Model Context Protocol (MCP): A Comprehensive Literature Review

Table of Contents

Introduction and Definition

The Model Context Protocol (MCP) is an open standard developed by Anthropic aimed at revolutionizing how large language models (LLMs) interact with external data sources and tools. As AI systems become more integrated into real-world applications, the need for standardized, scalable, and secure methods of connecting these models to diverse data repositories has become critical. MCP addresses this by providing a universal, open protocol that facilitates seamless, two-way communication between AI applications and external data sources, thereby enhancing the relevance, reliability, and functionality of AI-powered systems [1].

MCP serves as a universal connector—akin to a "USB-C port for AI"—allowing different systems to connect without requiring bespoke integrations for each data source [2]. The protocol enables secure, context-aware, and dynamic interactions, allowing AI models to read data, execute actions, and maintain context across multiple tools and datasets.

The primary goal of MCP is to replace the fragmented, custom integrations that currently hinder scalable, context-aware AI systems with a universal, open

standard that simplifies connectivity and enhances reliability [3]. By standardizing how AI models access and interact with external data sources and tools, MCP aims to break down data silos and promote interoperability across diverse systems.

Technical Details and Implementation

Core Architectural Components

The MCP architecture is built on a client-host-server model, where [4]:

- **Hosts** are LLM applications (e.g., Claude Desktop, AI IDEs) that initiate and manage connections.
- **Clients** are internal connectors within hosts that maintain individual sessions with servers.
- **Servers** are external or local services exposing resources, prompts, and tools.

This architecture supports multiple concurrent client instances managed by a host, with each client maintaining a dedicated, stateful connection to a server. The modular design enables developers to build and deploy connectors independently, promoting scalability and security [5].

Protocol Specification

MCP employs **JSON-RPC 2.0** as its message format, supporting three primary message types [4]:

1. **Request**: Bidirectional, expects a response, includes a unique ID.

```
{
  "jsonrpc": "2.0",
  "id": "123",
  "method": "listResources",
  "params": {}
}
```

2. **Response**: Replies to requests, includes the same ID, and contains either a result or an error.

```
{
  "jsonrpc": "2.0",
  "id": "123",
  "result": {
    "resources": [
      {"uri": "file://path/to/file", "name": "File Resource"}
    ]
  }
}
```

3. **Notification**: One-way message, does not require a response, used for status updates or events.

```
{
  "jsonrpc": "2.0",
  "method": "resourceUpdated",
  "params": {
    "uri": "file://path/to/file"
  }
}
```

The protocol lifecycle consists of three main phases [4]:

1. **Initialization**
   - The client initiates connection with an `initialize` request, exchanging protocol version, capabilities, and client info.
   - The server responds with its supported version and capabilities.
   - An `initialized` notification confirms readiness.
2. **Operation**
   - Capabilities such as resources, prompts, tools, and sampling are negotiated.
   - Clients invoke server methods (e.g., resource retrieval, tool execution).
   - Servers can send notifications for resource updates or status changes.
3. **Shutdown**
   - Clients or servers send a disconnect notification.
   - Resources are cleaned up, and connections are gracefully terminated.

Transport Mechanisms

MCP supports multiple transport layers [4]:

- **Stdio**: For local, process-to-process communication, using standard input/output streams.
- **HTTP SSE (Server-Sent Events)**: For remote, web-based interactions, supporting bidirectional communication via HTTP endpoints.
- **Custom transports**: Protocols can be extended to other channels, provided they preserve JSON-RPC message integrity.

Security and Trust Principles

Given MCP's powerful capabilities, security is paramount [4]:

- **User Consent & Control**: All data access and tool invocation require explicit user approval.
- **Data Privacy**: Sensitive resources are protected with access controls; data is not transmitted without consent.
- **Tool Safety**: Tools, which may execute arbitrary code, are invoked only after user approval, with clear descriptions.

- **Sampling Controls**: Users must authorize any LLM sampling requests, with options to control prompt visibility and server access.

Implementors are encouraged to: - Implement robust authentication and authorization flows. - Validate all incoming messages against schemas. - Use TLS for secure transport. - Log and monitor protocol activity for security auditing.

Implementation Details

The protocol schemas are primarily defined in **TypeScript**, ensuring type safety and clarity. These schemas are also available as **JSON Schema** for broader compatibility [4].

Core classes and interfaces include: - **Protocol**: Manages message handling, request/response lifecycle, and connection state. - **Transport**: Abstracts communication channels; implementations include stdio and HTTP SSE. - **Client/Server**: Encapsulate protocol-specific logic for initiating requests, handling responses, and managing capabilities.

