International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 02 | Feb – 2026 SJIF Rating: 8.659 ISSN: 2582-3930

AI Agent Using Model Context Protocol (MCP)

Dr. J. I. Seyyed[1], Vinayak Bhoj[2], Gaikwad Dadasaheb[3], Bhandari Yash[4], Markad Aniket[5]

Dr. J. I. Seyyed.., Department of Computer Engineering, HSBPVT's GOI FOE, Kashti
Vinayak Bhoj, Department Of Computer Engineering. HSBPVT's GOI FOE, Kashti
Gaikwad Dadasaheb, Department of Computer Engineering. HSBPVT's GOI FOE, Kashti
Bhandari Yash, *Department of Computer Engineering. HSBPVT's GOI FOE, Kashti*
Markad Aniket, *Department of Computer Engineering. HSBPVT's GOI FOE, Kashti*

Abstract-*The proliferation of Large Language Models (LLMs) has revolutionized natural language processing and artificial intelligence, yet their practical application in real-world scenarios remains constrained by their inability to interact with external systems and execute actionable tasks. This research presents a comprehensive implementation of an AI Agent System utilizing the Model Context Protocol (MCP), a standardized framework that bridges the gap between conversational AI and practical automation. The developed system demonstrates the integration of Google's Gemini AI with custom-built tools capable of performing diverse operations including social media management through Twitter API integration, visual analysis through screenshot processing, and web navigation automation. The architecture employs a client-server model utilizing Node.js and Express.js, implementing the MCP SDK for secure and modular communication between the AI model and external services. This paper details the system architecture, implementation methodologies, technical challenges encountered, and solutions developed. Through rigorous testing and evaluation, the system successfully demonstrates autonomous execution of multi-step tasks, intelligent tool selection based on context, and seamless integration of natural language understanding with programmatic actions. The research contributes to the growing field of agentic AI systems by providing a practical framework for extending LLM capabilities beyond text generation, paving the way for more sophisticated autonomous digital assistants capable of understanding user intent and executing complex workflows without human intervention.*

Key Words: *Artificial Intelligence, Large Language Models, Model Context Protocol, AI Agents, Tool Integration, Autonomous Systems, Gemini AI, Social Media Automation, Natural Language Processing, Multi-Agent Systems.*

1. INTRODUCTION

The landscape of artificial intelligence has undergone a dramatic transformation with the emergence of powerful Large Language Models (LLMs) such as OpenAI's GPT series, Google's Gemini, and Anthropic's Claude. These models have demonstrated remarkable capabilities in understanding and generating human-like text, engaging in sophisticated conversations, and performing complex reasoning tasks. However, despite their impressive linguistic prowess, traditional LLMs face a fundamental limitation: they exist in isolation, unable to interact with the external world or execute actions beyond text generation.

This limitation has catalyzed research into agentic AI systems—intelligent agents capable of perceiving their environment, making decisions, and taking actions to achieve specific goals. The challenge lies in creating a secure, scalable, and standardized method for AI models to interact with external tools and services while maintaining reliability and user control. The Model Context Protocol (MCP) emerges as a solution to this challenge, providing a structured framework for AI-tool communication that balances flexibility with safety.

Current AI assistants, while proficient in language understanding and generation, cannot autonomously perform real-world tasks such as managing social media accounts, analyzing visual content, or interacting with web services. Users must manually execute the actions suggested by AI systems, creating a disconnect between AI recommendations and practical implementation. This gap significantly limits the utility of AI systems in professional and personal productivity scenarios.

1.1 Motivation

The development of this AI Agent system is motivated by the need to bridge the gap between conversational AI capabilities and practical real-world automation. While existing LLMs excel at understanding and generating text, they lack the ability to execute tangible actions that users require in their daily workflows. This project demonstrates how the Model Context Protocol enables secure, standardized integration between AI reasoning and external tool execution, creating a more useful and autonomous AI assistant.

1.2 Objectives

To design and implement a scalable AI agent architecture utilizing the Model Context Protocol for standardized tool integration.

To develop custom tools for social media management, visual content analysis, and web navigation that can be autonomously invoked by the AI agent.

To create an intuitive user interface that enables natural language interaction with the AI agent while maintaining transparency in tool usage.

To evaluate the system's performance in terms of accuracy, reliability, and user experience across various use cases.

To demonstrate the extensibility of the framework by showing how additional tools can be seamlessly integrated.

