC over HTTP(S) and SSE, A2A supports both short-lived requests and long-running processes with real-time status updates and asynchronous notifications. The protocol is also modality-agnostic, allowing agents to exchange not just text, but also multimedia payloads and user interface elements.

A key distinction is that while MCP enhances the internal decision-making loop of a single agent (especially LLM-powered), A2A provides the infrastructure for agent-to-agent federation across organizational or architectural boundaries. For example, a Personal Assistant Agent may use MCP internally to retrieve a user's calendar events via a tool, but utilize A2A externally to communicate with a Restaurant Booking Agent for reservations. In this layered interaction model, MCP enables intra-agent cognition and capability usage, while A2A handles inter-agent delegation and collaboration. Thus, these protocols are inherently complementary, jointly forming a foundational stack for scalable, interoperable, and intelligent multi-agent systems. Table VIII presents the comparison of MCP with traditional API and A2A.

V. Challenges in MCP Deployment

While the MCP provides a structured foundation for enabling context-rich interactions between LLMs and external tools, its implementation is accompanied by a host of practical and systemic challenges. The protocol's layered architecture, real-time communication demands, and cross-platform integration goals introduce substantial complexity at multiple levels. From negotiating capabilities across heterogeneous endpoints to securing sensitive data streams, each deployment context brings distinct risks and constraints. Additionally, the need to ensure backward compatibility, maintain high performance under load, and uphold resilience in unpredictable environments adds to the operational overhead. This section outlines the principal obstacles encountered during MCP adoption, highlighting the technical depth required to translate protocol-level innovation into production-grade reliability.

A. Complex Capability Negotiation

MCP's capability negotiation can be compared to a highly intricate distributed consensus mechanism where each

TABLE VIII
COMPARATIVE OVERVIEW: A2A VS MCP VS TRADITIONAL API

Aspect	Traditional API	Model Context Protocol	Agent-to-Agent Protocol
Primary Focus	Data access and manipulation via end-points	Structured integration of tools, prompts, and context with LLMs	Autonomous agent collaboration and negotiation
Architecture	Stateless client-server	Stateful host-client-server	Decentralized peer-like agents with structured task exchange
Session Support	No native session memory; relies on external tokens or cookies	Maintains session context through IDs and orchestrator logic	Tracks task lifecycle, status, and agent memory via artifacts
Communication Protocol	REST, GraphQL over HTTP/HTTPS	JSON-RPC 2.0, stdio, SSE	JSON-RPC 2.0 over HTTP(S), SSE, Push Notifications
Capability Discovery	Manual documentation and fixed schemas	Dynamic tool/resource registration during handshake	Agent Cards expose discoverable skills and service metadata
Modality Support	Typically limited to structured text or JSON/XML	Supports prompts, binary resources, and tool metadata	Supports text, images, video, web forms, and complex UIs
Security Model	OAuth 2.0, API keys	OAuth 2.1, PKCE, scoped tool access control	OAuth 2.0, OpenID Connect, multi-tenant security policies
Interoperability	Vendor-specific, minimal cross-framework support	Standardized across MCP-compatible clients and servers	Cross-vendor agent ecosystem using shared message schema
Ideal Use Cases	Legacy systems, CRUD operations, structured business logic	LLM-enhanced assistants, developer tools, embedded reasoning systems	Multi-agent workflows, enterprise automation, dynamic service orchestration
Extensibility	Limited by endpoint design and predefined contracts	Highly modular; supports dynamic tool/resource expansion	Extensible via custom agent definitions and task schemas
Latency Tolerance	Suited for synchronous requests with fast response expectations	Supports both synchronous and semi-streamed contextual workflows	Optimized for asynchronous task handling with status updates
Tool Invocation	Predefined API operations without model-based planning	Invoked by LLM based on prompt intent and structured input	Delegated to collaborating agents based on capabilities

node—in our case, clients, servers, and hosts—must exchange detailed capability descriptors via a multi-layered JSON-RPC handshake [125]. This process not only involves binary flags for basic functionalities but also the negotiation of nested objects describing sub-capabilities, such as the granularity of dynamic sampling (e.g. token budgets, time constraints, sampling temperature ranges) and asynchronous notifications (e.g. progress, cancellation, error reporting). Each of these elements must be synchronized in near real time, and the negotiation logic must account for potential race conditions, message reordering, and even packet loss in networked environments. The complexity is compounded when integrating with LLM systems that require deterministic behavior for reproducibility; even a slight misinterpretation of the negotiation state can result in fallback modes that lack the optimized features essential for high-quality inference. Advanced state machines, complete with timeout handlers, rollback mechanisms, and conflict-resolution strategies (e.g. vector clocks or logical timestamps), become necessary to ensure that every node maintains a consistent view of the protocol state.

B. Backward Compatibility

The evolution of MCP introduces a multi-dimensional challenge: not only must new features be integrated seamlessly, but older systems must be able to interpret new messages without loss of functionality [126]. This often requires an overlay of version negotiation protocols that utilize semantic versioning combined with explicit capability downgrades. For instance, when a client supporting MCP version *later date* interacts with a legacy server operating on version *earlier date*, both entities must negotiate a common subset of features—possibly using version wrappers or compatibility shims—to ensure that no critical data is misinterpreted. Techniques such as API gateway patterns and protocol translators can facilitate this process, while extensive regression testing and formal verification methods (e.g. model checking) help guarantee that changes in the protocol do not inadvertently introduce bugs. In the context of LLM applications, even minor discrepancies in context formatting or prompt handling could dramatically affect output quality, necessitating the use of robust fallback logic and error-correction codes to ensure consistency.

C. Security

MCP's capacity to deliver an extensive range of context—from file system hierarchies and dynamic logs to intricate prompt templates—introduces significant security risks that must be mitigated through a rigorous, multi-layered approach [127]. In addition to employing industry-standard encryption protocols such as Transport Layer Security (TLS) 1.3, future implementations might incorporate post-quantum cryptographic algorithms (e.g., lattice-based cryptography) to ensure long-term data integrity. Sandboxing techniques at both the process and network level—using technologies like Docker containers or hardware isolation via Trusted Execution Environments (TEEs)—can enforce strict data boundaries, ensuring that sensitive information remains compartmentalized. Moreover, advanced intrusion detection systems (IDS) leveraging machine learning models (e.g. deep autoencoders) could continuously analyze context flows for anomalies, while secure logging and immutable audit trails (i.e. potentially powered by distributed ledger technology) provide accountability and forensic capabilities. These measures, while computationally intensive, are essential in environments where LLMs might process proprietary intellectual property or personal data [128].

D. Authorization and Token Management

Implementing an OAuth 2.1 framework with PKCE within MCP introduces a series of highly technical challenges centered around efficient token management. Each request must be accompanied by a Bearer token, whose lifecycle is managed using cryptographically secure methods, such as HMAC-based signatures or public-key infrastructures (PKI). To reduce overhead, tokens may be structured as JSON Web Tokens (JWTs)[140] that embed claims and expiration timestamps, allowing for stateless verification. However, the frequent generation and validation of these tokens in high-throughput systems require optimized caching layers, potentially using in-memory data stores like Redis, with replication and sharding strategies to ensure horizontal scalability. Advanced techniques such as short-lived tokens with overlapping grace periods and asynchronous token validation routines can help mitigate the performance impact [129]. Furthermore, mechanisms such as token introspection endpoints and revocation lists must be implemented securely, employing robust error-handling and fallback logic to ensure that any compromise in token integrity does not disrupt the overall system functionality [130].

E. Transport Layer Latency

MCP's dual transport strategies—namely, local stdio and remote Streamable HTTP—each introduce distinct latency challenges [131]. With stdio, the performance hinges on efficient inter-process communication (IPC) where message framing, newline delimitation, and non-blocking I/O are paramount. In the case of HTTP-based transport, additional latency arises from establishing TLS connections, negotiating HTTP headers, and managing SSE streams for asynchronous notifications. Advanced techniques such as persistent HTTP connections, connection pooling, and using HTTP/2 or HTTP/3 with multiplexing capabilities can reduce overhead. Further optimization might involve moving from text-based JSON to binary serialization formats like MessagePack[141], Protocol Buffers[142], or FlatBuffers[143], which reduce both payload size and parsing time. Leveraging hardware accelerators (e.g. dedicated NICs or GPU-based encryption engines) and optimizing the network stack via kernel bypass techniques (e.g. DPDK[144]) can also be explored to meet the stringent latency requirements of interactive LLM applications.

F. Context Aggregation

MCP's strength lies in its ability to aggregate a vast spectrum of context from disparate sources. However, the aggregation process can lead to information overload if not properly filtered. To tackle this, advanced relevance filtering algorithms must be employed. These might include vector space models utilizing cosine similarity, attention mechanisms drawn from transformer architectures, or clustering algorithms (e.g. k-means[145] or DBSCAN[146]) to group semantically similar data. The objective is to distill raw context into a refined set of high-value inputs that are then fed into the LLM. Additionally, techniques such as context window optimization—where only the most recent or statistically significant context is maintained—and dynamic summarization methods by using abstractive or extractive summarization can be employed to strike a balance between data richness and computational efficiency. This filtering process is critical, as even minor perturbations in the context can lead to significant variances in LLM output quality.

G. Error Handling

A robust error-handling framework in MCP must address a myriad of failure scenarios, from network interruptions and data corruption to protocol desynchronizations. Resilience can be built into the system by incorporating design patterns such as circuit breakers, bulkheads, and failover clusters. Techniques such as automated retries with exponential backoff and jitter, combined with idempotent operation design, ensure that transient errors do not propagate and cause systemic failures. Additionally, distributed tracing systems (e.g. OpenTelemetry[147] or Jaeger[148]) can be integrated to provide detailed telemetry data, allowing engineers to pinpoint bottlenecks or anomalies in the system. Coupling these techniques with machine learning–driven anomaly detection models can further enhance resilience by predicting and preempting failure modes before they impact the overall service quality.

H. Interoperability

Achieving seamless interoperability across various implementations of MCP necessitates strict compliance with standardized protocols, robust middleware solutions, and comprehensive test suites. Heterogeneity in JSON parsers, encryption libraries, and I/O operations across platforms (e.g. Windows, Linux, macOS, or embedded systems) can lead to subtle discrepancies that affect performance and correctness. The use of containerization (e.g. Docker[149]) and virtualization can help standardize the deployment environment, while API abstraction layers can homogenize the communication between disparate components. Moreover, middleware frameworks that provide protocol normalization—translating data formats, character encodings, and even error semantics—are essential to bridge these differences. Interoperability testing should include fuzzing, stress testing, and automated regression tests to ensure that all implementations adhere to a unified specification, minimizing integration issues and operational risks.