Example server implementation in TypeScript [4]:

```
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";

const server = new Server({ name: "ExampleServer", version: "1.0.0" }, {
  capabilities: {
    resources: {}
  }
});

server.setRequestHandler('listResources', async () => {
  return {
    resources: [
      { uri: "example://resource", name: "Example Resource" }
    ]
  };
});

const transport = new StdioServerTransport();
await server.connect(transport);
```

Example client implementation in Python [4]:

```
import asyncio
from mcp.server import Server
from mcp.server.stdio import stdio_server

async def main():
    async with stdio_server() as streams:
```

```
        server = Server("example-client")
        await server.run(streams[0], streams[1])
        await server.request('listResources', {})

if __name__ == "__main__":
    asyncio.run(main())
```

Anthropic provides SDKs in multiple programming languages—TypeScript, Python, Java, Kotlin, C#—to facilitate the development of MCP servers and clients [5]. These SDKs support various transport mechanisms, ensuring flexibility for local and remote deployments.

Benefits and Performance Improvements

Enhanced Performance and Efficiency

MCP significantly improves AI models' ability to retain and utilize extended contextual information through hierarchical memory management, contextual chunking, and adaptive retrieval mechanisms, which collectively reduce information loss over long interactions [6].

By employing token compression and selective recall, MCP optimizes token utilization, enabling models to process longer sequences without exceeding token limits, thus maintaining response relevance and coherence [6].

Efficient context management and retrieval reduce computational overhead, leading to quicker AI responses, especially in complex multi-turn conversations or data-intensive tasks [6].

Improved Data Integration and Interoperability

MCP provides a uniform API for connecting AI models with diverse external systems, databases, and tools, eliminating the need for bespoke integrations. This standardization accelerates deployment and reduces integration costs [3, 7].

As an open standard, MCP fosters interoperability across different AI providers and enterprise systems, avoiding vendor lock-in and enabling flexible, multi-vendor deployments [3].

Increased Security and Control

MCP enforces strict access controls, requiring explicit user or administrator approval for tool invocation and data access. Sensitive credentials remain within enterprise-controlled MCP servers, preventing exposure to external AI providers [3].

The protocol's logging and monitoring capabilities facilitate compliance, troubleshooting, and continuous improvement, which are critical for enterprise adoption [3].

Cost Reduction and Scalability

Standardized connectors and reusable components reduce development and maintenance efforts, cutting integration costs by approximately 30% [6].

MCP's modular design supports rapid scaling across multiple systems and data sources, accommodating enterprise growth and evolving data landscapes [3].

Limitations and Challenges

Despite its advantages, MCP faces several challenges:

1. **Adoption Barriers**: As a relatively new standard, MCP requires industry-wide adoption to realize its full potential. Organizations may be hesitant to invest in implementing MCP until it becomes more established [8].
2. **Implementation Complexity**: While MCP simplifies integration at scale, initial implementation requires technical expertise and resources, potentially limiting adoption by smaller organizations or teams [8].
3. **Security Concerns**: The ability to access external data sources and execute tools introduces potential security vulnerabilities if not properly implemented and monitored [4].
4. **Performance Overhead**: The additional layer of abstraction introduced by MCP may introduce some performance overhead, particularly in latency-sensitive applications [8].
5. **Standardization Challenges**: As with any open standard, ensuring consistent implementation across different vendors and platforms remains a challenge [8].

Comparisons with Other Approaches

MCP vs. Large-Context Window Techniques

Large-context window models like Anthropic's Claude (up to 200K tokens), GPT-4 Turbo (128K tokens), and Google Gemini 1.5 pro (2 million tokens) offer unprecedented capacity to process extensive data internally [9]. However, they face challenges related to cost, information retention, and the "lost in the middle" problem [10].

While large-context windows allow for ingestion of entire books or large document collections in a single prompt, they still face issues with effectively utilizing information buried in the middle of long inputs [10]. Additionally, inference costs increase linearly with input size, impacting efficiency [10].

In contrast, MCP provides a more scalable and cost-effective approach by enabling AI models to access external data sources on-demand, without the need to include all relevant information in the prompt [5].

MCP vs. RAG Approaches

Retrieval-Augmented Generation (RAG) involves retrieving relevant document chunks from external knowledge bases (via vector similarity search) and appending them to prompts for the LLM [11]. RAG is cost-effective, extendable, and capable of handling dynamic, frequently updated knowledge bases [11].

However, RAG faces challenges related to chunking complexity, relevance ranking, and reranking, which are crucial to avoid irrelevant or noisy data that can impair response quality [10].

MCP can be seen as complementary to RAG, providing a standardized way for AI models to access and interact with RAG systems and other external data sources [5]. By combining MCP with RAG techniques, organizations can achieve both scalability and accuracy in knowledge-intensive applications.

MCP vs. Other Integration Methods

Aspect	MCP	ChatGPT Plugins	Manual API Integration
Standardization	Fully standardized, language-agnostic, JSON-RPC based	Proprietary, plugin-specific	Ad hoc, API-specific code
Extensibility	Modular, supports multiple SDKs, capability negotiation	Plugin marketplace, limited to supported APIs	Custom code per integration

Aspect	MCP	ChatGPT Plugins	Manual API Integration
Security	Explicit user consent, capability negotiation, resource controls	User approval, plugin sandboxing	Developer-managed, less standardized
Interoperability	Cross-language, multi-platform	Limited to ChatGPT ecosystem	Varies, often complex
Ease of Use	SDKs and templates for rapid server/client development	User-friendly plugin interface	Manual coding, error-prone
Use Cases	Multi-tool, multi-resource, agent workflows	ChatGPT-specific, limited to supported APIs	Custom, one-off integrations

MCP offers a comprehensive, standardized, and extensible framework for integrating external tools and data sources into AI workflows, surpassing proprietary or ad hoc methods in interoperability, security, and scalability [5].

