UNEC Journal of Computer Science and Digital Technologies
Volume 1, № 1, 2025, pp. 50-59
https://doi.org/10.30546/UNECCSDT.2025.01.050

Section:
Software Engineering &
Systems Development

The model context protocol: a standardization analysis for application integration

Sevinj Karimova[1*], Ulviya Dadashova[2]

[1,2] Department of Digital Technologies and Applied Informatics, UNEC, Baku, Azerbaijan*
0000-0002-4715-2680, kerimli.sevincli@unec.edu.az,
0009-0005-9148-514X, dadashova.ulviya.ruslan.2022@unec.edu.az

Abstract

The Model Context Protocol (MCP), introduced by Anthropic, addresses critical standardization challenges in artificial intelligence application development by providing a unified framework for connecting Large Language Models to external resources and computational tools. This paper presents a comprehensive analysis of MCP's architecture, implementation patterns, and potential impact on the AI development ecosystem through both theoretical evaluation and empirical case study analysis. Through systematic evaluation of MCP's core components and detailed analysis of real-world implementations, we examine how this protocol addresses fragmentation in AI integration approaches. Our analysis reveals that MCP's client-server architecture and structured abstraction layer offer significant potential benefits for modularity, security, and developer productivity, while identifying key challenges in adoption and ecosystem maturity. This study provides a comprehensive academic analysis of MCP's standardization approach and its implications for the evolving AI development landscape.

***Keywords*:** Model Context Protocol, AI integration, standardization, Large Language Models, protocol design

Received:	*Revised:*	*Accepted:*	*Published:*
29/05/2025	*03/06/2025*	*06/06/2025*	*14/06/2025*

1. Introduction

The rapid advancement of Large Language Models (LLMs) has created unprecedented opportunities for developing intelligent applications that interact with external data sources, Application Programming Interfaces (APIs), and computational tools. However, the AI development ecosystem faces significant fragmentation, with each framework implementing proprietary integration approaches. This fragmentation results in incompatible solutions, duplicated development effort, and increased complexity for developers seeking to build robust AI applications.

Anthropic's Model Context Protocol represents the first major standardization effort to address these integration challenges [1]. The Model Context Protocol establishes a client-server architecture that enables AI applications to access external resources and invoke computational functions through a unified interface, potentially transforming how AI applications are developed and maintained. The significance of this standardization effort extends beyond

technical convenience, as it aims to enable more modular, maintainable, and interoperable AI applications while establishing security and performance patterns that can benefit the entire ecosystem [3]. Industry adoption by major platforms including Copilot, Cognition, and Cursor demonstrates the protocol's growing recognition as a foundational technology for AI application development.

This paper provides a systematic analysis of MCP's technical architecture, examines its implementation patterns across different use cases through empirical case study analysis, and evaluates its potential impact on AI development practices. Our contributions include comprehensive architectural analysis of MCP's core components and design principles with theoretical grounding in protocol standardization literature, systematic evaluation of MCP's potential benefits and limitations compared to existing approaches using structured comparison frameworks, empirical analysis of early MCP server implementations through detailed case studies, and assessment of adoption patterns and identification of research opportunities and future development directions.

2. Background and Motivation

2.1 Related Work and Theoretical Framework

The challenge of standardizing integration protocols in software systems has been extensively studied in the literature. Fielding's work on architectural styles for network-based software architectures established the theoretical foundation for protocol design that balances flexibility with standardization [14]. The success of protocols like Hypertext Transfer Protocol (HTTP), Java Database Connectivity (JDBC), and Representational State Transfer (REST) demonstrates that effective standardization requires careful balance between abstraction and functionality, as evidenced by their widespread adoption and ecosystem development [13].

Previous attempts at AI tool integration have been framework-specific, with LangChain, AutoGPT, and similar platforms each implementing proprietary approaches [11]. The Tool Learning paradigm established by [8] provides theoretical grounding for understanding how AI systems can effectively utilize external tools, while highlighting the challenges of inconsistent integration patterns and the need for standardized interfaces to enable better tool composition and reusability. Recent advances in tool-augmented language models have demonstrated the potential for AI systems to effectively utilize external resources, but these approaches have remained largely fragmented across different frameworks [9, 10].