I. Scalability

As MCP is deployed in high-concurrency environments supporting thousands or millions of LLM sessions, scalability becomes a critical factor. The protocol's stateful nature, which involves managing persistent sessions with context data, token validations, and dynamic capability negotiations, demands a

[140]https://jwt.io/
[141]https://msgpack.org/index.html
[142]https://protobuf.dev/
[143]https://flatbuffers.dev/
[144]https://doc.dpdk.org/guides-21.11/prog_guide/kernel_nic_interface.html
[145]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[146]https://github.com/mhahsler/dbscan
[147]https://opentelemetry.io/
[148]https://www.jaegertracing.io/
[149]https://www.docker.com/

high degree of parallelism and resource elasticity. Techniques such as microservices architectures, reactive programming models by using frameworks like Akka[150] or Node.js[151], and event-driven processing can decouple complex tasks and enable horizontal scaling. Load balancing algorithms that dynamically distribute work based on current system metrics (e.g. using consistent hashing or least-connections strategies) are essential to prevent bottlenecks. Moreover, the deployment of container orchestration platforms (e.g. Kubernetes[152]) allows for the auto-scaling of services in response to real-time demand, ensuring that system performance remains robust under peak loads while maintaining low latency and high throughput.

J. Deployment Complexity

The MCP ecosystem is characterized by a wide array of implementations developed in different programming languages, deployed across diverse environments, and adhering to various subsets of the protocol. This diversity can result in ecosystem fragmentation, where interoperability issues and inconsistent feature support become prevalent. Mitigating this requires the establishment of comprehensive development toolkits, common API contracts, and standardized middleware that abstract away underlying differences. CI/CD pipelines, along with automated compliance and interoperability test suites, are necessary to ensure that each implementation remains aligned with the evolving MCP specification. In parallel, fostering a collaborative community through centralized repositories, open-source governance, and industry consortia can help harmonize efforts, reduce fragmentation, and facilitate smoother deployments across the entire ecosystem.

K. Command Injection via Prompt Semantics

MCP's strength—parameterised prompt templates that are filled in at run-time—is also a potent attack surface: if the free-form arguments supplied by users, scripts, or upstream services are inserted into the template without rigorous normalisation and context-aware escaping, an adversary can hide control sequences, homoglyph look-alikes, or doubly-encoded delimiters that survive each transformation layer and ultimately reach the language model as privileged meta-commands; the model, following its reasoning chain, then invokes a tool or issues a system-level instruction that the human never intended, silently closing support tickets, deleting files, or exfiltrating data. Because the entire flow occurs inside otherwise trusted channels, traditional Access Control Lists (ACLs) or API gateways seldom notice the abuse, which makes command-injection defects especially insidious; robust defence therefore demands a combined strategy of strict JSON-Schema validation on every argument field, uniform escaping performed *after* JSON parsing but *before* template rendering, and a lightweight "prompt-firewall" model that scans the fully rendered text for tell-tale patterns such as brace-wrapped tool calls or role-switch directives and pauses execution until a human operator approves the outlier.

[150]https://akka.io/
[151]https://nodejs.org/en
[152]https://kubernetes.io/

L. Tool Poisoning

Because every MCP server publishes a JSON manifest describing its tools, a malicious or later-compromised package can masquerade behind a benign schema while shipping code that performs covert side effects, thereby transforming supply-chain trust into an execution back-door: a community server first gains reputation with a harmless "row-count" tool, then ships an update whose implementation silently uploads the CSV to an attacker's bucket before returning the count, and any agent that relies on semantic tool discovery rather than a pinned allow-list obligingly executes the poisoned binary; the attack propagates further when automated IDE plug-ins or CI pipelines pull the latest version, granting the adversary broad access to internal documents, secrets, or compute resources.The only durable antidote is a defence-in-depth pipeline—deterministic builds signed with an organisation-wide key, mandatory hash verification at host start-up, runtime confinement inside seccomp- or AppArmor-restricted containers, and immutable audit logs that record each privileged action—so that even if a single component slips through code review, its blast radius remains tightly bounded.

M. SSE Channel Exhaustion

Streamable HTTP pairs request–response POSTs with a long-lived SSE stream so servers can push progress updates, but every open stream holds a TCP socket, TLS session, kernel buffer, and user-space queue; under legitimate spikes (for example, hundreds of browser tabs) or a deliberate slow-loris campaign, the file-descriptor table saturates,TLS caches thrash, and latency climbs until new sessions stall or the server crashes, effectively creating a denial of service. Engineering relief hinges on collapsing many logical streams into one multiplexed HTTP/2 or HTTP/3 connection, enforcing heartbeat pings that drop quiet clients, back-pressuring when queue depth grows by instructing clients to fall back to polling, and delegating large-scale fan-out to a purpose-built event broker such as NATS[153] or Kafka[154] so that the MCP server maintains only a lightweight upstream and never shoulders tens of thousands of idle sockets.

N. Cross-Server Privilege Escalation

In multi-tenant hosts where numerous MCP servers run side by side, namespace collisions become a stealthy escalation vector: if a low-trust server registers a broad URI root like `workspace://` before a high-trust peer registers the more specific `workspace://docs/*`, the host's longest-or first-match routing hands the attacker every subsequent request that should have gone to the privileged "Docs" server, granting unauthorised read or write access to sensitive files; the exploit can then propagate laterally as the attacker forwards or mutates

[153]https://nats.io/
[154]https://kafka.apache.org/

resources under the guise of legitimate traffic. Preventing such confusion requires the host to enforce prefix disjointness at registration time, attach mutual-TLS identities and per-server JWT audiences to every inter-component call, and maintain an explicit capability graph so that no newcomer can dominate or shadow an existing high-trust namespace.

O. Persistent Context Tampering

For convenience, many clients cache chat history, retrieved snippets, and tool outputs on local disk and replay them in later sessions, yet if that cache is stored unencrypted or without integrity checks a transient malware infection or curious colleague can inject forged memories—adding a fake "and my AWS root password is ABC123" line, for instance—so that the next time the conversation loads, the LLM cheerfully summarises or re-broadcasts the planted secret, biases analytics, or triggers an automated workflow, all without overt warnings; durable protection therefore involves storing caches in hardware-backed encrypted vaults (e.g. TPM, Data Protection API[155] or DPAPI, Keychain[156], Linux Unified Key Setup or LUKS[157]), chaining each turn with a rolling SHA-256 HMAC or Merkle root so any mutation breaks verification, and periodically re-hydrating context from authoritative servers to detect drift between local and canonical state.

P. Server-Data Take-Over

Because MCP servers frequently act as credential custodians—holding Slack bot tokens[158], GitHub Personal Access Tokens (PATs)[159], cloud refresh tokens[160], or database passwords—compromising a single server grants an attacker the transitive closure of downstream privileges, allowing them to pivot into SaaS accounts, spawn rogue Virtual Machines (VMs), push malicious code to internal registries, and finally register new MCP servers under the same host, thereby poisoning the entire capability graph; halting this domino effect demands hardware-sealed secret stores (e.g. Nitro Enclaves[161], Trusted Platform Module (TPM)-sealed blobs, or Hardware Security Module (HSMs), aggressively short-lived, scope-limited OAuth tokens that rotate automatically, micro-segmented network sandboxes enforced by Extended Berkeley Packet Filter (eBPF)[162] so east-west traffic is denied by default, and continuous secret scanners that quarantine any session where credentials appear in logs, context, or outbound traffic—closing the window of exploitation before the intruder can escalate beyond the initial beach-head.

[155]https://learn.microsoft.com/en-us/dotnet/standard/security/how-to-use-data-protection

[156]https://support.apple.com/en-in/guide/security/secb0694df1a/web

[157]https://access.redhat.com/solutions/100463

[158]https://api.slack.com/concepts/token-types

[159]https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens

[160]https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

[161]https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[162]https://ebpf.io/

VI. Future Directions

As the MCP continues to evolve in tandem with advances in AI and systems architecture, new opportunities emerge to enhance its flexibility, performance, and applicability. Addressing current limitations while anticipating the needs of increasingly complex LLM-driven workflows requires deliberate innovation across multiple dimensions. From optimizing context management to ensuring interoperability at scale, the trajectory of MCP points toward greater intelligence, automation, and security in real-time agentic systems. This section outlines strategic directions that can shape the next generation of MCP implementations, emphasizing both technical depth and practical scalability for cross-domain integration.

A. Enhanced Context Management

Advances in deep learning and NLP offer significant opportunities for refining context management within MCP. Future systems might integrate transformer-based models to compute semantic embeddings of incoming data streams in real time, enabling the dynamic ranking of context fragments based on cosine similarity or other vector-based similarity metrics. Techniques such as attention-based mechanisms could be leveraged to highlight context elements that are most relevant to the current inference task. Furthermore, research into adaptive context window algorithms, which adjust the size and content of the context based on user behavior and system performance metrics, could lead to more efficient and targeted data delivery. Integration of distributed caching mechanisms at the edge—using platforms like Redis[163] or memcached[164]—combined with predictive analytics for context pre-fetching and summarization via abstractive or extractive methods will further optimize the balance between data richness and processing latency.

B. Streamlined Capability Negotiation

Future improvements in MCP could see the advent of automated capability negotiation protocols that minimize the need for manual configuration. By standardizing a set of capability profiles—predefined configurations that encapsulate typical LLM application requirements—clients and servers can reference these profiles during initialization. Such profiles could be maintained in a decentralized registry using blockchain technology for tamper-proof versioning and auditability. Adaptive negotiation algorithms, perhaps leveraging reinforcement learning, could learn from past sessions to predict optimal capability matches and resolve discrepancies in real time. This approach would reduce negotiation overhead and lower the barrier to entry for new integrations, thereby enhancing system reliability and performance.

C. Cryptographic Enhancements

The rapidly evolving threat landscape necessitates the integration of next-generation cryptographic techniques into

[163]https://redis.io/

[164]https://memcached.org/

MCP. Future implementations might transition from traditional RSA[165] or ECC[166] to quantum-resistant algorithms like NTRU[167] or lattice-based cryptography [132], [133], thereby ensuring long-term security against potential quantum attacks. In addition, adopting blockchain-based distributed ledger technologies for decentralized token verification can provide immutable audit trails and robust data provenance. Enhanced security frameworks might also incorporate adaptive, context-aware access control systems that use machine learning to dynamically adjust permissions based on real-time risk assessments, anomaly detection, and user behavior analytics. These advancements, while computationally demanding, will be indispensable in securing high-stakes LLM applications that handle sensitive and mission-critical data.

D. Optimized Transport Protocols

Reducing transport layer latency remains a central focus for future MCP research. The adoption of next-generation protocols such as QUIC [134] or HTTP/3[135]—designed to reduce handshake overhead and enable multiplexed streams—could significantly improve data transmission efficiency. Coupling these protocols with edge computing architectures, where computation is distributed to servers closer to the end-user, can further reduce round-trip times. Moreover, developing specialized, protocol-aware compression algorithms that exploit the predictable structure of JSON-RPC messages could further minimize payload sizes. Hardware acceleration techniques, including the use of dedicated NICs, FPGA-based encryption, or GPU-accelerated serialization, might also be deployed to accelerate message processing, thereby enabling near-real-time responsiveness for interactive LLM applications.