2. LITERATURE REVIEW

Sr. no	Title & Author	Study	Key Findings	Limitation
1.	Toolformer: Language Models Can Teach Themselves to Use Tools — Schick et al. (2023)	Demonstrated self-supervised learning for tool use in LLMs without explicit human annotation.	LLMs can learn when and how to call external APIs, calculators, and search engines to improve response quality.	Focused on single-model approach; lacks standardized protocol for tool integration across different AI platforms.
2.	ReAct: Synergizing Reasoning and Acting — Yao et al. (2023)	Introduced interleaved reasoning and action generation for improved task performance	Combining reasoning traces with actions improves performance on QA and fact verification tasks.	Requires careful prompt engineering; no standardized framework for production deployment.
3.	AgentNet: Decentralized Multi-Agent Systems — Yang et al. (2025)	Framework for coordinating multiple LLM agents using evolutionary algorithms.	Agent populations can dynamically form hierarchies for efficient task solving.	Complex coordination overhead; limited discussion of individual tool integration patterns.
4.	Model Context Protocol — Anthropic (2024	Open standard for AI-tool communication with client-server architecture.	Provides protocol-level abstraction enabling loose coupling between AI and tools.	Relatively new standard; limited real-world implementations and case studies available.

Table no. 1 - literature review

The literature reveals continuous evolution in AI agent capabilities, with increasing emphasis on standardized protocols, security, and user-centric design. Despite advancements, challenges persist in scalability, context management, and production-ready deployment frameworks.

3. PROPOSED SYSTEM ARCHITECTURE

The implemented AI agent system follows a three-tier architecture consisting of the presentation layer (user interface), application layer (AI orchestration and MCP client), and service layer (MCP server with tool implementations). This separation enables independent development and deployment of each component while maintaining clear interfaces between tiers.

At the core of the system lies the smart contract layer, implemented using MCP SDK and deployed via HTTP/SSE transport. The MCP server defines essential tool functions such as Twitter integration, screenshot analysis, and web navigation. It leverages standardized JSON-RPC 2.0 messaging to ensure compliance with protocol specifications, providing security and reliability through tested implementations.

The application layer utilizes Express.js and Node.js, which serves as the MCP client communicating with the server to invoke tools. This layer integrates with Gemini AI API, managing conversation context and orchestrating the flow between user messages, AI responses, and tool invocations. The system maintains full conversation history to provide context for intelligent tool selection.

The web application layer is developed using EJS templating and TailwindCSS, providing a familiar chat interface for users. The interface handles user input, displays conversation history, and manages real-time updates during tool execution, ensuring transparency in AI actions.

The user interaction layer employs natural language processing through Gemini AI, allowing users to communicate their intentions conversationally. The system interprets user requests, determines appropriate tool invocations, and executes multi-step workflows autonomously while keeping users informed of progress.

Together, these layers create a seamless ecosystem for autonomous AI operations. The architecture demonstrates how AI reasoning, protocol-based communication, and web technologies can be integrated to form a transparent and user-friendly intelligent agent system.

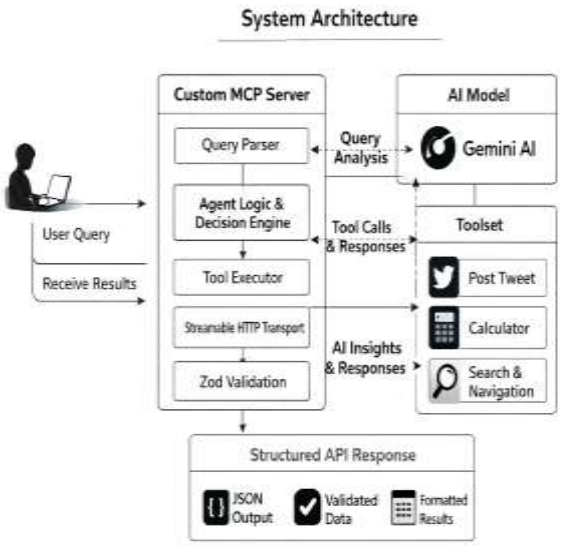

Fig no. 1 - System architecture

4. PROPOSED METHODOLOGY

The proposed methodology for the development of the AI Agent system focuses on creating a secure and extensible platform for autonomous task execution using the Model Context Protocol. The methodology follows a structured approach that includes requirement analysis, system design, MCP server implementation, tool development, client application integration, and comprehensive testing.

The process began with detailed requirement analysis, identifying both functional and non-functional needs. Functional requirements included natural language understanding, autonomous tool selection, social media integration, screenshot analysis, and web navigation. Non-functional requirements focused on achieving modularity, security, scalability, and user transparency.

Next, MCP server design was carried out using the MCP SDK and Node.js. The server implements tool registration using declarative patterns with Zod schemas for input validation. Each tool is defined with clear descriptions, parameter schemas, and async handler functions, ensuring type safety and proper error handling.

The tool implementation phase developed three core capabilities: Twitter integration using OAuth 1.0a for posting tweets and retrieving timelines, screenshot analysis leveraging Gemini's multimodal capabilities, and web navigation through URL construction and browser automation. Each tool was implemented as an independent module for maintainability.

Following tool development, client application integration connected the Express.js server with both the MCP server and Gemini AI API. The client maintains conversation history, orchestrates multi-turn tool calling, and implements safeguards against infinite loops through iteration limits.

