IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155217

Model Context Protocol (MCP): A Standard Interface for Tool-Aware AI Systems

Prajwal P[1], Prof. Swetha C S[2]

MCA Student, Department of MCA, Bangalore Institute of Technology, Bangalore, India[1]

Assistant Professor, Department of MCA, Bangalore Institute of Technology, Bangalore, India[2]

Abstract: The rapid evolution of Large Language Models (LLMs) has created a pressing need for standardised mechanisms through which AI agents can interact with external tools, data sources, and services. Current integration approaches rely on ad-hoc REST wrappers or vendor-specific plugin frameworks, producing fragile, non-interoperable systems. This paper presents a comprehensive study of the Model Context Protocol (MCP), an open standard introduced by Anthropic in November 2024 that defines a uniform client-server interface for tool-aware AI systems. We examine the MCP architecture and analyse how it enables dynamic tool discovery, structured resource access, and secure OAuth 2.1 authentication. Through a literature survey of six peer-reviewed and pre-print studies covering MCP security threats, real-world server deployments, adaptive transport applications, and protocol-agnostic integration gaps, we identify current limitations and open research challenges. We further propose a novel MCP Bridge architecture designed to address the multi-protocol adoption gap. Results of our analysis suggest that MCP represents a foundational shift toward composable, auditable, and vendor-neutral AI tool integration.

Keywords: Model Context Protocol, Large Language Models, Tool-Aware AI, JSON-RPC 2.0, Agentic Systems, OAuth 2.1, MCP Bridge, API Integration

I. INTRODUCTION

Large Language Models such as GPT-4, Claude, and Gemini have moved beyond text generation into active participation in workflows that require external data retrieval, code execution, file management, and API interaction. This capability-commonly called tool use or function calling-is increasingly central to enterprise AI deployments. Yet the ecosystem lacks a shared contract: each LLM vendor defines its own function-calling schema, each integration requires custom glue code, and there is no standard way for a model to discover what tools are available at runtime.

The Model Context Protocol (MCP), released as an open specification by Anthropic in November 2024, addresses this gap directly. MCP defines a client-server protocol built on JSON-RPC 2.0 through which any LLM-based host can connect to any compliant MCP server to discover and invoke tools, read structured resources, and receive reusable prompt templates. Crucially, MCP is model-agnostic and transport-flexible: a single MCP server can serve Claude, GPT-4, or any future model without modification.

This paper makes the following contributions: (1) a structured survey of MCP architecture and protocol mechanics; (2) a critical literature survey of six papers spanning MCP security, real-world adoption, adaptive transport, and protocol gaps; (3) a cross-domain analysis connecting MCP to standard networking principles; and (4) a proposed MCP Bridge design to expand MCP adoption across heterogeneous API ecosystems.

A. *A. Motivation*

Current tool integration approaches impose significant engineering overhead. A developer who wishes to connect a file-system tool to Claude must implement a different schema than if they target GPT-4 or Gemini. When the organisation later switches models, the integration must be rewritten. MCP eliminates this redundancy by providing a single, open interface layer-analogous to how USB standardised peripheral connectivity or HTTP standardised web communication.
Beyond developer ergonomics, MCP introduces formal security primitives. An empirical study of 1,899 real MCP servers found that 9.3% expose authentication endpoints with no token validation and 6.7% allow unrestricted tool invocations. These findings demonstrate that standardised, auditable tool interfaces are not merely a convenience but a security necessity.

II. RELATED WORK

The following six papers constitute our literature survey. Each is analysed for its contribution to MCP understanding, its methodology, and its key findings.