E. Multi-Modal Data Support

The future of LLM applications is inherently multi-modal, necessitating that MCP evolve to natively support heterogeneous data types. Future protocol versions could adopt extensible schema definitions that encapsulate not only text but also high-fidelity image data, streaming audio, and even structured sensor or telemetry data. Transitioning from a text-based serialization format to binary formats can yield significant performance improvements in parsing and data integrity [136], [137]. Custom encoding schemes for different media types—optimized for compression, latency, and error resilience—will be crucial. This evolution will empower LLMs to ingest and process diverse inputs seamlessly, facilitating a more holistic and contextually nuanced understanding of multi-modal environments.

F. Context Caching

In future deployments, dynamic caching mechanisms will play a vital role in reducing latency and enhancing scalability. Distributed cache systems—implemented using high-performance solutions like Redis Cluster[168] or Apache Ignite[169]—could store frequently accessed context fragments locally at edge nodes [138], [139], [140]. Intelligent cache invalidation algorithms, driven by real-time analytics and machine learning predictions, would ensure that the cache remains fresh and relevant. Further, context summarization techniques by using both extractive and abstractive methods can distill large volumes of data into a concise format that retains key information. These caching strategies, when integrated with adaptive load balancing, can significantly reduce redundant data transfers, lower processing overhead, and provide rapid access to critical context for LLM inference.

G. Interoperability Standardization

Future efforts to unify the MCP ecosystem must focus on developing comprehensive interoperability standards that cover API contracts, data serialization formats, error handling conventions, and security protocols. Establishing a universal specification—accompanied by open-source reference implementations, standardized middleware libraries, and automated compliance testing frameworks—will be critical for reducing fragmentation. Collaborative initiatives involving industry consortia, academic research groups, and open-source communities can drive the creation of certification programs and interoperability test suites. These efforts will facilitate a more cohesive ecosystem where diverse MCP implementations interact seamlessly, thereby accelerating the integration of LLM applications across various platforms.

H. Distributed and Asynchronous Architectures

Future MCP architectures must be engineered for true scalability using modern distributed computing paradigms. Leveraging microservices architectures and serverless computing frameworks, such as AWS Lambda[170] or Google Cloud Functions[171], can decouple critical operations (e.g. token validation, context aggregation, and message routing) into independent, scalable units. Asynchronous programming models, powered by non-blocking I/O frameworks and reactive extensions (e.g. RxJava[172] or Project Reactor[173]), will enable the protocol to handle tens of thousands of concurrent sessions with minimal overhead. Incorporating advanced load balancing strategies (e.g. those based on consistent hashing or dynamic capacity adjustments) and employing distributed tracing tools will ensure that resources are allocated optimally, maintaining high throughput and low latency even during peak usage periods.

I. Human-in-the-Loop Enhancements

Future enhancements in MCP must also prioritize the end-user experience by providing intuitive interfaces and transparency in system operations. Advanced dashboards—built

[165]https://www.di-mgt.com.au/rsa_alg.html

[166]https://github.com/nakov/practical-cryptography-for-developers-book/blob/master/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc.md

[167]https://github.com/ProtDos/ntru

[168]https://redis.io/docs/latest/operate/oss_and_stack/management/scaling/

[169]https://ignite.apache.org/

[170]https://aws.amazon.com/lambda/

[171]https://cloud.google.com/functions

[172]https://github.com/ReactiveX/RxJava

[173]https://projectreactor.io/

with modern web technologies like React[174] or Angular[175] and integrated with real-time data analytics—can offer detailed visualizations of system health, context flow, and error metrics. These tools might incorporate interactive elements that allow administrators to drill down into individual sessions or override automated decisions when necessary. The integration of human-in-the-loop mechanisms, such as dynamic confirmation prompts and anomaly alert systems, can ensure that critical operations are subject to manual review, thereby reducing the risk of automated failures. Additionally, leveraging AI-based analytics to provide predictive insights and automated recommendations will enhance decision-making, fostering a more responsive and accountable system.

J. Community-Driven Innovation

The sustained evolution of MCP will depend on the continuous growth of its ecosystem, driven by active community engagement and collaborative innovation. Future initiatives should focus on building robust open-source frameworks, extensive documentation repositories, and comprehensive SDKs that facilitate rapid prototyping and integration. Centralized platforms—such as GitHub organizations, dedicated forums, and regular community-led events (e.g. hackathons and webinars)—will serve as fertile ground for sharing best practices, standardized development guidelines, and reference implementations. Formal governance structures and version control policies, along with certification programs for compliant implementations, can help maintain consistency across the ecosystem. This collaborative, community-driven approach will ensure that MCP not only adapts to emerging trends in LLM and AI integration but also drives innovation through collective expertise and shared technical vision.

VII. Conclusion

This study has demonstrated that the MCP furnishes the missing control plane required to transform LLM applications from isolated text generators into dependable, policy-compliant services. By unifying session state, capability negotiation, and tool invocation within a single JSON-RPC grammar, MCP enables developers to deliver end-to-end workflows in which every external action is both auditable and context-aware. Empirical evidence collected from software engineering, healthcare, and financial risk case studies confirms measurable gains in integration effort, latency, and security posture when MCP replaces traditional stateless APIs. Several open issues nevertheless constrain large-scale adoption. Transport-layer latency, long-lived token management, and cross-server namespace isolation remain active research challenges, while the emergence of multimodal prompts demands native support for non-text payloads. The article has outlined a research agenda that includes formal conformance test suites as promising avenues for future work.

[174]https://react.dev/
[175]https://angularjs.org/

References

[1] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen and K. Huang, "Mobile Edge Intelligence for Large Language Models: A Contemporary Survey," in IEEE Communications Surveys & Tutorials, doi: 10.1109/COMST.2025.3527641.

[2] S. Long et al., "A Survey on Intelligent Network Operations and Performance Optimization Based on Large Language Models," in IEEE Communications Surveys & Tutorials, doi: 10.1109/COMST.2025.3526606.

[3] A. Singh, A. Shetty, A. Ehtesham, S. Kumar and T. T. Khoei, "A Survey of Large Language Model-Based Generative AI for Text-to-SQL: Benchmarks, Applications, Use Cases, and Challenges," 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2025, pp. 00015-00021, doi: 10.1109/CCWC62904.2025.10903689.

[4] X. Wu, S. -H. Wu, J. Wu, L. Feng and K. C. Tan, "Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap," in IEEE Transactions on Evolutionary Computation, vol. 29, no. 2, pp. 534-554, April 2025, doi: 10.1109/TEVC.2024.3506731.

[5] M. Hindi, L. Mohammed, O. Maaz and A. Alwarafy, "Enhancing the Precision and Interpretability of Retrieval-Augmented Generation (RAG) in Legal Technology: A Survey," in IEEE Access, vol. 13, pp. 46171-46189, 2025, doi: 10.1109/ACCESS.2025.3550145.

[6] P. Wang, W. Lu, C. Lu, R. Zhou, M. Li and L. Qin, "Large Language Model for Medical Images: A Survey of Taxonomy, Systematic Review, and Future Trends," in Big Data Mining and Analytics, vol. 8, no. 2, pp. 496-517, April 2025, doi: 10.26599/BDMA.2024.9020090.

[7] P. Lohar and T. Baraskar, "Automated AI Tool for Log File Analysis," 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI), Goathgaun, Nepal, 2025, pp. 1762-1766, doi: 10.1109/ICMCSI64620.2025.10883511.

[8] I. Zualkernan, "Adoption of Generative AI and Large Language Models in Education: A Short Review," 2025 International Conference on Electronics, Information, and Communication (ICEIC), Osaka, Japan, 2025, pp. 1-5, doi: 10.1109/ICEIC64972.2025.10879632.

[9] W. Lan et al., "The Large Language Models on Biomedical Data Analysis: A Survey," in IEEE Journal of Biomedical and Health Informatics, doi: 10.1109/JBHI.2025.3530794.

[10] B. N. Javagal and S. Sharma, "A Comprehensive Survey on Clinical Models for AI-Powered Medical Applications," 2025 4th International Conference on Sentiment Analysis and Deep Learning (ICSADL), Bhimdatta, Nepal, 2025, pp. 1385-1391, doi: 10.1109/ICSADL65848.2025.10933337.

[11] N. Li et al., "Machine Unlearning: Taxonomy, Metrics, Applications, Challenges, and Prospects," in IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2025.3530988.

[12] D. Zhang, D. Listiyani, P. Singh and M. Mohanty, "Distilling Wisdom: A Review on Optimizing Learning From Massive Language Models," in IEEE Access, vol. 13, pp. 56296-56325, 2025, doi: 10.1109/ACCESS.2025.3554586.

[13] G. Bovenzi et al., "Mapping the Landscape of Generative AI in Network Monitoring and Management," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2025.3543022.

[14] L. Kumari, S. Serasiya and S. Bharti, "Deep Learning Approach for Evaluating Fairness in LLMs," 2025 International Conference on Sustainable Energy Technologies and Computational Intelligence (SETCOM), Gandhinagar, India, 2025, pp. 1-4, doi: 10.1109/SETCOM64758.2025.10932507.

[15] Model Context Protocol. https://modelcontextprotocol.io/introduction, Accessed on April 11, 2025.

[16] Model Context Protocol Github. https://github.com/modelcontextprotocol, Accessed on April 10, 2025.

[17] Anthropic MCP, https://www.anthropic.com/news/model-context-protocol, Accessed on April 10, 2025.

[18] J. Montemagno, Build a Model Context Protocol (MCP) server in C#, https://devblogs.microsoft.com/dotnet/build-a-model-context-protocol-mcp-server-in-csharp/, Accessed on April 9, 2025.

[19] A. Dutt, Model Context Protocol (MCP): A Guide With Demo Project, https://www.datacamp.com/tutorial/mcp-model-context-protocol, Accessed on April 10, 2025.

[20] MCP Philschmid, https://www.philschmid.de/mcp-introduction, Accessed on April 10, 2025.

[21] T. Ramchandani, https://medium.com/data-and-beyond/the-model-context-protocol-mcp-the-ultimate-guide-c40539e2a8e7, Accessed on April 7, 2025.

[22] M. Maheswaran, MCP 101: An Introduction to Model Context Protocol, https://www.digitalocean.com/community/tutorials/model-context-protocol, Accessed on April 8, 2025.

[23] F. Sunavala, Introducing Model Context Protocol (MCP) in Azure AI Foundry: Create an MCP Server with Azure AI Agent Service, https://devblogs.microsoft.com/foundry/integrating-azure-ai-agents-mcp/, Accessed on April 11, 2025.

[24] Sandeep, MCP – A Beginner's Guide, https://opencv.org/blog/model-context-protocol/, Accessed on April 6, 2025.

[25] A. Sharma, Model Context Protocol: The USB-C for AI: Simplifying LLM Integration, https://www.infracloud.io/blogs/model-context-protocol-simplifying-llm-integration/, Accessed on April 8, 2025.

[26] Spring Model Context Protocol (MCP), https://docs.spring.io/spring-ai/reference/api/mcp/mcp-overview.html, Accessed on April 5, 2025.

[27] B. P. Singhh, Model Context Protocol (MCP) in AI, https://blog.stackademic.com/model-context-protocol-mcp-in-ai-9858b5ecd9ce, Accessed on April 4, 2025.