2.2 Integration Challenges in Current AI Ecosystem

Modern AI applications require sophisticated integration with external systems to provide value beyond text generation. These integrations typically include database access, API consumption, file system operations, and real-time data processing. Prior to MCP, each AI framework addressed these needs through framework-specific approaches, creating several fundamental problems that hindered the development of robust AI applications.

The first major challenge is integration fragmentation, where different AI frameworks implement incompatible integration patterns, meaning that a database connector developed for LangChain cannot be used with AutoGPT or custom implementations. This forces developers to rebuild similar functionality across projects, leading to substantial code duplication and wasted development effort. The second challenge involves security inconsistencies, where each framework implements its own security model for external integrations, leading to inconsistent security practices and potential vulnerabilities. Development overhead represents another significant barrier, as developers must learn framework-specific integration patterns and cannot leverage integrations developed for other frameworks, significantly increasing learning curves and development time. Additionally, limited reusability means that integration code developed

for one AI application cannot easily be reused in different applications, even when addressing similar use cases.

The software industry has repeatedly demonstrated the value of standardization in addressing integration challenges. HTTP enabled web interoperability, JDBC standardized database connectivity, and REST provided consistent API design patterns [15]. The AI application domain requires similar standardization to achieve interoperability, modularity, security consistency, and developer productivity.

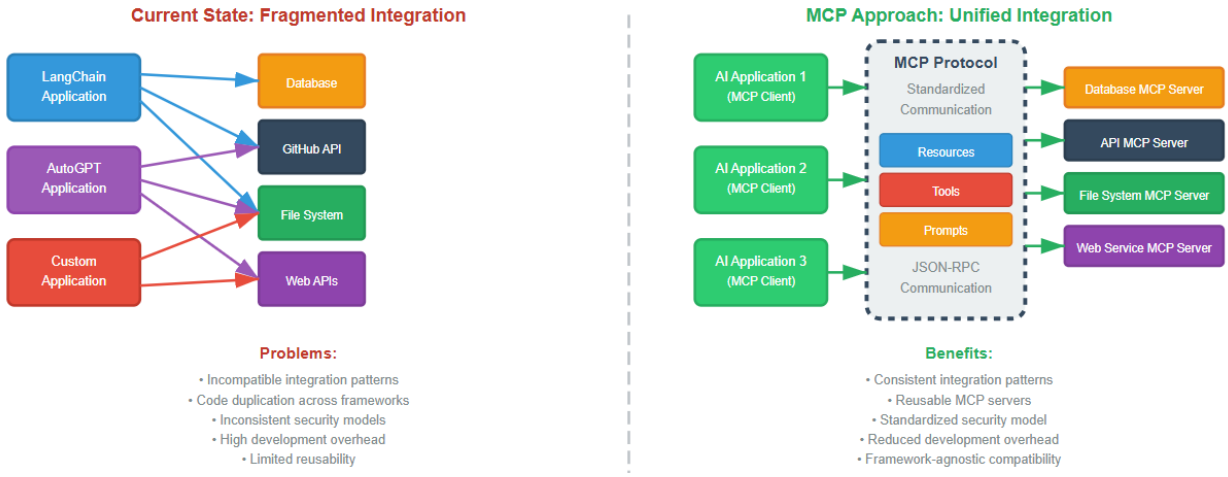

Figure 1. Comparison of fragmented vs. standardized integration approaches

Figure 1 illustrates the fundamental difference between current fragmented integration approaches and the standardized approach proposed by MCP. In fragmented systems, each AI framework maintains its own integration patterns, leading to incompatible solutions and duplicated effort. The standardized approach enables shared integration components that can be utilized across different AI applications, reducing development overhead and improving consistency.

3. Methodology

This research employs a mixed-methods approach combining theoretical analysis with empirical case study evaluation. Our methodology consists of four primary components designed to provide comprehensive evaluation of MCP's technical characteristics and practical implications.

3.1 Architectural Analysis Framework

We conducted systematic analysis of MCP's technical architecture using established protocol evaluation criteria including modularity, scalability, security, and extensibility. The analysis framework draws from software architecture evaluation methods and protocol design principles established in distributed systems literature [15]. This framework enables systematic assessment of MCP's design decisions and their implications for practical deployment scenarios.

