WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Comprehensive Review of MCP-Enabled AI Agent Systems

Revuri Ragamayee

Department of Computer Engineering, Stanley College Of Engineering & Technology For Women, Telangana, India

A B S T R A C T :

Large Language Models (LLMs) are used as a tool for content creation, automation, and language translation. These models are increasingly evolving over time, with various techniques developed to improve the accuracy and reliability of their outputs. The Model Context Protocol (MCP) has emerged as a standard that enables AI models to connect to external systems, thereby enhancing contextual understanding and response generation. Such external systems may include data sources such as files and databases, tools like search engines and calculators, and workflows such as specialized prompts [3]. This integration allows users to deploy personalized models trained on their own data, resulting in more reliable and dynamic AI assistants [4].

Keywords: Artificial Intelligence (AI), Model Context Protocol (MCP), AI Agents, AI Frameworks, Large Language Models (LLMs), Humanity's Last Exam (HLE)

1. Introduction

Recent LLMs offer advanced features that enable the generation of detailed reports, including citations to existing sources. Humanity's Last Exam (HLE) is a performance metric that assesses the quality of a model response **[1]**. According to **[2]**, leading models such as 'gemini-3-pro-preview' (HLE score: 37.52), 'gpt-5-pro-2025-10-06' (31.64), and 'claude-opus-4-5-20251101-thinking' (25.2) have been evaluated, with Gemini 3 Pro achieving the highest score as of 2025.

AI agents are task-specific roles assigned to an LLM, which work together to form a network and distribute the workload balance as well as limit the scope of each agent, so it can give an efficient output **[5]**. This can be achieved by using popular AI Agent frameworks such as LangChain, CrewAI, Microsoft AutoGen and LangGraph **[6]**. Normally, AI agents rely on custom APIs, plugins, or framework-specific methods to perform actions, which makes development slow and often incompatible across platforms. These agents can work in an MCP environment, which takes it a step further by providing the framework with various tools and data from different real-time sources. MCP solves this problem by providing a standardized way for agents to discover and use external tools, resources, and services. When AI agents work through MCP, they can retrieve data, execute tasks, and communicate with different systems more reliably and securely. For example, in a healthcare setting, an MCP-enabled AI agent can access real-time patient records from secure databases, interact with medical imaging tools, and provide clinicians with efficient, up-to-date diagnostic suggestions while maintaining compliance with privacy standards.

Beyond addressing interoperability challenges, recent advancements in MCP research highlight how the protocol strengthens tool reliability, enhances contextual awareness, and improves the long-term maintainability of agent ecosystems [7], [8]. These developments are crucial as modern AI systems increasingly rely on multi-agent collaboration, autonomous workflows, and standardized access to external resources. The integration of MCP with agent frameworks opens new pathways for building intelligent systems that operate with greater consistency, reduced fragmentation, and improved real-world applicability. This creates the groundwork for analyzing how agent architectures, tool schemas, and orchestrated workflows can benefit from a unified protocol that aligns with emerging trends in AI automation and scalable context management.

2. Background

Foundational Models (FMs) are widely adopted because of their powerful capabilities of language understanding, generation, and reasoning. Many applications across various domains for e.g., healthcare, e-commerce, and finance, use these FMs. However, the traditional FMs are operated through isolated "textual" interfaces which limit its ability to interact dynamically **[8]**. This problem is also known as 'Data Silo' in AI, which means providing training data which is isolated and inconsistent. Such fragmentation can affect the AI model's comprehensive understanding and logical reasoning.

Model Context Protocol addresses this problem by providing the AI system with diverse data sources and tools **[9]**. MCP is proposed as a unifying standard for connecting LLMs with external tools and resources. It discusses the large-scale empirical study of the MCP ecosystem and validates the protocol's centrality and rapid adoption. The key participants in the MCP architecture are:

- **MCP Host**: The AI application that coordinates and manages one or multiple MCP clients.
- **MCP Client**: maintains a connection to an MCP server and obtains context from the server for the MCP host to use.
- **MCP Server**: A program that provides context to the MCP clients **[3]**.

MCP Hosts are software applications that interact with large language models, like Claude Desktop or Cursor IDE. They are responsible for establishing connections and providing an environment where users can run tasks. Inside each MCP Host, MCP Clients maintain continuous connections with MCP Servers and handling tasks like discovering available tools, formatting requests, and delivering results. MCP Servers, on the

other hand, provide the core functionality, offering structured APIs for tools (which are functions with specific inputs and outputs), resources, and prompts. **[11]**

MCP is sizable; however, this ecosystem is structurally fragile **[7]**. Based on the analysis in **[9]**, models which have top-performing scores still lack effectiveness in real-world MCP environments. MCP has its own challenges, which is not the scope of this paper.

3. Methodology

3.1 MCP-Prompt based Context Integration

This methodology examines how MCP's prompt capabilities provide AI agents with structured and reusable task templates. MCP defines "prompts" as standardized, server-exposed workflows that supply contextual instructions, metadata, and constraints to guide LLM behavior [3]. These prompts help reduce ambiguity and improve task consistency, especially in environments where AI agents rely on stable contextual grounding. Studies on the MCP ecosystem indicate that structured prompts support more predictable tool usage and reduce execution variability across different model providers [7]. Additionally, existing analyses on MCP maintainability show that prompt definitions play a key role in enhancing reliability by enforcing schema-driven interactions between LLMs and external systems [8]. This method provides insight into how MCP prompts act as cognitive scaffolding agents, improving reasoning accuracy and reducing hallucination, particularly in tool-intensive workflows.