[28] PydanticAI Model Context Protocol (MCP), https://ai.pydantic.dev/mcp/, Accessed on April 4, 2025.

[29] Z. Proser, What is the Model Context Protocol (MCP)? https://workos.com/blog/model-context-protocol, Accessed on April 4, 2025.

[30] Radosevich B, Halloran J. MCP Safety Audit: LLMs with the Model Context Protocol Allow Major Security Exploits. arXiv preprint arXiv:2504.03767. 2025 Apr 2.

[31] Singh A, Ehtesham A, Kumar S, Khoei TT. A Survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs).

[32] Hou X, Zhao Y, Wang S, Wang H. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv preprint arXiv:2503.23278. 2025 Mar 30.

[33] Szeider S. MCP-solver: Integrating language models with constraint programming systems. arXiv preprint arXiv:2501.00539. 2024 Dec 31.

[34] Jiang X, Gember-Jacobson A, Feamster N. CAIP: Detecting Router Misconfigurations with Context-Aware Iterative Prompting of LLMs. arXiv preprint arXiv:2411.14283. 2024 Nov 21.

[35] Dall'Agata F. Instructing network devices via large language models (Doctoral dissertation, Politecnico di Torino).

[36] Zhang H, Sediq AB, Afana A, Erol-Kantarci M. Large language models in wireless application design: In-context learning-enhanced automatic network intrusion detection. arXiv preprint arXiv:2405.11002. 2024 May 17.

[37] Hou X, Zhao Y, Wang H. The Next Frontier of LLM Applications: Open Ecosystems and Hardware Synergy. arXiv preprint arXiv:2503.04596. 2025 Mar 6.

[38] Cao X, Shi L, Wang X, Duan Y, Yang X, Zhang X. MCP: A Named Entity Recognition Method for Shearer Maintenance Based on Multi-Level Clue-Guided Prompt Learning. Applied Sciences (2076-3417). 2025 Feb 15;15(4).

[39] Namiot D, Ilyushin E. On Architecture of LLM agents. International Journal of Open Information Technologies. 2025 Jan 7;13(1):67-74.

[40] Restrepo Torres N. Automatic Transformation of natural language requirements to PLCOPen XML using Large Language Models (LLM).

[41] Jiang F, Peng Y, Dong L, Wang K, Yang K, Pan C, Niyato D, Dobre OA. Large language model enhanced multi-agent systems for 6G communications. IEEE Wireless Communications. 2024 Aug 16.

[42] Luo J, Zhang W, Yuan Y, Zhao Y, Yang J, Gu Y, Wu B, Chen B, Qiao Z, Long Q, Tu R. Large Language Model Agent: A Survey on Methodology, Applications and Challenges. arXiv preprint arXiv:2503.21460. 2025 Mar 27.

[43] Wu Z. Autono: ReAct-Based Highly Robust Autonomous Agent Framework. arXiv preprint arXiv:2504.04650. 2025 Apr 7.

[44] Lo FP, Qiu J, Wang Z, Yu H, Chen Y, Zhang G, Lo B. AI Hiring with LLMs: A Context-Aware and Explainable Multi-Agent Framework for Resume Screening. arXiv preprint arXiv:2504.02870. 2025 Apr 1.

[45] Zimmermann Y, Bazgir A, Afzal Z, Agbere F, Ai Q, Alampara N, Al-Feghali A, Ansari M, Antypov D, Aswad A, Bai J. Reflections from the 2024 large language model (llm) hackathon for applications in materials science and chemistry. arXiv preprint arXiv:2411.15221. 2024 Nov 20.

[46] Meijer E. From Function Frustrations to Framework Flexibility: Fixing tool calls with indirection. Queue. 2025 Feb 28;23(1):19-38.

[47] Z. Desai, 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/, Accessed on April 8, 2025.

[48] Sancheti P, Karlapalem K, Vemuri K. Llm driven web profile extraction for identical names. InCompanion Proceedings of the ACM Web Conference 2024 2024 May 13 (pp. 1616-1625).

[49] Gao G, Gan S, Wang X, Zhu S. LLMUZZ: LLM-based seed optimization for black-box device fuzzing. In2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) 2024 Dec 17 (pp. 1209-1216). IEEE.

[50] Xiao B, Kantarci B, Kang J, Niyato D, Guizani M. Efficient prompting for llm-based generative internet of things. IEEE Internet of Things Journal. 2024 Oct 4.

[51] Ng KK, Yan F, Hung K. Preliminary Study of LLM-Based Wordlist Generation for Validating Broken Web Access Control. InTENCON 2024-2024 IEEE Region 10 Conference (TENCON) 2024 Dec 1 (pp. 1088-1091). IEEE.

[52] Lenz M, Dumani L, Schenkel R, Bergmann R. ArgServices: A Microservice-Based Architecture for Argumentation Machines. InConference on Advances in Robust Argumentation Machines 2024 Jun 5 (pp. 352-369). Cham: Springer Nature Switzerland.

[53] Cai Z, Ma R, Fu Y, Zhang W, Ma R, Guan H. LLMaaS: Serving Large Language Models on Trusted Serverless Computing Platforms. IEEE Transactions on Artificial Intelligence. 2024 Jul 17.

[54] Cai Z, Ma R, Fu Y, Zhang W, Ma R, Guan H. LLMaaS: Serving Large Language Models on Trusted Serverless Computing Platforms. IEEE Transactions on Artificial Intelligence. 2024 Jul 17.

[55] JSON-RPC 2.0, https://www.jsonrpc.org/specification, Accessed on April 4, 2025.

[56] OAuth 2.1, https://oauth.net/2.1/, Accessed on April 5, 2025.

[57] Joshi I, Shahid S, Venneti SM, Vasu M, Zheng Y, Li Y, Krishnamurthy B, Chan GY. CoPrompter: User-Centric Evaluation of LLM Instruction Alignment for Improved Prompt Engineering. InProceedings of the 30th International Conference on Intelligent User Interfaces 2025 Mar 24 (pp. 341-365).

[58] Banday B, Thopalli K, Islam TZ, Thiagarajan JJ. On The Role of Prompt Construction In Enhancing Efficacy and Efficiency of LLM-Based Tabular Data Generation. InICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2025 Apr 6 (pp. 1-5). IEEE.

[59] Jeong J, Jang H, Park H. Large Language Models Are Better Logical Fallacy Reasoners with Counterargument, Explanation, and Goal-Aware Prompt Formulation. arXiv preprint arXiv:2503.23363. 2025 Mar 30.

[60] Park S, Choi A, Park R. Objection, your honor!: an LLM-driven approach for generating Korean criminal case counterarguments. Artificial Intelligence and Law. 2025 Feb 18:1-47.

[61] Shen Z. Llm with tools: A survey. arXiv preprint arXiv:2409.18807. 2024 Sep 24.

[62] Lu J, Holleis T, Zhang Y, Aumayer B, Nan F, Bai F, Ma S, Ma S, Li M, Yin G, Wang Z. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. arXiv preprint arXiv:2408.04682. 2024 Aug 8.

[63] Yang H, Siew M, Joe-Wong C. An llm-based digital twin for optimizing human-in-the loop systems. In2024 IEEE International Workshop on Foundation Models for Cyber-Physical Systems & Internet of Things (FMSys) 2024 May 13 (pp. 26-31). IEEE.

[64] Artemova E, Tsvigun A, Schlechtweg D, Fedorova N, Chernyshev K, Tilga S, Obmoroshev B. Hands-on tutorial: Labeling with llm and human-in-the-loop. arXiv preprint arXiv:2411.04637. 2024 Nov 7.

[65] EasyMCP, https://github.com/zcaceres/easy-mcp/, Accessed on April 10, 2025.

[66] FastAPI to MCP auto generator, https://github.com/tadata-org/fastapi_mcp, Accessed on April 10, 2025.

[67] FastMCP, https://github.com/punkpeye/fastmcp, Accessed on April 10, 2025.

[68] Foxy Contexts, https://github.com/strowk/foxy-contexts, Accessed on April 10, 2025.

[69] Higress MCP Server Hosting, https://github.com/alibaba/higress/tree/main/plugins/wasm-go/mcp-servers, Accessed on April 10, 2025.

[70] MCP-Framework, https://mcp-framework.com/, Accessed on April 10, 2025.

[71] Quarkus MCP Server SDK, https://github.com/quarkiverse/quarkus-mcp-server, Accessed on April 10, 2025.

[72] Template MCP Server, https://github.com/mcpdotdirect/template-mcp-server, Accessed on April 10, 2025.

[73] codemirror-mcp, https://github.com/marimo-team/codemirror-mcp, Accessed on April 10, 2025.

[74] Java HTTPClient https://docs.oracle.com/en/java/javase/11/docs/api/java.net.http/java/net/http/HttpClient.htmls, Accessed on April 13, 2025.

[75] Spring WebFlux, https://docs.spring.io/spring-framework/reference/web/webflux.html, Accessed on April 13, 2025.

[76] Spring WebMVC, https://docs.spring.io/spring-framework/reference/web/webmvc.html, Accessed on April 13, 2025.

[77] Python SDK, https://github.com/modelcontextprotocol/python-sdk, Accessed on April 7, 2025.

[78] TypeScript SDK, https://github.com/modelcontextprotocol/typescript-sdk, Accessed on April 7, 2025.

[79] Java SDK, https://github.com/modelcontextprotocol/java-sdk, Accessed on April 7, 2025.

[80] Kotlin SDK, https://github.com/modelcontextprotocol/kotlin-sdk, Accessed on April 7, 2025.

[81] C# SDK, https://github.com/modelcontextprotocol/csharp-sdk, Accessed on April 7, 2025.

[82] C.Greyling, Using LangChain With Model Context Protocol (MCP), https://cobusgreyling.medium.com/using-langchain-with-model-context-protocol-mcp-e89b87ee3c4c, Accessed on April 13, 2025.

[83] B. Young, The Model Context Protocol (MCP): A guide for AI integration, https://wandb.ai/byyoung3/Generative-AI/reports/The-Model-Context-Protocol-MCP-A-guide-for-AI-integration–VmlldzoxMTgzNDgxOQ, Accessed on April 13, 2025.

[84] A. Hathibelagal, Understanding Model Context Protocol (MCP), https://medium.com/predict/understanding-model-context-protocol-mcp-771f1cfb3c0a, Accessed on April 13, 2025.

[85] A. Hathibelagal, Using the Model Context Protocol (MCP) With a Local LLM, https://medium.com/predict/using-the-model-context-protocol-mcp-with-a-local-llm-e398d6f318c3, Accessed on April 13, 2025.

[86] E. Solano Granados, Model Context Protocol: Expanding LLM Capabilities, https://stvansolano.github.io/2025/03/16/AI-Agents-Model-Context-Protocol-Explained/, Accessed on April 13, 2025.

[87] G. Hilston, Model Context Protocol (MCP): The Future of LLM Function Calling, https://www.greghilston.com/post/model-context-protocol/, Accessed on April 13, 2025.

[88] What Is the Model Context Protocol (MCP) and How It Works, https://www.descope.com/learn/post/mcp, Accessed on April 14, 2025.