Current Applications and Case Studies

Banking Sector - Rapid Data-Driven Decision Making

A leading bank implemented MCP to connect its legacy transaction systems with AI-powered risk assessment tools. The integration enabled real-time fraud detection and risk analysis, reducing analysis time by 50% and false positive rates by 7%. The standardized MCP connectors facilitated quick deployment and

seamless data access, leading to improved operational efficiency and customer trust [6].

Healthcare - Accelerated Diagnostics and Patient Data Management

A hospital network adopted MCP to securely integrate patient records, lab results, and imaging data with AI diagnostic models. The result was a 20% improvement in diagnostic turnaround times and enhanced patient care. The protocol's security features ensured compliance with healthcare regulations, while its long-context retention capabilities allowed AI to consider comprehensive patient histories [6].

Retail - Personalized Customer Engagement

A major retail chain used MCP to connect real-time sales, inventory, and customer feedback systems with AI-driven personalization engines. This integration led to a 15% increase in sales through dynamic demand forecasting and targeted promotions. The reusable connectors reduced development time and enabled rapid iteration of AI features [6].

Enterprise IT - Knowledge Management and Internal Support

A multinational corporation deployed MCP to integrate its legacy ERP and knowledge base systems with AI chatbots. The result was a 40% reduction in employee search times and a 50% decrease in routine query handling time. The structured, secure access to enterprise data improved AI accuracy and user satisfaction [7].

Future Directions

The MCP project is actively evolving, with plans including [5]:

1. **Registry for server discovery**: Facilitating easier discovery and connection to available MCP servers.
2. **Enhanced agent workflows**: Supporting more complex, multi-step workflows involving multiple tools and data sources.
3. **Multimodal support**: Extending the protocol to handle video, streaming, and other modalities beyond text.
4. **Community-led governance**: Establishing a governance model to ensure the standard evolves in a way that meets the needs of the broader community.
5. **Compliance and testing suites**: Developing tools to verify compliance with the MCP standard and ensure interoperability.

Anthropic's Contributions to MCP

Anthropic has played a pivotal role in the development and promotion of MCP:

1. **Initial Development**: Anthropic developed and open-sourced the Model Context Protocol as a standard for connecting AI assistants to data systems [3].
2. **SDK Development**: Anthropic has provided SDKs in multiple programming languages to facilitate the development of MCP servers and clients [5].
3. **Pre-built MCP Servers**: Anthropic has shared pre-built MCP servers for popular enterprise systems such as Google Drive, Slack, GitHub, Git, Postgres, and Puppeteer, accelerating adoption by providing ready-to-use implementations [3].
4. **Claude Integration**: Anthropic has integrated MCP support into Claude Desktop applications, allowing organizations and individuals to connect their internal datasets and systems directly to their AI assistants [3].
5. **Ecosystem Building**: Anthropic has worked with partners like Block, Apollo, Zed, Replit, Codeium, and Sourcegraph to build an ecosystem around MCP [3].
6. **Open-Source Commitment**: Anthropic has emphasized the open-source nature of MCP, inviting contributions from the community and industry partners to ensure the protocol evolves to meet diverse needs [3].

References

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

[2] Beebom. (2025). "Model Context Protocol (MCP) Explained". https://beebom.com/model-context-protocol-mcp-explained/

[3] Anthropic. (2024). "Model Context Protocol (MCP) - Documentation". https://docs.anthropic.com/en/docs/agents-and-tools/mcp

[4] "Technical Details of the Model Context Protocol (MCP) Implementation". https://modelcontextprotocol.info/specification/

[5] "Research Report: Technical Specification and Implementation Comparison of the Model Context Protocol (MCP)". https://github.com/modelcontextprotocol/modelcontextprotocol

[6] "How Model Context Protocol Boosts AI's Long-Context Retention". https://www.amplework.com/blog/model-context-protocol-ai-long-context-retention/

[7] Kumar, P. (2025). "Building Better AI Integrations with Model Context Protocol: A Confluence Case Study". https://pawan-kumar94.medium.com/building-better-ai-integrations-with-model-context-protocol-a-confluence-case-study-ef502364369e

[8] “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/

[9] “Comparative Analysis of Large-Context Window Techniques and RAG Approaches in Anthropic Claude”. https://docs.anthropic.com/en/docs/build-with-claude/context-windows

[10] Medium. (2025). “Long-Context Window Models vs. RAG”. https://medium.com/@jm_51428/long-context-window-models-vs-rag-a73c35a763f2

[11] DevThink. (2025). “RAG vs. Large Context Windows: A Comparison”. https://devthink.ai/p/rag-vs-large-context-windows-a-comparison