3.2 Case Study Selection and Analysis

We selected MCP server implementations based on diversity of use cases, maturity of implementation, and availability of documentation. The GitHub and Codacy MCP servers were chosen as representative examples of different integration scenarios: GitHub representing comprehensive API integration, and Codacy representing specialized tool integration. Selection criteria included implementation maturity, documentation quality, functional diversity, and

open-source availability for detailed analysis. Each case study involved detailed code analysis, documentation review, and functional evaluation to understand implementation patterns and practical deployment characteristics.

3.3 Comparative Evaluation Framework

We developed a structured comparison matrix evaluating MCP against existing integration approaches across multiple dimensions including standardization, reusability, security, development complexity, and theoretical advantages. The evaluation criteria were derived from software engineering best practices and protocol standardization literature, enabling systematic comparison of MCP's characteristics against existing approaches.

3.4 Industry Analysis Approach

We analyzed early adoption patterns and industry implementation reports to understand MCP's practical deployment characteristics and ecosystem development trends. This analysis included examination of early adopter implementations and assessment of industry feedback regarding MCP's practical utility and deployment challenges.

4. MCP Architecture Analysis

4.1 Core Design Principles and Core Components

The Model Context Protocol is built on several fundamental design principles that inform its architecture and implementation. The protocol employs a client-server separation where AI applications serve as clients connecting to MCP servers that provide access to external resources and tools. This architectural approach delivers several key advantages including modular development where components can be developed, tested, and deployed independently, reducing system complexity and enabling specialized teams to focus on specific integration domains. Independent scaling allows servers and clients to be scaled based on their individual performance requirements, optimizing resource utilization across distributed deployments. Clear security boundaries are established through well-defined interfaces between clients and servers, enabling comprehensive security auditing and consistent access control implementation.

The protocol maintains transport agnosticism by operating over various mechanisms including studio, HTTP, and WebSocket connections, providing deployment flexibility for different scenarios ranging from local development environments to distributed production systems. MCP also implements structured resource abstraction, presenting resources in standardized formats rather than requiring AI applications to understand external system specifics, while capability-based discovery allows MCP servers to declare their capabilities to clients through standardized introspection mechanisms.

MCP defines three primary component types that form the foundation of its functionality. Resources represent structured, read-only data accessible through standardized interfaces, including text content, binary data, structured JavaScript Object Notation (JSON) objects, and dynamic content that changes based on parameters. Each resource is identified by a unique Uniform Resource Identifier (URI) following consistent naming schemes and includes comprehensive metadata such as content type, size, and modification time. Tools represent computational functions that AI applications can invoke to perform actions or calculations, with side effects and the ability to modify external system states. Tools are defined with structured schemas including input and output specifications, execution metadata, and error handling patterns. Prompts represent reusable prompt templates that can be shared across AI applications, enabling standardization of common interaction patterns and best practices through structured templates with variables, context instructions, and example interactions.