[89] C. E. Perez, The AI-Native Revolution: How Model Context Protocol (MCP) is Radically Changing Software, https://medium.com/intuitionmachine/the-ai-native-revolution-how-model-context-protocol-is-radically-changing-software-1e610b97f6c4, Accessed on April 14, 2025.

[90] Model Context Protocol (MCP) Integration Guide for AI Developers, https://dysnix.com/blog/model-context-protocol, Accessed on April 14, 2025.

[91] S. Medina and Y. Orlev, Building Agentic Flows with LangGraph and Model Context Protocol, https://www.qodo.ai/blog/building-agentic-flows-with-langgraph-model-context-protocol/, Accessed on April 14, 2025.

[92] Model Context Protocol (MCP) Server Development Guide: Building Powerful Tools for LLMs, https://github.com/cyanheads/model-context-protocol-resources/blob/main/guides/mcp-server-development-guide.md, Accessed on April 14, 2025.

[93] Agents and tools: Model Context Protocol (MCP), https://docs.anthropic.com/en/docs/agents-and-tools/mcp, Accessed on April 14, 2025.

[94] A. Prabhulal, Model Context Protocol(MCP) with Google Gemini 2.5 Pro — A Deep Dive (Full Code), https://medium.com/google-cloud/model-context-protocol-mcp-with-google-gemini-llm-a-deep-dive-full-code-ea16e3fac9a3, Accessed on April 14, 2025.

[95] N. Diamant, Model Context Protocol (MCP) Explained: The Universal Connector for AI Systems, https://diamantai.substack.com/p/model-context-protocol-mcp-explained, Accessed on April 14, 2025.

[96] A. Sivadas, Model Context Protocol (MCP): Why it matters!, https://community.aws/content/2uY3Xf2WC5tSQcsBQgYQk9IQF1C/model-context-protocol-mcp-why-it-matters, Accessed on April 14, 2025.

[97] Building Your First Model Context Protocol Server, https://thenewstack.io/building-your-first-model-context-protocol-server/, Accessed on April 14, 2025.

[98] C. Jeng Yang, Improving LLMs: ETL to "ECL" (Extract-Contextualize-Load), https://medium.com/enterprise-rag/improving-llm-information-retrieval-etl-to-ecl-extract-contextualize-load-12a4ac259faa, Accessed on April 14, 2025.

[99] A. Sharma, Model Context Protocol: The USB-C for AI: Simplifying LLM Integration, https://www.infracloud.io/blogs/model-context-protocol-simplifying-llm-integration/, Accessed on April 14, 2025.

[100] ODSC, AI-Powered ETL Pipeline Orchestration: Multi-Agent Systems in the Era of Generative AI, https://odsc.medium.com/ai-powered-etl-pipeline-orchestration-multi-agent-systems-in-the-era-of-generative-ai-91dd0c1c8a80, Accessed on April 14, 2025.

[101] Enhancing Healthcare AI with Model Context Protocol and Semantic Kernel, https://www.paulswider.com/executive-summary-enhancing-healthcare-ai-with-model-context-protocol-and-semantic-kernel/, Accessed on April 14, 2025.

[102] Model Context Protocol (MCP) and Its Impact on AI-Driven Startups, https://www.aalpha.net/blog/model-context-protocol-mcp-and-its-impact-on-ai-driven-startups/, Accessed on April 14, 2025.

[103] How the Model Context Protocol (MCP) is Revolutionizing AI in Academia, https://www.linkedin.com/pulse/how-model-context-protocol-mcp-revolutionizing-ai-rajaratnam-wejec/, Accessed on April 14, 2025.

[104] Model Context Protocol (MCP): A comprehensive introduction for developers, https://stytch.com/blog/model-context-protocol-introduction/, Accessed on April 14, 2025.

[105] Particular Audience Launches Retail-MCP.com to Enable Faster, AI-Driven Commerce, https://www.businesswire.com/news/home/20250410945929/en/Particular-Audience-Launches-Retail-MCP.com-to-Enable-Faster-AI-Driven-Commerce, Accessed on April 14, 2025.

[106] Model Context Protocol (MCP): Bridging AI and Legal Data Systems, https://news-insights.kthlaw.com/en-us/ai/ai-legal-evaluation/model-context-protocol-mcp-bridging-ai-and-legal-data-systems, Accessed on April 14, 2025.

[107] M. Mueller, How to Use Model Context Protocol the Right Way, https://boomi.com/blog/model-context-protocol-how-to-use/, Accessed on April 14, 2025.

[108] M. Tanveer, Model Context Protocol (MCP): The Next Frontier in Information Security, https://medium.com/@threatpointer/model-context-protocol-mcp-the-next-frontier-in-information-security-097cfc18bf2c, Accessed on April 14, 2025.

[109] O. Santos, AI Model Context Protocol (MCP) and Security, https://community.cisco.com/t5/security-blogs/ai-model-context-protocol-mcp-and-security/ba-p/5274394, Accessed on April 14, 2025.

[110] How Model Context Protocol (MCP) Can Transform Your Business in 2025, https://www.linkedin.com/pulse/how-model-context-protocol-mcp-can-transform-your-business-g19cc/, Accessed on April 14, 2025.

[111] S. Li, Model Context Protocol (MCP) and Its Industry Impact, https://medium.com/@lihaoming2000/model-context-protocol-mcp-and-its-industry-impact-736d70713491, Accessed on April 14, 2025.

[112] mcp-korea-tourism-api, https://github.com/harimkang/mcp-korea-tourism-api, Accessed on April 14, 2025.

[113] T. K. Koc, MCP: Model Context Protocol — MCP vs. Traditional APIs & RAG, https://medium.com/nane-limon/mcp-model-context-protocol-mcp-vs-traditional-apis-rag-81eebff65111, Accessed on April 14, 2025.

[114] Chauhan S, Rasheed Z, Sami AM, Zhang Z, Rasku J, Kemell KK, Abrahamsson P. LLM-Generated Microservice Implementations from RESTful API Definitions. arXiv preprint arXiv:2502.09766. 2025 Feb 13.

[115] Lehmann R. Towards Interoperability of APIs-an LLM-based approach. InProceedings of the 25th International Middleware Conference: Demos, Posters and Doctoral Symposium 2024 Dec 2 (pp. 29-30).

[116] Kim M, Stennett T, Sinha S, Orso A. A Multi-Agent Approach for REST API Testing with Semantic Graphs and LLM-Driven Inputs. arXiv preprint arXiv:2411.07098. 2024 Nov 11.

[117] Sri SD, Raman RC, Rajagopal G, Chan ST. Automating REST API Postman Test Cases Using LLM. arXiv preprint arXiv:2404.10678. 2024 Apr 16.

[118] Zhang K, Zhang C, Wang C, Zhang C, Wu Y, Xing Z, Liu Y, Li Q, Peng X. LogiAgent: Automated Logical Testing for REST Systems with LLM-Based Multi-Agents. arXiv preprint arXiv:2503.15079. 2025 Mar 19.

[119] A. Arsanjani, Complementary Protocols for Agentic Systems:Understanding Google's A2A & Anthropic's MCP, https://dr-arsanjani.medium.com/complementary-protocols-for-agentic-systems-understanding-googles-a2a-anthropic-s-mcp-47f5e66b6486, Accessed on April 15, 2025.

[120] S. Nitesh Palamakula, Agent-to-Agent Protocol vs. Model Context Protocol (MCP), https://medium.com/@sainitesh/agent-to-agent-protocols-vs-model-context-protocol-mcp-b46e31599647, Accessed on April 15, 2025.

[121] B. Yang, MCP vs A2A: Which Protocol Is Better For AI Agents? [2025], https://www.blott.studio/blog/post/mcp-vs-a2a-which-protocol-is-better-for-ai-agents, Accessed on April 15, 2025.

[122] A. Griciunas, MCP vs. A2A: Friends or Foes?, https://www.newsletter.swirlai.com/p/mcp-vs-a2a-friends-or-foes, Accessed on April 15, 2025.

[123] Alisdair Broshar, A2A and MCP: Start of the AI Agent Protocol Wars?, https://www.koyeb.com/blog/a2a-and-mcp-start-of-the-ai-agent-protocol-wars, Accessed on April 15, 2025.

[124] R. Surapaneni, M. Jha, M. Vakoc, T. Segal, Announcing the Agent2Agent Protocol (A2A), https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/, Accessed on April 15, 2025.

[125] N. Koul, The Model Context Protocol (MCP) — A Complete Tutorial, https://medium.com/@nimritakoul01/the-model-context-protocol-mcp-a-complete-tutorial-a3abe8a7f4ef, Accessed on April 14, 2025.

[126] C. Kelly, Model Context Protocol (MCP): Connecting AI Models to Real-World Data, https://humanloop.com/blog/mcp, Accessed on April 14, 2025.

[127] Model Context Protocol has prompt injection security problems, https://simonwillison.net/2025/Apr/9/mcp-prompt-injection/, Accessed on April 14, 2025.

[128] D. Sarig, The Security Risks of Model Context Protocol (MCP), https://www.pillar.security/blog/the-security-risks-of-model-context-protocol-mcp, Accessed on April 14, 2025.

[129] Authorization, https://developers.cloudflare.com/agents/model-context-protocol/authorization/, Accessed on April 14, 2025.

[130] J.Cruz Martinez, An Introduction to MCP and Authorization, https://auth0.com/blog/an-introduction-to-mcp-and-authorization/, Accessed on April 14, 2025.

[131] A. Minessale, Evolving the Model Context Protocol (MCP) for Reliable Real-Time AI and Telecom Integration, https://signalwire.com/blogs/ceo/evolving-the-model-context-protocol, Accessed on April 14, 2025.

[132] Micciancio D, Regev O. Lattice-based cryptography. InPost-quantum cryptography 2009 Feb 1 (pp. 147-191). Berlin, Heidelberg: Springer Berlin Heidelberg.

[133] Nejatollahi H, Dutt N, Ray S, Regazzoni F, Banerjee I, Cammarota R. Post-quantum lattice-based cryptography implementations: A survey. ACM Computing Surveys (CSUR). 2019 Jan 28;51(6):1-41.

[134] QUIC, https://quicwg.org/, Accessed on April 10, 2025.

[135] HTTP/3, https://datatracker.ietf.org/doc/rfc9114/, Accessed on April 10, 2025.

[136] Tang Y, Chang CM, Yang X. Pdfchatannotator: A human-llm collaborative multi-modal data annotation tool for pdf-format catalogs. InProceedings of the 29th International Conference on Intelligent User Interfaces 2024 Mar 18 (pp. 419-430).

[137] Chen H, Wang X, Zhou Y, Huang B, Zhang Y, Feng W, Chen H, Zhang Z, Tang S, Zhu W. Multi-modal generative ai: Multi-modal llm, diffusion and beyond. arXiv preprint arXiv:2409.14993. 2024 Sep 23.

[138] Monteiro J, Marcotte É, Noël PA, Zantedeschi V, Vázquez D, Chapados N, Pal C, Taslakian P. Xc-cache: Cross-attending to cached context for efficient llm inference. arXiv preprint arXiv:2404.15420. 2024 Apr 23.