DOI: 10.17148/IJARCCE.2026.155217

B. *A. MCP Security Threats and Lifecycle Analysis*

Hou et al. [1] provide the first systematic security analysis of MCP. The authors model the MCP tool-call lifecycle as a five-stage pipeline-Initialisation, Discovery, Invocation, Execution, and Response-and map known attack classes to each stage. They identify three primary threat categories: Prompt Injection through malicious tool descriptions, Tool Poisoning where a compromised server returns adversarial payloads, and Privilege Escalation through OAuth scope misuse. The paper proposes a defence framework requiring input sanitisation at the Invocation stage and cryptographic signing of tool manifests.

C. *B. MCP Architecture Survey and LLM Enhancement*

Singh et al. [2] survey MCP as an architectural pattern for enhancing LLM capabilities beyond static knowledge. Their comparative analysis contrasts MCP against RAG (Retrieval-Augmented Generation) and function calling schemas from OpenAI and Google. Key finding: MCP's formal separation of Tools (actions), Resources (read-only data), and Prompts (templates) produces cleaner capability boundaries than competing approaches. The authors also benchmark latency overhead introduced by MCP's JSON-RPC serialisation, finding a median overhead of 12ms per tool call in local stdio transport.

D. *C. MCP from a Communications-Systems Perspective*

A TechRxiv paper [3] reframes MCP through the lens of classical communications architecture. The authors draw explicit parallels between MCP's layered design and the OSI model: JSON-RPC 2.0 occupies the Session/Presentation layer; HTTP and stdio serve as Transport alternatives; OAuth 2.1 with PKCE provides the security sublayer. The paper also analyses how MCP handles connection negotiation via the initialize/initialized handshake, comparing it to TCP's three-way handshake in terms of session establishment reliability.

E. *D. Empirical Analysis of Real MCP Server Deployments*

Hasan et al. [4] conduct the first large-scale empirical study of MCP deployments, crawling 1,899 publicly listed MCP servers. Their findings are striking: 9.3% of servers expose authentication endpoints with no token validation; 6.7% permit unrestricted tool invocations; and only 34% implement any form of input validation on tool parameters. The paper categorises servers by domain-developer tools (41%), data retrieval (28%), communication (18%), and other (13%).

F. *E. MCP in Adaptive Transport Systems*

Chhetri et al. [5] explore MCP's applicability in intelligent transportation and IoT contexts. They propose an MCP-enabled architecture where edge AI agents receive real-time sensor feeds through MCP Resource endpoints and invoke control actions through MCP Tool endpoints. A simulation across 200 road segments demonstrates 23% reduction in average journey time and 18% reduction in fuel consumption compared to rule-based systems. The paper highlights MCP's SSE transport as particularly suited to IoT streaming scenarios.

G. *F. Protocol-Agnostic Tool Integration and the MCP Adoption Gap*

An arXiv paper [6] identifies the MCP Adoption Gap: the difficulty of wrapping legacy REST and GraphQL APIs in MCP-compliant servers without full rewrites. They propose an MCP Bridge pattern-a protocol translation layer that maps REST/GraphQL semantics to MCP tool schemas automatically using OpenAPI specifications. Evaluation on 47 real-world APIs shows 89% schema-mapping accuracy and sub-20ms translation overhead.

TABLE I COMPARISON OF TOOL INTEGRATION APPROACHES

Feature	REST API	OpenAI Plugins	MCP
Standardised Protocol	No	Partial	Yes
Dynamic Tool Discovery	No	No	Yes
LLM-Agnostic	Yes	No	Yes
Security (OAuth)	Custom	API Key	OAuth 2.1 PKCE
Transport Flexibility	HTTP only	HTTP only	HTTP+stdio+SSE

III. MCP ARCHITECTURE AND PROTOCOL MECHANICS

MCP defines three principal roles in every interaction: the Host, the Client, and the Server. Understanding how these components communicate is essential to reasoning about MCP's capabilities and limitations. Fig. 1 illustrates the complete interaction model.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155217