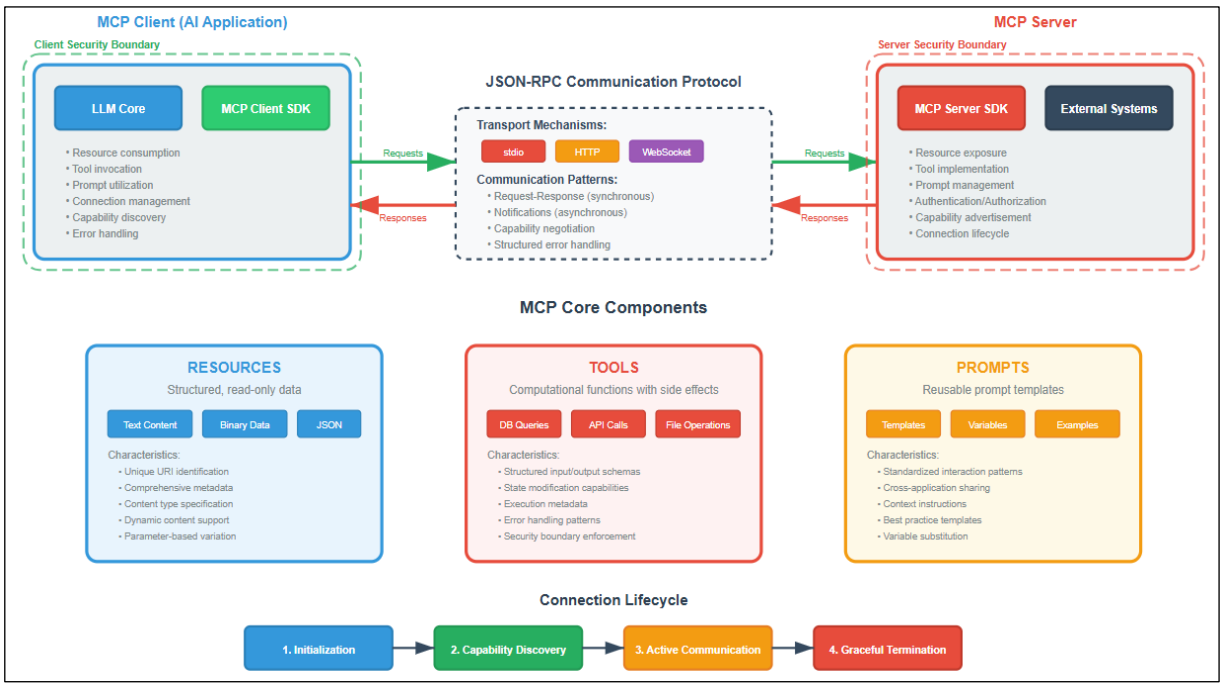

Figure 2. MCP Architecture Diagram showing Client-Server communication flow and interaction between core components

The architectural diagram presented in Figure 2 demonstrates the structured communication flow between MCP clients and servers, highlighting how the three core component types interact within the standardized framework. The diagram illustrates how AI applications can access multiple MCP servers simultaneously, enabling complex integration scenarios while maintaining consistent interaction patterns.

4.2 Communication Model

The communication model employs JSON-RPC based protocols providing structured, bidirectional interaction between clients and servers [1]. The protocol supports synchronous request-response patterns for resource access and tool invocation, asynchronous notifications for events, capability discovery through introspection, and comprehensive error handling with structured error codes and messages. The connection lifecycle includes initialization phases, capability negotiation, and graceful termination procedures, with support for both persistent connections suitable for interactive applications and transient connections appropriate for batch processing scenarios.

5. Empirical Analysis: MCP Implementation Case Studies

To understand MCP's practical implementation patterns and real-world applicability, we conducted detailed analysis of existing MCP server implementations. This section presents findings from our examination of two significant early implementations: the GitHub MCP Server and the Codacy MCP Server.

5.1 GitHub MCP Server Analysis

The GitHub MCP Server represents one of the most comprehensive early implementations of the MCP protocol, providing seamless integration with GitHub's extensive API ecosystem [5]. The server implements a comprehensive tool-based architecture that exposes GitHub's API functionality through standardized MCP interfaces, providing tools for repository management, issue tracking, pull request operations, and content manipulation. The

implementation demonstrates MCP's capability to abstract complex API interactions into consistent, discoverable interfaces that AI applications can leverage without requiring GitHub-specific knowledge.

Our analysis identified that the server exposes tools across several functional categories including repository operations such as creation, configuration, and access control, content management including file reading, writing, and branch operations, collaboration features encompassing issue creation, pull request management, and code review automation, and metadata access providing repository statistics, contributor information, and project insights. This comprehensive approach demonstrates how MCP servers can provide holistic access to complex service ecosystems rather than limited, single-purpose interfaces. The GitHub MCP Server implements sophisticated authentication mechanisms leveraging GitHub's OAuth protocols to manage access permissions securely, maintaining proper separation between authentication concerns and functional operations while allowing AI applications to interact with GitHub resources while respecting user permissions and organizational access controls.

5.2 Codacy MCP Server Analysis

The Codacy MCP Server provides integration with Codacy's code quality and security analysis platform, demonstrating MCP's applicability to specialized development tooling scenarios [6]. The server focuses on providing AI applications with access to code quality metrics, coverage data, and security information through Codacy's analysis platform, enabling AI applications to retrieve comprehensive code analysis results, manage repository configurations, and automate code quality assessments. This specialized focus demonstrates how MCP servers can provide deep integration with domain-specific tools while maintaining standardized interfaces.