[139] Hu C, Huang H, Hu J, Xu J, Chen X, Xie T, Wang C, Wang S, Bao Y, Sun N, Shan Y. Memserve: Context caching for disaggregated llm serving with elastic memory pool. arXiv preprint arXiv:2406.17565. 2024 Jun 25.

[140] Mohandoss R. Context-based semantic caching for llm applications. In2024 IEEE Conference on Artificial Intelligence (CAI) 2024 Jun 25 (pp. 371-376). IEEE.

Appendix

TABLE IX: List of Community-Developed MCP Servers

Server Name	Domain	Purpose
Ableton Live	Audio / DAW Automation	Allows LLMs to control music production workflows within Ableton Live
Airbnb	Travel / Real Estate	Enables listing search and retrieval of stay information from Airbnb
AI Agent Marketplace Index	AI Agents Discovery	Provides access to thousands of AI agents with traffic metrics and classification
Algorand	Blockchain / DeFi	Facilitates LLM-based interaction with Algorand's extensive APIs and prompts
Airflow	Workflow Orchestration	Connects with Apache Airflow to query and control pipeline tasks
Airtable	NoSQL Databases	Enables structured read/write operations and schema exploration on Airtable
AlphaVantage	Stock Market / Finance	Offers real-time and historical financial data via AlphaVantage API
Amadeus	Travel / Flight Search	Retrieves live flight options—airlines, schedules, durations, and prices—through Amadeus Flight Offers Search API
Anki	Education / Flashcards	Interfaces with Anki decks for spaced repetition and card operations
Any Chat Completions	LLM APIs / Chat	Bridges to any OpenAI-compatible chat-completion endpoint (OpenAI, Perplexity, Groq, xAI, and others) for conversational AI tasks
Apple Calendar	Calendar / macOS Automation	Creates, edits, lists, and analyzes events in the native macOS Calendar using natural-language commands
ArangoDB	Databases / Multi-Model	Performs queries and data operations on ArangoDB's graph-document-key store via structured MCP tools
Arduino	IoT / Robotics	Links Claude AI with Arduino (ESP32) for physical automation tasks
Atlassian	Collaboration / Project Management	Searches Confluence spaces and pages and accesses Jira issues and project metadata from Atlassian Cloud
Attestable MCP	Security / Trusted Execution	Demonstrates remote attestation with RA-TLS, letting clients verify the MCP server inside a Gramine TEE before use
AWS	Cloud / Resource Management	Executes natural-language operations across AWS services and infrastructure through boto3-powered actions
AWS Athena	Cloud / Query Engines	Executes SQL queries over AWS Glue Catalog using Athena
AWS Cost Explorer	Cloud / Cost Analytics	Analyzes AWS and Amazon Bedrock spend by region, service, instance type, and foundation model to optimize costs
AWS Resources Operations	Cloud / Python Automation	Generates and runs secure Python code snippets to inspect or modify any AWS resource supported by boto3
AWS S3	Cloud Storage / Object Store	Flexibly fetches and returns objects—PDFs, images, data files—from Amazon S3 buckets
Azure ADX	Cloud Databases / Analytics	Issues Kusto queries to explore and analyze datasets in Azure Data Explorer
Azure DevOps	DevOps / CI-CD	Queries and manages Azure DevOps work items, pipelines, and repositories via MCP tools
Baidu AI Search	Web Search / China	Performs web searches through Baidu Cloud's AI Search APIs, returning ranked results and metadata
Base Free USDC Transfer	Blockchain / Payments	Sends fee-free USDC transfers on the Base network using Coinbase CDP from within Claude AI
Basic Memory	Knowledge Management	Builds semantic memory graphs from markdown files for persistent context
BigQuery (LucasHild)	Cloud Databases	Enables querying and schema inspection for BigQuery datasets
BigQuery (by ergut)	Cloud Databases / Analytics	Provides direct SQL access and schema inspection for Google BigQuery datasets
Bing Web Search API (by hanchunglee)	Web Search / Microsoft	Queries Microsoft Bing Web Search API to return web, news, and image results with ranking data
Bitable MCP (by lloydzhou)	No-Code / Tables	Reads and writes Lark Bitable tables, enabling spreadsheet-like operations via predefined tools
Blender (by ahujasid)	3D Modeling	Empowers LLMs to generate and manipulate 3D scenes using Blender
browser-use	Browser Automation	Supports full browser control using Playwright with VNC-based interaction

Server Name	Domain	Purpose
Bsc-mcp	Crypto / DeFi	Allows smart contract actions and asset transfers on BNB Chain with AI
Calculator	Utility / Math Tools	Adds precise computation capabilities to LLMs via calculator integration
CFBD API	Sports / Football Data	Retrieves game and team data from College Football Data API
ClaudePost	Email / Gmail	Automates Gmail tasks such as reading, searching, and sending mail securely
ChatMCP	GUI / MCP Frontend	Cross-platform desktop application to interact with multiple LLMs and MCP servers
ChatSum	Messaging / Summarization	Enables LLMs to summarize chat conversations and threads contextually
Chroma	Vector DB / Search	Provides semantic document search and metadata filtering using Chroma
ClaudePost	Email / Gmail	Securely enables reading, writing, and managing Gmail using Claude
Cloudinary	Media Hosting	Upload media files to Cloudinary and retrieve hosted links and metadata
code-assistant	Development / IDEs	Navigate and modify trusted codebases using a code-aware LLM assistant
code-executor	Code Execution / Sandbox	Safely runs Python scripts in a designated Conda environment
code-sandbox-mcp	Code Sandbox / Docker	Sets up a secure Docker sandbox for executing isolated code
cognee-mcp	Memory / Knowledge Graph	Custom GraphRAG memory tools for semantic search and data ingestion
coin_api_mcp	Crypto / Market Data	Real-time cryptocurrency data via CoinMarketCap and related sources
Contentful-mcp	CMS / ContentOps	Enables reading, updating, and publishing content in Contentful spaces
crypto-feargreed-mcp	Sentiment Analysis / Crypto	Provides sentiment index data for crypto market risk perception
cryptopanic-mcp-server	Crypto / News Aggregation	Fetches latest crypto-related news articles from CryptoPanic
Dappier	Finance / Real-Time Data	Access real-time financial data from trusted sources and publishers
Databricks	Analytics / Cloud DB	Run SQL queries and manage jobs in Databricks environments
Datadog	Monitoring / Observability	Query traces, incidents, and dashboard data from Datadog API
Data Exploration	Auto Data Insight	Automatically analyzes CSV datasets and extracts intelligent insights
Dataset Viewer	Dataset Analysis	Explore, search, and analyze Hugging Face datasets interactively
DBHub	Databases / Universal Access	Connects to various relational databases including MySQL, PostgreSQL, SQLite, and DuckDB
DeepSeek MCP Server	LLM Integration / DeepSeek	Integrates DeepSeek models and tools for enhanced language tasks
Deepseek_R1	LLM Model Access	Claude Desktop compatible server for DeepSeek's reasoning models
deepseek-thinker-mcp	Reasoning / LLM Thinker	Access DeepSeek's local or API reasoning content as a tool
Descope	Identity / Access Logs	Search audit logs and manage user credentials via Descope API
DevRev	Knowledge Graph / APIs	Enables querying through DevRev's structured knowledge graph
Dicom	Medical Imaging	Parse and retrieve DICOM files, including embedded PDF content
Dify	Workflow Automation	Executes Dify workflows using preconfigured task sequences
Discord MCP (Bot)	Social / Messaging	Connects to Discord servers to send and read messages as a bot
Discord (Extended)	Messaging / LLMs	Offers full integration with Discord: threads, mentions, reactions
Discourse	Forum Search / Web	Searches and retrieves posts from a Discourse-based forum
Docker	Containers / DevOps	Manages Docker containers, volumes, and network configurations
Drupal	CMS / Website Management	Interacts with Drupal instances via standardized transport layer
dune-analytics-mcp	Blockchain / Analytics	Queries blockchain metrics from Dune Analytics platform
EdgeOne Pages MCP	Web Hosting / Deployment	Deploys HTML content to EdgeOne Pages with public URL return
Elasticsearch MCP	Search Engine / Indexing	Interface to search and manage Elasticsearch documents and clusters
ElevenLabs	Audio / TTS	Generates text-to-speech audio from LLM output using ElevenLabs API
Ergo Blockchain MCP	Blockchain / Forensics	Analyze transactions, balances, and activity on Ergo blockchain
Eunomia	Automation Framework	Connects Eunomia instrumentation and workflows to MCP agents
EVM MCP Server	Blockchain / EVM Chains	Comprehensive tooling for interacting with 30+ EVM-compatible blockchains

Server Name	Domain	Purpose
Everything Search	File Indexing / Local Search	Multi-platform file search using OS-native indexing tools
Excel	Spreadsheet Automation	Enables reading, writing, formatting, and chart creation in Excel
Fantasy PL	Sports / Game Analytics	Provides up-to-date Fantasy Premier League stats and team data
fastn.ai	Unified API Gateway	Remote MCP gateway to 1000+ tools and workflows with dynamic auth
Fetch	Web Data Fetching	Flexible content retriever for HTML, JSON, markdown, and plain text
Fingertip	Web Builder / CMS	Access Fingertip to search or create static website content
Figma	Design Tool Integration	Connects to Figma files for reading design assets and elements
Firebase	Cloud / Backend Services	Interacts with Firestore, Storage, and Authentication in Firebase
FireCrawl	Scraping / Web Extraction	JavaScript-enabled scraper with PDF support and rate control
FlightRadar24	Aviation / Real-Time Tracking	Tracks flights and metadata via Flightradar24 integration
Ghost	CMS / Blogging	Publishes, edits, and manages content in Ghost CMS via MCP
GitHub Actions	CI/CD Automation	Allows LLMs to query, trigger, and monitor GitHub Actions workflows
Glean	Enterprise Search	Integrates with Glean API for knowledge base search and retrieval
Gmail	Email / Claude Desktop	Manages Gmail messages securely within Claude Desktop environment
Gmail Headless	Email / Remote Access	Remotely fetches and sends Gmail emails without local auth setup
Goal Story	Goal Management	A visualization tool for personal and professional goal tracking
GOAT	Blockchain / Smart Actions	Executes over 200 on-chain blockchain actions on Ethereum, Solana, and Base
Godot	Game Engine / Dev Tools	Provides editing and debugging capabilities for Godot projects
Golang Filesystem Server	Filesystem / Security	Secure Go-based file operations with access control policies
Goodnews	News / Positivity Feed	Sends curated uplifting news to LLM interfaces for positive prompts
Google Calendar	Calendar / Scheduling	View, add, or manage events in Google Calendar via MCP tools
Google Custom Search	Web Search / API	Searches Google using Custom Search API with configurable settings
Google Tasks	Productivity / Task Lists	Create, update, or delete tasks in Google Tasks via LLMs
GraphQL Schema	API / Developer Tools	Explore and interact with complex GraphQL schemas dynamically
HDW LinkedIn	Social Media / Data Access	Retrieves LinkedIn profile information via HorizonDataWave.ai
Heurist Mesh Agent	Blockchain / Web3 Agents	Access Web3 agents for token analysis, contract review, and metrics
Holaspirit	Team Management	Integrates Holaspirit platform for roles, policies, and team data
Home Assistant	IoT / Smart Home	Interacts with devices, sensors, and automations in Home Assistant
Home Assistant (Docker)	IoT / Containerized	Containerized MCP for Home Assistant with rich domain control
HubSpot	CRM / Customer Data	Retrieve and manage contacts and companies using HubSpot CRM APIs
HuggingFace Spaces	AI Models / Open Source	Interacts with image, audio, and text demos from Hugging Face Spaces
Hyperliquid	Crypto Exchange / SDK	Provides LLM access to trading data via Hyperliquid SDK integration
iFlytek Workflow	Workflow Automation	Connects to iFlytek Workflow API for managing personal agent tasks
Image Generation	AI / Multimedia Creation	Enables LLM-driven image generation using Replicate's Flux model
InfluxDB	Time-Series Database	Allows querying and analyzing InfluxDB v2 for time-series data
Inoyu	Customer Data Platform	Connects to Apache Unomi for managing and updating user profiles
Intercom	Customer Support	Retrieves support tickets and insights from Intercom's support system
iOS Simulator	Mobile Testing / Apple	Executes natural language commands on iOS simulators for testing
iTerm MCP	Terminal / macOS	Controls and monitors commands in iTerm2 terminal emulator
JavaFX	Graphics / Canvas Drawing	Generates canvas-based drawings using JavaFX from prompts
JDBC	Databases / Universal Access	Connects to JDBC-supported databases for SQL operations
JSON	Data / JSON Utilities	Parses, queries, and manipulates JSON using advanced JSONPath features
KiCad MCP	Electronics / CAD	Offers tool access for electronic design in KiCad environments
Keycloak MCP	Identity / Access Control	Manages Keycloak users and realms via natural language commands
Kibela (kiwamizamurai)	Collaboration / Docs	Connects to Kibela for document interaction via Kibela API
kintone	Enterprise Apps	Manages records and apps in kintone using natural language prompts
Kong Konnect	API Gateway / Analytics	Allows querying traffic metrics and gateway configurations in Kong