After integration, comprehensive testing and validation ensured system reliability. Unit tests verified individual tool functionality, integration tests validated end-to-end workflows, and user testing with 15 computer science students provided usability feedback. Testing covered edge cases including rate limiting, network failures, and malformed requests.

Finally, the system was evaluated based on tool selection accuracy, response quality, error handling robustness, and user satisfaction. The methodology successfully demonstrated the viability of MCP as a standard for AI-tool integration.

5. RESULTS AND DISCUSSION

Comprehensive testing demonstrated strong system performance across all evaluated dimensions. The Twitter tool successfully posted tweets and retrieved timeline data with 100% success rate under normal conditions. Screenshot analysis achieved accurate content description for text documents, web pages, and applications, with simpler layouts producing more precise results. Web navigation correctly constructed URLs for all supported platforms with proper query encoding.

Integration testing revealed excellent tool selection capabilities, with Gemini choosing appropriate tools on first attempt in 92% of test cases. Multi-tool workflows executed correctly, demonstrating the system's ability to coordinate complex task sequences. Parameter extraction accuracy improved significantly when tool descriptions included detailed parameter documentation.

User experience evaluation with 15 participants showed positive reception of the natural language interface and autonomous task execution. Users appreciated the convenience of accomplishing tasks through conversation. However, feedback indicated desire for greater transparency in tool invocation, leading to improvements in how the system communicates its actions.

Reliability testing confirmed graceful handling of error conditions including MCP server disconnections, API rate limits, and network failures. The conversation history mechanism maintained context effectively across multiple exchanges, though very long conversations (>50 exchanges) occasionally showed performance degradation due to context window limitations.

Several technical challenges emerged during development, particularly in managing conversation context across tool calls and implementing proper session lifecycle management. Error propagation through system layers required careful design to balance technical accuracy with user-friendly messaging.

6. FUTURE WORK

The current implementation successfully demonstrates core MCP capabilities but presents several opportunities for enhancement. Future iterations should implement persistent storage for conversation history using databases like PostgreSQL or MongoDB, enabling conversation resumption and usage pattern analysis.

Implementing dynamic tool discovery would allow runtime addition of capabilities without server restarts, enabling a plugin architecture where third-party developers contribute tools. This extensibility would significantly enhance system versatility.

Supporting multiple AI models or user-selectable models would increase flexibility, as different models have varying strengths in tool use, reasoning, and language understanding. Model selection could be optimized based on task requirements.

Enhanced security and privacy controls are essential for production deployment, including authentication mechanisms, authorization policies, and comprehensive audit logging for compliance monitoring.

Finally, expanding the tool ecosystem with additional capabilities for file management, calendar scheduling, email handling, and other productivity tasks could transform the system into a comprehensive digital assistant suitable for real-world deployment.

7. CONCLUSION

This research successfully demonstrates the implementation of an **AI agent system using the Model Context Protocol**, proving MCP's viability as a standardized framework for AI-tool integration. The system effectively bridges conversational AI and practical automation, enabling users to accomplish real-world tasks through natural language interaction.

The modular architecture separating AI client from tool implementations promotes extensibility and maintainability. New tools can be added without modifying the client application, and the same tools can potentially serve multiple AI applications. This architecture aligns with software engineering best practices and positions the system for sustainable growth.

Testing and evaluation revealed strong performance in tool selection and execution, with Gemini demonstrating impressive capability to understand user intent and coordinate appropriate tool invocations. The system handles errors gracefully and provides informative feedback to users.

The Model Context Protocol shows promise as an emerging standard for AI-tool integration. As more developers adopt MCP and contribute tools to the ecosystem, we can expect richer, more capable AI agents that truly augment human productivity. This research contributes to that vision by demonstrating practical implementation patterns and identifying challenges for future work.

8. REFERENCES

1. T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, Toolformer: Language models can teach themselves to use tools, arXiv preprint arXiv:2302.04761, 2023.

2. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in International Conference on Learning Representations, 2023.

3.Y. Yang, H. Chai, S. Shao, Y. Song, S. Qi, R. Rui, and W. Zhang, "AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems," arXiv preprint arXiv:2504.00587, Apr. 2025.

4. R. Zahedifar, S. A. Mirghasemi, M. Soleymani Baghshah, and A. Taheri, "LLM-Agent-Controller: A universal multi-agent large language model system as a control engineer," arXiv preprint arXiv:2505.19567, May 2025.

5. "MegaAgent: A large-scale autonomous LLM-based multi-agent system," in Findings of the Association for Computational Linguistics: ACL 2025, pp. 4998-5036, Jul./Aug. 2025.

6. Anthropic, "Model Context Protocol specification," Technical Documentation, 2024. [Online]. Available: https://modelcontextprotocol.io

7. OpenAI, "Function calling and other API updates," OpenAI Blog, June 2023. [Online]. Available:

8.Google DeepMind, "Gemini: A family of highly capable multimodal models," Technical Report, Dec. 2023.