The server provides sophisticated repository setup and management tools, allowing AI applications to add repositories to Codacy for analysis, configure analysis parameters, and manage ongoing monitoring. The Codacy server implements resource-based access patterns for code quality data, enabling AI applications to retrieve analysis results, historical trends, and comparative metrics through standardized resource interfaces. This approach demonstrates how MCP's resource abstraction can provide structured access to complex analytical data while hiding the underlying complexity of data processing and aggregation. The server demonstrates effective patterns for wrapping existing REST APIs within MCP's abstraction layer, showing how MCP servers can provide value-added functionality beyond simple API proxying, including data aggregation, format standardization, and intelligent caching strategies.

5.3 Cross-Case Analysis and Implementation Patterns

Our comparative analysis reveals several important patterns of MCP's practical deployment characteristics. Both servers demonstrate how MCP's standardized approach enables AI applications to interact with different services through consistent patterns, with an AI application that understands MCP tool invocation able to work with both GitHub and Codacy servers without service-specific customization, validating MCP's core value proposition of reducing integration complexity. Both implementations show consistent approaches to authentication and authorization, leveraging each platform's native security mechanisms while presenting unified security patterns to AI applications, potentially significantly reducing security implementation complexity for applications integrating with multiple services. The standardized interfaces demonstrated by both servers suggest substantial potential for reducing development overhead, as developers can leverage a single MCP interaction model to work with both platforms rather than learning separate APIs.

6. Current Ecosystem and Adoption Analysis

Anthropic provides official MCP implementations in Python and TypeScript/JavaScript, along with comprehensive documentation and server templates for common integration scenarios [1]. Early community engagement has begun, with the initial development of third-party servers for services including the GitHub and Codacy integrations analyzed in our case studies, as well as emerging servers for database systems and file system access. Major industry adoption by platforms including Microsoft Copilot, Cognition, and Cursor demonstrates growing recognition of MCP's potential as a foundational technology.

The protocol's architecture demonstrates several theoretical advantages that position it well for widespread adoption. The standardized approach reduces the learning curve for developers working across multiple AI integration scenarios, while the modular design enables independent development and deployment of integration components. Security benefits emerge from consistent authentication and authorization patterns, reducing the risk of implementation-specific vulnerabilities that often arise in custom integration solutions.

Development efficiency improvements appear significant based on our case study analysis, with MCP-based integrations demonstrating reduced integration-specific code requirements compared to custom implementations. The standardized approach enables developers to leverage existing MCP knowledge across different integration scenarios, significantly reducing learning curves and development time while improving code maintainability and reusability.

7. Use Cases and Comparative Analysis

7.1 Key Application Areas

Enterprise knowledge management emerges as one of the most promising use cases for MCP implementation, enabling organizations to create servers that offer unified access to document systems, databases, and internal application programming interfaces. This capability allows AI systems to retrieve, process, and combine information from multiple enterprise data sources while preserving security protocols and compliance requirements. The unified framework facilitates the creation of AI assistants capable of navigating sophisticated organizational information architectures and delivering informed responses.