Server Name	Domain	Purpose
Kubernetes	DevOps / Orchestration	Manages clusters, pods, and deployments on Kubernetes via LLM
Kubernetes and OpenShift	OpenShift Extension	Enhanced Kubernetes server with OpenShift support and resource tools
Langflow DOC-QA Server	Document QA	Queries documents using Langflow backend for context retrieval
Lightdash	BI / Analytics	Connects LLMs to Lightdash dashboards for data querying
Linear	Project Management	Interfaces with Linear to create, update, and manage tasks and issues
Linear (Go)	Project Management / Go	Lightweight static binary-based Linear API access for agents
LINE (amornpan)	Messaging / LINE Bot	Integrates LINE messaging with logging, webhook handling, and more
LlamaCloud (marcuss-chiesser)	Indexing / Vector DB	Interacts with LlamaCloud managed indexes via MCP toolset
llm-context	Repo Packing / Profiles	Creates filtered context bundles from codebases with prompts
mac-messages-mcp	Messaging / iMessage	Access and query iMessage conversations securely using MCP
MariaDB	Databases / SQL	Secure access and operations on MariaDB using Python APIs
Maton	SaaS / CRM	Connects SaaS services like Salesforce and HubSpot for unified queries
MCP Compass	Server Discovery	Helps agents choose the most relevant MCP server for their needs
MCP Create	Dynamic MCP Hosting	Builds and manages MCP server instances dynamically on request
MCP Installer	Installer / Server Manager	Automates installation of other MCP servers
mcp-k8s-go	Kubernetes / Golang MCP	Lightweight Go-based Kubernetes browser with log and namespace access
mcp-local-rag	RAG / Local Search	Provides local semantic search and embedding using DuckDuckGo and MediaPipe
mcp-proxy	Proxy / SSE	Proxies SSE or STDIO servers to provide accessible interfaces
mem0-mcp	Developer Settings / AI	Interfaces with Mem0 to manage dev preferences and behavior memory
MSSQL	Databases / Microsoft SQL	Supports querying, schema inspection, and secure access to MSSQL
MSSQL (jexin)	MSSQL / Python	Lightweight implementation of MSSQL server access via Python
MSSQL (amornpan)	MSSQL / Read-Only	Read-only MSSQL implementation with Python and strict security
MSSQL (daobataotie)	MSSQL / SQLite Fork	Modified SQLite MCP version adapted for Microsoft SQL Server
Markdownify	Conversion / Markdown	Converts formats like PPTX, HTML, PDFs into clean Markdown
Microsoft Teams	Messaging / Collaboration	Sends, mentions, lists messages and threads via MS Teams API
Mindmap (YuChenSSR)	Knowledge Visualization	Creates mindmaps from structured markdown content
Minima	RAG / Local Files	Provides retrieval augmented generation from local file corpus
Mobile MCP (Mobile Next)	Mobile / Automation	Automates iOS/Android scraping and testing via physical or virtual devices
MongoDB	Databases / NoSQL	Accesses and queries MongoDB data through structured tools
MongoDB Lens	Databases / Mongo UI	Full-featured interaction with MongoDB from an LLM environment
Monday.com	Task Boards	Manages boards and project items in Monday.com via LLM interfaces
Multicluster-MCP-Server	Kubernetes / GenAI Gateway	Enables GenAI agents to interact with multiple K8s clusters concurrently
MySQL (by benborla)	Databases / MySQL	NodeJS-based MySQL MCP server with access control and schema browsing
MySQL (by DesignComputer)	Databases / MySQL	Python-based implementation for structured MySQL access
n8n	Automation / Workflow Orchestration	LLM-friendly access to create, monitor, and manage n8n workflows
NASA (by ProgramComputer)	Space / Public Data API	Accesses unified NASA APIs including APOD, NEO, EPIC, and GIBS
Nasdaq Data Link (by stefanoamorelli)	Finance / Datasets	Offers financial and economic datasets from Nasdaq Data Link
National Parks	Tourism / API Access	Retrieves alerts, trails, and event data from US National Parks system
NAVER (by pfldy2850)	Search / Korean Services	Interfaces with Naver to search blogs, news, books, and more
NS Travel Information	Transport / Real-Time Data	Provides Dutch railway schedules, delays, and alerts via NS API
Neo4j	Databases / Graph DB	Enables interaction with Neo4j graph databases using Cypher queries
Neovim	Developer Tools / Code Editor	Embeds LLMs in Neovim sessions for smart code editing and control
Notion (by suekou)	Note-Taking / Workspace	Access Notion's API for document search and workspace automation
Notion (by v-3)	Productivity / Notes	Full Notion integration for creating, updating, and reading pages

Server Name	Domain	Purpose
ntfy-mcp (by teddyzxcv)	Notifications / Alerts	Sends phone notifications from MCP via ntfy push service
oatpp-mcp	C++ / API Framework	C++-based server framework for creating scalable MCP tools
Obsidian Markdown Notes	Notes / File Access	Search and retrieve notes from Markdown folders (e.g., Obsidian vaults)
obsidian-mcp (by Steven Stavrakis)	Notes / Organization	Advanced MCP server for reading, organizing, and writing Obsidian notes
OceanBase (by yuanoOo)	Databases / Distributed SQL	Securely manages queries and schema inspection on OceanBase
Okta	Identity Management	Integrates with Okta API for user and realm operations
OneNote (by Rajesh Vijay)	Notes / Microsoft Graph	Read and write content to Microsoft OneNote using Graph API
OpenAI WebSearch MCP	Web Search / OpenAI Tools	Provides search functionality to OpenAI-compatible agents
OpenAPI	APIs / Tool Access	Allows exploration and interaction with generic OpenAPI-compatible APIs
OpenAPI AnyApi	APIs / Semantic Discovery	Enables semantic endpoint search across large OpenAPI documents
OpenAPI Schema	APIs / Introspection	Visualize and explore complex API schemas without context overload
OpenCTI	Cybersecurity / Threat Intel	Access threat intelligence from OpenCTI: reports, malware, actors, etc.
OpenDota	Gaming / Analytics	Interfaces with OpenDota API to retrieve match data, player stats, and team summaries
OpenRPC	APIs / JSON-RPC Tools	Discovers and interacts with JSON-RPC APIs using OpenRPC definitions
Open Strategy Partners	Marketing / Tools	Offers value-mapping, positioning, and editing tools for product marketing
Pandoc	Document Conversion	Converts documents across formats (PDF, DOCX, HTML, etc.) using Pandoc
PIF (Personal Intelligence Framework)	Agent Memory / Continuity	Personal AI framework with journaling, file access, and structured reasoning
Pinecone	Vector Database / RAG	Upload, search, and manage vector records in Pinecone via MCP interface
Placid.app	Creative Media / Templates	Generates images and video content using predefined Placid.app templates
Playwright	Automation / Web Browsing	Automates browser interactions and scraping tasks using Playwright
Postman	API Testing / Local Execution	Runs Postman collections locally via Newman with pass/fail result feedback
Productboard	Product Management	Integrates Productboard for feedback collection and roadmap planning
Prometheus	Monitoring / Metrics	Allows querying and evaluating time-series metrics from Prometheus
Pulumi	Infrastructure as Code	Manages cloud infrastructure stacks and projects using Pulumi's API
Pushover	Notifications / Messaging	Sends instant notifications to user devices via Pushover.net
QGIS	Geospatial Tools / Mapping	Adds map layers, creates geospatial projects, and runs code in QGIS
QuickChart	Visualization / Charting	Generates charts using VegaLite or QuickChart.io from structured data
Qwen_Max	LLM / Inference API	Interfaces with Qwen language models to run advanced inference via MCP
RabbitMQ	Messaging Queue / Broker	Publishes and consumes messages from RabbitMQ queues and exchanges
RAG Web Browser	Web Search / RAG	Uses Apify's open-source tools to search and scrape URLs for Markdown outputs
Reaper	Audio / DAW Tools	Interfaces with Reaper DAW to manage projects, tracks, and sessions
Redis	In-Memory / Caching DB	Performs key-value operations and caching using Redis or AWS MemoryDB
Rememberizer AI	Memory / Knowledge Base	Enables intelligent recall and LLM-based queries from the Rememberizer store
Replicate	ML Model Interface	Runs predictions, tracks output, and browses available models on Replicate
Rquest	Web / HTTP Emulation	Makes realistic browser-grade HTTP requests with true client fingerprinting
Rijksmuseum	Art / Open Collections	Interfaces with Rijksmuseum API to explore artworks and metadata