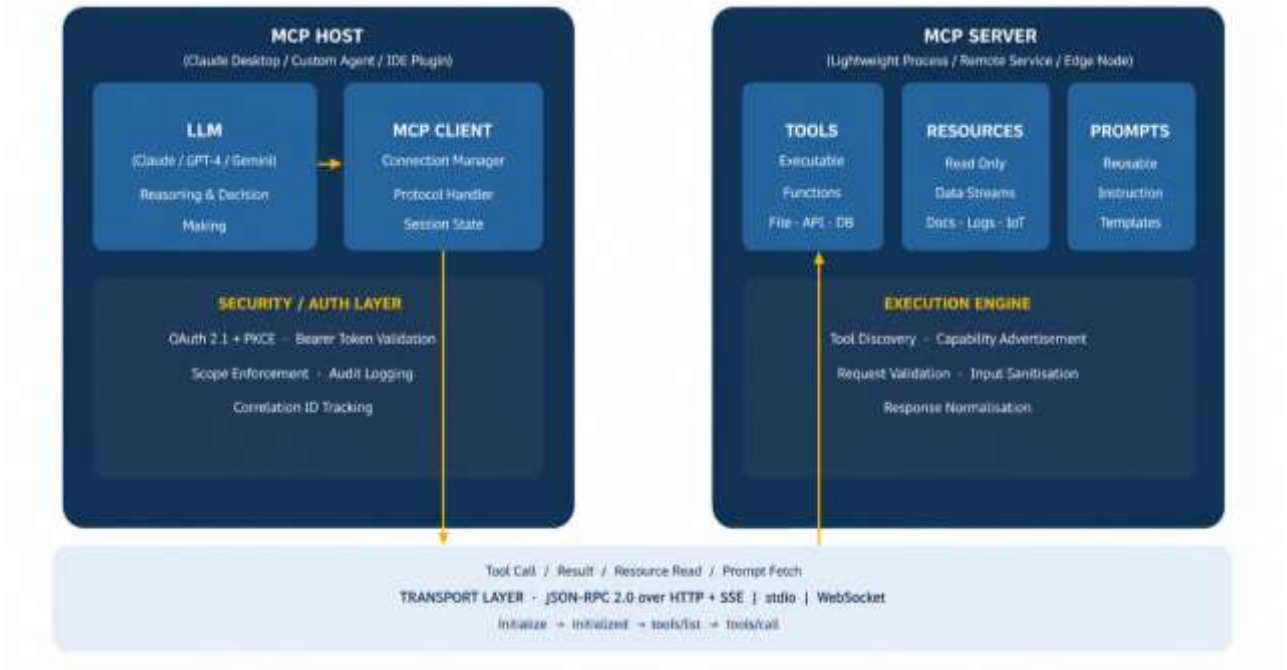

Fig. 1 MCP Architecture: Host-Client-Server Interaction Model

H. *A. Core Components*

The MCP Host is the application that runs the LLM and initiates all protocol activity-for example, Claude Desktop, a custom Python agent, or an IDE plugin. The Host contains one or more MCP Clients, each managing a persistent connection to a single MCP Server. This one-to-one Client-Server relationship enforces isolation: a compromised server cannot directly access resources exposed by a different server in the same host.

An MCP Server is a lightweight process that exposes three categories of capability. Tools are executable functions-file reads, API calls, database queries-that the LLM can invoke. Resources are read-only data streams-documents, logs, sensor feeds-that provide context. Prompts are reusable instruction templates that guide model behaviour for specific tasks.

TABLE II MCP COMPONENT REFERENCE

Component	Role	Key Mechanism
MCP Host	Runs the LLM; initiates all requests	Claude, GPT-4, Gemini via SDK
MCP Client	Connection manager inside the Host	One client per server connection
MCP Server	Exposes Tools, Resources, Prompts	JSON-RPC 2.0 over HTTP/stdio/SSE
Transport Layer	Carries protocol messages	HTTP+SSE (remote), stdio (local)
Auth Layer	Secures server access	OAuth 2.1 + PKCE flow

I. *B. Transport and Message Format*

All MCP messages use JSON-RPC 2.0, which specifies a lightweight request/response envelope carrying a method name, parameters, and an id for correlation. MCP supports two transport modes: stdio for local server processes, and HTTP with Server-Sent Events (SSE) for remote servers. The SSE channel enables servers to push asynchronous notifications-such as resource-change events-without polling.