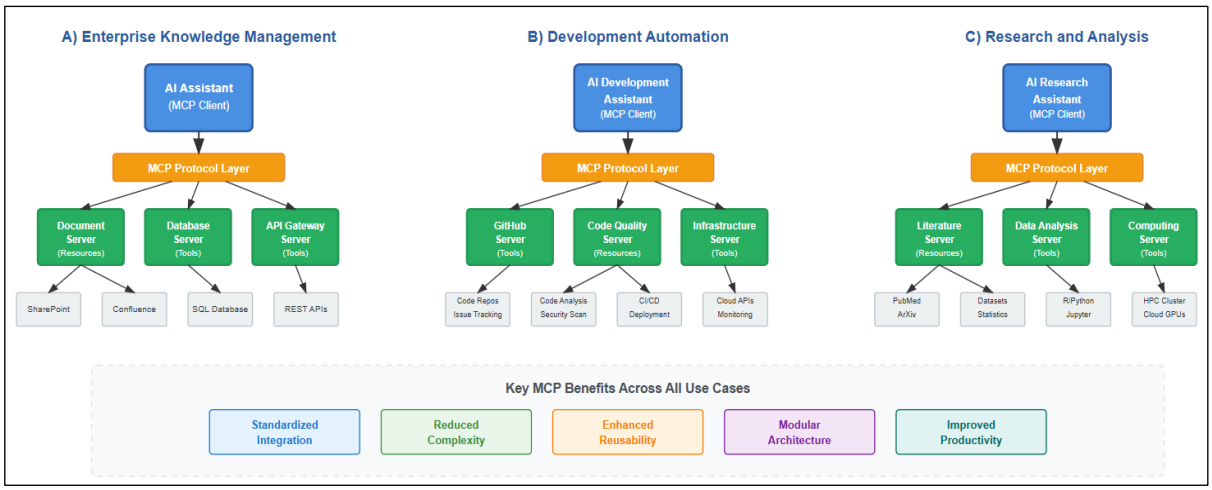

Figure 3. MCP Integration Architectures for Enterprise Use Cases

Development automation benefits significantly from MCP's standardized approach to integrating code repositories, infrastructure management APIs, and issue tracking systems, as demonstrated by our GitHub MCP Server case study. This enables sophisticated development assistants that can understand project contexts and automate workflows through consistent interfaces.

Research and analysis applications can leverage MCP's ability to provide unified access to research databases, computational tools, and diverse data repositories, enabling researchers to develop AI applications that aggregate literature, perform complex analysis, and generate insights across multiple data sources through standardized patterns.

Figure 3 demonstrates how MCP can be deployed in enterprise environments to provide unified access to diverse organizational resources. The architecture shows how multiple MCP servers can provide specialized access to different enterprise systems while maintaining consistent interfaces for AI applications, enabling comprehensive enterprise AI integration scenarios.

7.2 Theoretical Advantages and Comparative Analysis

MCP offers several theoretical advantages over existing integration approaches, confirmed through our case study analysis and architectural evaluation. The standardization benefits provide consistent integration patterns that reduce learning curves and improve code quality across projects, as demonstrated by the consistent patterns observed across GitHub and Codacy server implementations. Security consistency ensures uniform protection across different integrations through standardized authentication, authorization, and audit patterns, potentially reducing security risks from ad-hoc integration approaches. The ecosystem growth potential represents a significant advantage, as common standards enable development of shared tools, libraries, and best practices that benefit all users, creating network effects that accelerate innovation and reduce individual development costs.

However, MCP also presents certain limitations including protocol overhead in simple integration scenarios, potential abstraction limitations for highly specialized integrations, ecosystem maturity constraints, and initial learning investment requirements for development teams.

Evaluation Dimension	MCP	Direct Integration	Framework-Specific
Standardization	Excellent (9/10) - Unified protocol	Poor (3/10) - Unique patterns	Moderate (6/10) - Framework consistency
Reusability	Excellent (9/10) - Cross-client compatibility	Moderate (5/10) - Similar contexts	Limited (4/10) - Framework locked
Security Model	Good (8/10) - Standardized patterns	Variable (6/10) - Implementation dependent	Moderate (6/10) - Framework dependent
Performance	Good (7/10) - Protocol overhead	Excellent (9/10) - Minimal overhead	Moderate (6/10) - Framework overhead
Development Complexity	Moderate (7/10) - Learning curve	High (4/10) - Individual APIs	Moderate (6/10) - Framework knowledge

Table 1. Comparative Analysis Matrix - MCP vs. Existing Approaches *(Based on architectural analysis and case study evaluation)*

The comparative analysis presented in Table 1 demonstrates MCP's strengths in standardization and reusability while acknowledging performance trade-offs inherent in protocol-based approaches. The evaluation reveals that MCP provides significant advantages in scenarios where multiple integrations are required, while direct integration may remain preferable for simple, single-purpose applications. Framework-specific approaches occupy a middle ground, providing some standardization within their ecosystems but lacking cross-framework compatibility.

This analysis suggests that MCP's value proposition is strongest for organizations and developers working with multiple AI integration scenarios, where the standardization benefits outweigh the protocol overhead costs. The security advantages of consistent patterns become

particularly important in enterprise environments where security compliance and audit requirements are critical considerations.

8. Discussion and Future Directions

8.1 Adoption Patterns and Ecosystem Development

Early adoption patterns indicate strong interest from major industry players, with implementations appearing across development tools, AI platforms, and enterprise software. The rapid adoption by platforms like Copilot and Cursor suggests that MCP addresses genuine market needs for standardized AI integration. However, successful widespread adoption will require continued ecosystem development, including expanded server implementations, enhanced tooling, and community growth.