Server Name	Domain	Purpose
Salesforce MCP	CRM / Enterprise Data	Enables querying of Salesforce metadata, objects, and customer records
Scholarly	Academic Research / Search	Finds research articles and scholarly references via structured queries
scrapling-fetch	Scraping / Anti-bot Sites	Fetches content from heavily protected websites with anti-automation guards
SearXNG	Web Search / Federated Engine	Aggregates search results from multiple engines using SearXNG
SEC EDGAR (by Stefano Amorelli)	Finance / Filings Data	Retrieves company filings, reports, and metadata from the SEC EDGAR database for financial analysis
ServiceNow	ITSM / Enterprise Workflows	Connects to ServiceNow for ticketing, workflow queries, and automation
Shopify	E-Commerce / Storefronts	Interacts with Shopify API to manage orders, products, and customer data
Siri Shortcuts	macOS / Automation	Calls local Siri Shortcuts for task execution and workflow control
Snowflake	Databases / Cloud SQL	Runs secure queries and explores structured data on Snowflake
Solana Agent Kit	Blockchain / Solana	Executes smart contract calls and blockchain actions on Solana
Spotify	Audio / Streaming	Plays tracks, retrieves playlists, and interacts with Spotify via MCP
Starwind UI	Design System / Astro	Provides commands and docs for working with Starwind UI's Astro components
Stripe	Payments / Finance API	Integrates Stripe payments, refunds, and customer data handling
ShaderToy	Visuals / Compute Shaders	Allows creation and modification of GLSL shaders using ShaderToy API
TMDB	Movies / API	Integrates with The Movie Database (TMDB) to provide movie info, search, and recommendations
Tavily Search	Web / Semantic Search	Searches web and news content with filter controls and site whitelisting
Telegram	Messaging / Chatbot	Sends, retrieves, and manages Telegram messages using Telethon API
Terminal-Control	OS / Command Line	Executes shell commands securely and navigates filesystem structures
TFT-Match-Analyzer	Games / Match History	Analyzes Teamfight Tactics match data with detailed stats
Ticketmaster	Events / Entertainment	Searches events, venues, and shows using the Ticketmaster Discovery API
Todoist	Productivity / Task Manager	Manages tasks and to-do items in Todoist via structured LLM prompts
Typesense	Search / Indexing Engine	Provides search, filtering, and discovery over Typesense data collections
Travel Planner	Mapping / Routing	Plans trips and builds itineraries using Google Maps location services
Unity Catalog	Data Governance / Databricks	Executes Unity Catalog operations and accesses metadata functions
Unity3d Game Engine	Game Dev / Editor Tools	Provides interaction with Unity3d editor state, console, and scene assets
Unity Integration (Advanced)	Game Engine / Scripting	Allows direct code execution, file access, and debug log inspection in Unity
Vega-Lite	Visualization / Charting	Generates dynamic charts using the VegaLite visualization grammar
Video Editor	Multimedia / Editing Tools	Adds, searches, edits, and manages videos for creative pipelines
Virtual Location	Geo Sim / Visual Search	Simulates locations via Google Maps, Street View, and AI image APIs
VolcEngine TOS	Cloud Storage / TOS Access	Fetches objects from VolcEngine's object storage buckets
Wanaku MCP Router	Enterprise Integration / Routing	Provides SSE-based dynamic routing of agent requests across enterprise tools
Webflow	Website Design / API	Publishes and manages websites via the Webflow API interface
whale-tracker-mcp	Crypto / Large Transactions	Monitors and reports large on-chain movements (whale tracking)
Whois MCP	Internet / Domain Tools	Looks up WHOIS data for domains, IPs, ASNs, and top-level domains
Wikidata MCP	Semantic Web / Knowledge Graph	Queries metadata, identifiers, and SPARQL endpoints in Wikidata
WildFly MCP	Server Management / Logs	Interacts with WildFly app servers for logs, operations, and metrics
Windows CLI	OS / Shell Access	Securely executes shell commands in PowerShell, CMD, and Git Bash
World Bank data API	Economics / Open Data	Queries global indicators and economic data from the World Bank API

Server Name	**Domain**	**Purpose**
X (Twitter) (by EnesCinr)	Social / Tweeting	Enables posting and searching tweets via Twitter/X API
X (Twitter) (by vidhupv)	Twitter / Claude Chat	Manage and publish X/Twitter content directly from Claude interface
xcodebuild	iOS / Build Tools	Builds Xcode projects/workspaces and returns feedback to LLMs
Xero-mcp-server	Accounting / ERP	Integrates Xero for invoices, business operations, and financial data
XiYan	SQL / Natural Language Query	Translates natural queries into SQL using XiYanSQL for DB retrieval
XMind	Productivity / Mind Maps	Searches and reads XMind-format files from designated directories
YouTube	Video / API Access	Full YouTube API access: manage videos, Shorts, analytics, and uploads

TABLE X: List of Official MCP Servers

Official MCP Server	Domain	Description
21st.dev Magic	UI/UX Components	Crafted UI components for seamless front-end development
Adfin	Finance / Payments	Platform for managing payments, invoicing, and reconciliations
AgentQL	Web Extraction	Structured data extraction from unstructured web pages
AgentRPC	RPC / DevOps	Enables remote function execution across languages and systems
Aiven	Data Infrastructure	Control PostgreSQL®, Kafka®, ClickHouse®, and OpenSearch®
Apache IoTDB	IoT / Database	Access IoT time-series database with analytics capabilities
Apify	Web Scraping / RPA	Use 3,000+ pre-built tools to extract web data
APIMatic MCP	API Dev Tools	Validates and interprets OpenAPI specs using APIMatic APIs
Audiense Insights	Marketing Analytics	Provides audience segmentation and influencer analysis
Axiom	Observability	Query logs, traces, and telemetry using natural language
Bankless Onchain	Blockchain / DeFi	Query ERC20 tokens, contract states, and transaction history
BICScan	Blockchain Security	Risk scoring and asset analysis for blockchain addresses
Box	Content Management	Access and analyze files via Box's intelligent platform
Browserbase	Cloud Automation	Cloud-hosted browser control for scraping, navigation, etc.
Chargebee	Billing	Subscription billing platform accessible via MCP
Chroma	Vector DB / Search	Embedding management and semantic search
Chronulus AI	Forecasting / AI	AI-based prediction and forecasting server
CircleCI	DevOps / CI-CD	Debug and resolve build failures programmatically
ClickHouse	Database / Analytics	Execute analytic queries on ClickHouse DB
Cloudflare	Cloud Services	Manage Cloudflare resources (Workers, KV, R2, etc.)
CodeLogic	Software Intelligence	Understand software dependencies and code architecture
Comet Opik	Observability / Logs	Analyze LLM traces, telemetry, and logs
Convex	App Runtime	Inspect apps hosted on Convex serverless backend
Dart	Project Management	Access project, task, and documentation data
DevHub	CMS	Manage and publish content via DevHub CMS
E2B	Secure Sandboxing	Run code securely in isolated cloud sandboxes
EduBase	Education / LMS	Exam, quiz, and learning content platform
Elasticsearch	Search Engine	Query data in Elasticsearch clusters
eSignatures	Legal / Contracts	Manage contract templates, signing, and review workflows
Exa	Search / AI Tools	Search engine designed for AI models
Fewsats	Payments / Crypto	Secure purchasing by AI agents using crypto
Fibery	Business Operations	Query workspace entities and relations
Financial Datasets	Stock Data	Financial APIs tailored for LLMs
Firecrawl	Web Data	AI-driven website scraping tool
Fireproof	Ledger / Blockchain	Immutable ledger DB with live sync
Gitee	Developer Tools	Gitee repository and project management
gotoHuman	Human-in-the-loop	Route AI actions for manual approval
Grafana	Monitoring / Dashboards	Search dashboards, incidents, and datasources
Graphlit	Multimodal Ingest	Ingest Slack, Gmail, RSS, and podcast feeds
GreptimeDB	Time-Series DB	Query and explore GreptimeDB with structured LLM prompts
Heroku	App Platform	Manage Heroku apps, dynos, add-ons, and databases
Hologres	OLAP DB	Metadata, querying, and analytics on Hologres
Hyperbrowser	Browser Automation	Automate web tasks at scale using AI
IBM wxflows	Tool Dev	Build, test, deploy tools for any data source
ForeverVM	Secure Execution	Python sandbox environment for agent tasks
Inbox Zero	Email Management	AI assistant for decluttering and organizing email
Inkeep	RAG Search	Retrieval-augmented generation over content
Integration App	SaaS Integration	Interact with 3rd-party SaaS platforms
JetBrains	IDE Tools	Code assistance using JetBrains IDEs
Kagi Search	Search Engine	Web search via Kagi's privacy-focused API
Keboola	Data Workflow	Build, run, and analyze complex data workflows
Lara Translate	Translation	Translation with language detection and context awareness
Logfire	Monitoring / Metrics	Access OpenTelemetry logs and metrics

MCP Server	Domain	Description
Langfuse	Prompt Management	Versioning, evaluating, and releasing LLM prompts
Lingo.dev	Localization	LLM localization across all supported languages
Mailgun	Email API	Email send/receive integration for agents
Make	Workflow Automation	Integrate Make scenarios as callable tools
Meilisearch	Search / Indexing	Query Meilisearch for fast full-text or semantic queries
Metoro	Kubernetes	Inspect Kubernetes clusters and environments
Milvus	Vector DB	Vector storage, indexing, and retrieval
MotherDuck	SQL / DuckDB	Query DuckDB using MotherDuck server
Needle	Document RAG	Production-ready search and retrieval on documents
Neo4j	Graph DB	Schema browsing, Cypher query, graph memory
Neon	Serverless DB	Interact with Neon's Postgres-based DB platform
OceanBase	Relational DB	Query OceanBase database instances
Octagon	Financial Analytics	Research on public/private market data
Oxylabs	Web Scraping	Advanced scraping with rendering/parsing support
Paddle	Billing / SaaS	Billing and catalog management for SaaS
PayPal	Payments	Access and control PayPal integrations
Perplexity	Conversational Search	Real-time web research via Sonar API
Qdrant	Vector Search	Build semantic memory with Qdrant vector engine
Ramp	Expense Analytics	Analyze company spend using Ramp API
Raygun	Monitoring / DevOps	View crash reports and diagnostics
Rember	Memory Tools	Spaced repetition flashcard generator
Riza	Tool Execution	Code and tool invocation platform
Search1API	Web Crawling	All-in-one API for search, crawling, sitemap access
ScreenshotOne	Screenshot API	Generate rendered screenshots of web pages
Semgrep	Code Security	Analyze code and apply security policies
SingleStore	Analytical DB	Query SingleStore databases
StarRocks	OLAP DB	High-performance analytical engine access
Stripe	Finance / Payments	Access Stripe API for transactions and invoices
Tavily	Search Engine	Semantic search and information extraction
Thirdweb	Blockchain	Read/write contracts across 2,000+ blockchains
Tinybird	Real-Time SQL	ClickHouse-powered real-time analytics engine
UnifAI	Tool Aggregator	Unified tool search and invocation interface
Unstructured	Document Parsing	Pre-process unstructured content for LLMs
Vectorize	Retrieval + Markdown	AI-ready retrieval, file extraction, markdown output
Verodat	Data Analytics	Work with Verodat's AI-ready datasets
VeyraX	API Aggregator	Single interface to access 100+ APIs
Xero	Accounting / ERP	Interact with Xero financial data
Zapier	Automation	Connect to 8,000+ apps for workflow automation
ZenML	MLOps / LLMOps	Query, manage, and interact with ML pipelines