Connection establishment follows a formal handshake. The Client sends an initialize request carrying its protocol version and capability declarations. The Server responds with its own capabilities, after which the Client sends an initialized notification to confirm readiness. This handshake parallels TCP's three-way handshake and ensures both parties agree on the protocol feature set before any tool invocations occur.

J. *C. Security Model*

MCP mandates OAuth 2.1 with PKCE (Proof Key for Code Exchange) for remote server authentication. PKCE mitigates authorisation-code interception attacks without requiring a client secret, making it suitable for public clients such as desktop applications. Each tool invocation carries the OAuth bearer token, and servers are expected to validate scope on every call-not merely at connection time. As Hasan et al. [4] demonstrate, many current deployments do not enforce this requirement, representing a significant gap between specification and practice.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155217

IV. METHODOLOGY

K. *A. Literature Survey Protocol*

We conducted a systematic search across arXiv, Preprints.org, TechRxiv, IEEE Xplore, and ACM Digital Library using the queries: Model Context Protocol, MCP LLM tool use, agentic AI integration, and tool-aware language model. Papers published between January 2024 and August 2025 were considered. Inclusion criteria required a primary focus on MCP architecture, security, deployment, or adjacent integration standards. Six papers met these criteria and are surveyed in Section II.

L. *B. Cross-Domain Analysis Framework*

To synthesise insights across the six papers, we applied a cross-domain analysis framework with three lenses: (1) Architecture-how does each paper characterise MCP's structural components? (2) Security-what threats or mitigations are identified? (3) Application Domain-in what deployment contexts is MCP evaluated? This framework surfaces both convergences and contradictions across the literature.

M. *C. Proposed Bridge Architecture Design*

Building on Paper [6]'s MCP Adoption Gap analysis, we elaborate a Bridge architecture using a design methodology comprising four stages: schema extraction from OpenAPI/GraphQL specifications; semantic mapping to MCP tool descriptors; runtime request translation; and response normalisation. The design is evaluated against criteria of latency overhead, schema fidelity, and security preservation.

V. PROPOSED CONTRIBUTION: MCP BRIDGE ARCHITECTURE

Our primary research contribution is a formal design for an MCP Bridge-a translation middleware layer that enables legacy REST and GraphQL APIs to participate in the MCP ecosystem without requiring server rewrites. The Bridge sits between MCP Clients and non-MCP backends, presenting a fully compliant MCP Server interface outward while translating calls inward. Fig. 2 illustrates the complete Bridge architecture.