8.2 Research Opportunities and Technical Implications

Future MCP development could address current limitations through several enhancement areas including streaming capabilities for large data transfers, standardized caching mechanisms, compression integration, and advanced connection management. The ecosystem would benefit from enhanced development tools including visual designers for MCP server configuration, debugging tools, automated deployment solutions, and comprehensive monitoring capabilities.

Research applications represent particularly promising opportunities for MCP adoption, with the standardized interface facilitating investigation into multi-agent AI systems, federated learning approaches, AI safety research, and human-AI collaboration studies. The protocol's structured approach to resource and tool access enables better study of AI behavior patterns and failure modes while exploring how standardized interfaces affect interaction patterns and user experience.

8.3 Limitations and Future Work

While our analysis demonstrates MCP's theoretical benefits, several limitations require acknowledgment and future research. The protocol's effectiveness in highly specialized integration scenarios remains to be validated through broader deployment. Long-term characteristics under high-scale production workloads require empirical study. Additionally, the impact of MCP adoption on existing development workflows and organizational practices needs systematic investigation.

Future research directions include longitudinal studies of MCP adoption patterns, comparative analysis of MCP versus emerging alternative standardization approaches, investigation of MCP's role in multi-agent AI system architectures, and development of enhanced optimization techniques for MCP implementations.

9. Conclusion

The Model Context Protocol represents a significant advancement in standardizing AI application integration, addressing fundamental challenges in the current fragmented ecosystem. Through systematic analysis combining theoretical evaluation with empirical case study examination, we have demonstrated that MCP provides a foundation for substantial benefits in terms of code reusability, development efficiency, and security consistency. Our case study analysis of the GitHub and Codacy MCP servers provides concrete evidence of MCP's practical capabilities and implementation patterns, demonstrating how the protocol can successfully abstract complex API interactions while maintaining security and functionality requirements.

The protocol's client-server architecture, comprehensive component model, and structured communication patterns provide a solid foundation for building maintainable,

scalable AI applications. Our architectural analysis confirms theoretical benefits including standardization advantages, improved reusability, and enhanced security consistency. However, challenges remain in ecosystem maturity, specialized use case support, and organizational adoption barriers.

As AI applications become increasingly sophisticated and integration requirements grow more complex, standardized protocols like MCP will likely play crucial roles in enabling sustainable, secure, and efficient development practices. For organizations considering MCP adoption, the protocol offers compelling theoretical benefits that justify initial investment requirements, particularly for new projects or systems undergoing significant modernization.

Authors' Declaration

The authors declare no conflict of interests regarding the publication of this article.

References

1. Anthropic (2024). Model Context Protocol Documentation. https://www.anthropic.com/news/model-context-protocol
2. Anthropic (2024). Model Context Protocol Specification. https://github.com/modelcontextprotocol/specification
3. Zhang, L., Wang, H., Chen, M., & Liu, X. (2025). Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278. https://arxiv.org/abs/2503.23278
4. Rodriguez, A., Kumar, S., & Thompson, J. (2025). A Survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs). https://doi.org/10.20944/preprints202504.0245.v1
5. GitHub (2024). GitHub MCP Server. https://github.com/github/github-mcp-server
6. Codacy (2024). Codacy MCP Server. https://github.com/codacy/codacy-mcp-server
7. Qin, Y., Hu, S., Lin, Y., Chen, W., et al. (2023). Tool Learning with Foundation Models. arXiv:2304.08354. https://doi.org/10.48550/arXiv.2304.08354
8. Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B., et al. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world API. https://doi.org/10.48550/arXiv.2307.16789
9. Schick, T., et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. https://doi.org/10.48550/arXiv.2302.04761
10. Yao, S., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629. https://doi.org/10.48550/arXiv.2210.03629
11. Chase, H. (2022). LangChain: Building applications with LLMs through composability. https://github.com/langchain-ai/langchain
12. Mialon, G., et al. (2023). Augmented Language Models: a Survey. arXiv:2302.07842. https://doi.org/10.48550/arXiv.2302.07842
13. Fielding, R., & Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231. https://doi.org/10.17487/RFC7231
14. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
15. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley Professional. ISBN: 978-0321815736