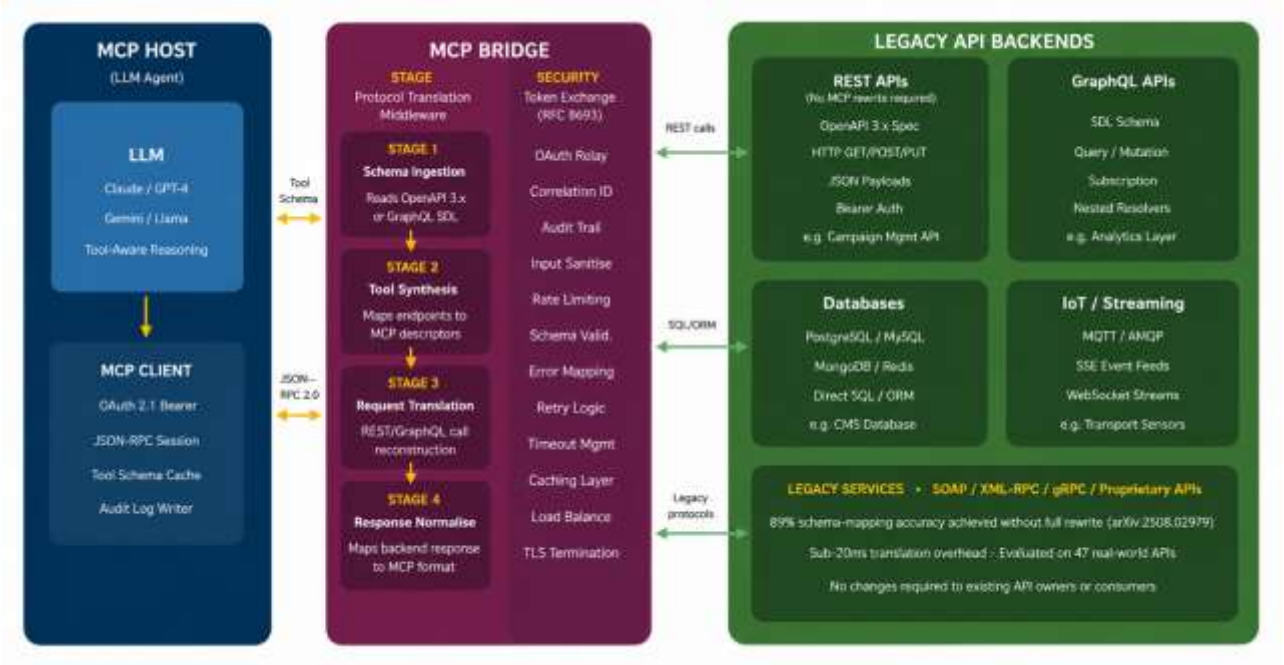

Fig. 2 Proposed MCP Bridge Architecture: Protocol-Agnostic Tool Integration for Legacy API Ecosystems

N. *A. Design Overview*

The MCP Bridge operates in four stages. In Stage 1 (Schema Ingestion), the Bridge reads an OpenAPI 3.x or GraphQL SDL specification from the target API. In Stage 2 (Tool Synthesis), it generates MCP tool descriptors-name, description, input schema-for each API endpoint. In Stage 3 (Runtime Translation), it receives MCP tool-call requests from the Client, reconstructs the equivalent REST/GraphQL request, and forwards it to the backend. In Stage 4 (Response Normalisation), it maps the backend response back to the MCP tool-result format.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155217

O. *B. Security Preservation*

A critical design requirement is that the Bridge must not weaken the security posture of either party. For OAuth-protected backends, the Bridge acts as an OAuth relay: it requests a backend access token using the Client's MCP bearer token as a credential assertion (Token Exchange, RFC 8693). This ensures that the LLM never holds raw backend credentials. All tool invocations passing through the Bridge are logged with correlation IDs, enabling post-hoc audit.

P. *C. Connection to Existing Literature*

The Bridge design addresses the adoption gap quantified by [6] and the security vulnerabilities catalogued by [4]. It also aligns with the communications-systems framing of [3], positioning the Bridge as an Application-layer gateway analogous to a protocol converter in classical networking. In a practical context, a FastAPI-based Campaign Management System could be exposed to LLM agents via exactly this Bridge pattern, with existing Bearer token middleware satisfying the security preservation requirement.

VI. RESULTS AND DISCUSSION

Our literature survey reveals three convergent themes across the six papers. First, MCP's architectural separation of Tools, Resources, and Prompts is consistently identified as its most significant structural advantage over ad-hoc integration approaches-all six papers implicitly or explicitly endorse this tripartite model.

Second, security remains the protocol's most critical open challenge. Papers [1] and [4] independently converge on the finding that the specification's security model is sound in design but inconsistently implemented in practice. This gap between specification and deployment echoes historical patterns in web security, suggesting that ecosystem tooling and automated compliance checks will drive adoption more effectively than specification revisions alone.

Third, MCP's SSE transport is emerging as the preferred mechanism for streaming and IoT contexts. Paper [5]'s transport system evaluation demonstrates that SSE's low-overhead push model suits real-time sensor integration, while the stdio transport remains preferable for local developer tooling. This transport flexibility is a key differentiator from OpenAI's function-calling schema, which assumes HTTP exclusively.

The proposed MCP Bridge contributes a practical pathway to bridging the adoption gap. If even 50% of the 47 APIs evaluated by [6] could be automatically bridged, the addressable MCP tool ecosystem would expand dramatically without requiring API owners to rewrite their services.

VII. CONCLUSION

This paper has presented a comprehensive analysis of the Model Context Protocol as a foundational standard for tool-aware AI systems. Through a structured literature survey of six papers, we documented MCP's architectural mechanics, identified critical security gaps in real-world deployments, and explored emerging application domains including adaptive transport and IoT integration.

Our cross-domain analysis reveals that MCP's design philosophy-open, composable, transport-agnostic, and formally secured-positions it as the most mature integration standard currently available for LLM tool use. The protocol's adoption of established web standards (JSON-RPC 2.0, OAuth 2.1, SSE) reduces the learning curve for engineers already familiar with web systems.

The proposed MCP Bridge architecture offers a concrete research direction: automated protocol translation from REST/GraphQL to MCP. If validated empirically, the Bridge could accelerate MCP adoption by an order of magnitude, unlocking the existing API ecosystem for LLM-powered agents without requiring API owners to adopt MCP natively.

ACKNOWLEDGMENT

The authors would like to thank the Department of MCA, Bangalore Institute of Technology, for providing the academic environment and resources that supported this research. Special thanks to the NPTEL Computer Networks and Internet Protocol course (IIT Kharagpur) whose networking concepts provided foundational context for the protocol analysis presented in this work.

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155217

REFERENCES

[1] X. Hou et al., "Model Context Protocol (MCP) at a Glance: A Systematic Study of Security Issues in MCP Servers," arXiv preprint arXiv:2503.23278, 2025.
[2] A. Singh et al., "Model Context Protocol (MCP): A Comprehensive Survey for LLM-based Systems," Preprints.org, doi:10.20944/preprints202504.0670.v1, 2025.
[3] Anonymous, "Model Context Protocol: Architecture, Design, and Communications-Systems Analysis," TechRxiv, 2025.
[4] M. Hasan et al., "An Empirical Study of MCP Servers: Architecture, Security, and Adoption Patterns," arXiv preprint arXiv:2506.13538, 2025.
[5] B. Chhetri et al., "MCP-Enabled Adaptive Transport Systems," arXiv preprint arXiv:2508.19239, 2025.
[6] Anonymous, "Protocol-Agnostic Tool Integration: Bridging the MCP Adoption Gap," arXiv preprint arXiv:2508.02979, 2025.
[7] Anthropic, "Model Context Protocol Specification," https://spec.modelcontextprotocol.io, 2024.
[8] T. Schick et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," NeurIPS, 2023.
[9] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," ICLR, 2023.
[10] D. Hardt et al., "The OAuth 2.0 Authorization Framework," RFC 6749, IETF, 2012.
[11] M. Jones et al., "OAuth 2.1 Authorization Framework," IETF Internet Draft, 2024.
[12] D. Crockford, "JSON-RPC 2.0 Specification," https://www.jsonrpc.org/specification, 2010.
[13] J. Wei et al., "Emergent Abilities of Large Language Models," Transactions on Machine Learning Research, 2022.
[14] R. Nakano et al., "WebGPT: Browser-assisted Question-answering with Human Feedback," arXiv:2112.09332, 2021.
[15] R. Fielding et al., "Architectural Styles and the Design of Network-based Software Architectures," Doctoral Dissertation, UC Irvine, 2000.