3.2 MCP with LangChain for Tool Invocation

This methodology explores integrating MCP-exposed tools directly into LangChain pipelines. LangChain typically connects to tools using custom wrappers, which can lead to fragmentation and inconsistencies across agent implementations. MCP addresses this challenge by offering a unified schema for tool description and invocation, allowing agents to discover available capabilities dynamically rather than relying on hard-coded integrations [3]. Recent MCP benchmarking studies demonstrate that standardized tool interfaces significantly improve LLM performance when interacting with real-world servers, reducing tool-call errors, and enabling better generalization across model families [9]. Research on MCP's security and maintainability further confirms that consistent tool schemas reduce misuse, incorrect parameter formatting, and failure modes that are common in non-standardized environments [8]. By embedding MCP tools directly into LangChain chains, this methodology evaluates improvements in tool coordination, reasoning depth, and multi-step task execution reliability.

3.3 MCP-Enabled AI Orchestration for Multi-Agent Systems

The final methodology investigates the role of MCP in orchestrated multi-agent environments. Modern agent frameworks such as AutoAgent, CrewAI, and LangGraph depend heavily on tool access, shared memory, and inter-agent communication to perform distributed tasks [6]. MCP provides a standardized layer through which all agents (regardless of their specialization) can access consistent tools, data sources, and process workflows. Measurement studies of MCP adoption confirm that a unified protocol reduces duplicated engineering effort and enables multiple agents to coordinate shared tool definitions [7]. Threat analysis and landscape evaluations emphasize that MCP's structured interfaces not only support interoperability but also introduce clearer security boundaries for multi-agent execution [10]. Through this methodology, the study evaluates how MCP enhances coordination efficiency, scalability, and context sharing among agents, enabling more reliable orchestration patterns such as planner–executor–validator loops.

3.4 Integration of MCP in Cursor for AI-Assisted Development

This methodology examines how MCP is utilized within *Cursor*, an AI-powered coding environment designed to enhance software development workflows. Cursor integrates MCP servers to provide LLMs with structured access to project files, system resources, development tools, and code-generation utilities through standardized interfaces [3]. By exposing the local filesystem, codebase context, and developer tools through MCP schemas, Cursor enables AI agents to perform tasks such as code retrieval, file modification, dependency inspection, and project refactoring with greater precision. Studies measuring MCP usage patterns show that environments like Cursor benefit significantly from MCP's standardized tool invocation, reducing inconsistencies that typically arise in editor-specific extensions [7]. Security and maintainability research highlights that MCP's schema-driven approach helps minimize unsafe file operations and enforces clear permission boundaries when agents modify code or execute development tasks [8]. This methodology therefore evaluates the role of MCP in improving code intelligence, development automation, and workflow reliability within agent-assisted IDEs, demonstrating how standardized protocols support safe and scalable integration of AI agents into real-world software engineering environments.

4. Discussion

MCP provides a standardized interface for context exchange and tool invocation, reducing the fragmentation that currently affects agent-tool ecosystems. As highlighted in recent studies on MCP adoption, a unified protocol eliminates duplicated engineering efforts and introduces consistent integration patterns that earlier agent systems lacked **[10]**. AI agents that are specially designed to operate autonomously in dynamic environments

benefit from MCP, which is a standardized alternative to traditional APIs. Another important aspect revealed in recent literature is how MCP enhances *interoperability* among heterogeneous AI agent frameworks. Systems such as LangChain, CrewAI, AutoGen, and LangGraph each implement their own mechanism for tool invocation and context sharing, which often results in incompatibility and re-implementation overheads across platforms. MCP acts as a neutral protocol layer, enabling these frameworks to uniformly access tools and resources regardless of the model provider or programming environment.

MCP introduces new opportunities for multi-agent collaboration. By offering a unified protocol through which all agents can access shared tools, memory resources, and data workflows, MCP can strengthen multi-agent coordination models described. This protocol supports emerging architectures where specialized agents (such as planners, retrievers, validators, or analyzers) can interact more efficiently using a common protocol layer.

In addition to improving interoperability and multi-agent coordination, the methodological approaches used in this study highlight how MCP strengthens the practical usability of AI agents across different environments. The analysis of MCP prompts demonstrates that providing structured, reusable prompt definitions improves the consistency and reliability of LLM outputs, especially in tasks that require step-by-step reasoning or strict adherence to workflows. This contributes to reducing hallucinations and ensuring that tool interactions remain predictable, which continues to be a major challenge in agent-tool ecosystems.

The methodology involving MCP integration with LangChain further shows that standardized tool schemas significantly reduce the engineering complexity associated with adding new capabilities to agent pipelines. Instead of writing custom wrappers or tool-specific adapters for every model or framework, developers can expose tools once through an MCP server. LangChain agents can then discover and invoke these tools dynamically, improving cross-platform compatibility and enabling rapid experimentation with different agent strategies. This is especially important as modern AI workflows grow more modular, and developers seek flexible, plug-and-play architectures.

Similarly, the examination of MCP-enabled AI orchestration illustrates how standardized tool access enhances scalability in multi-agent systems. Coordinating specialized agents (such as planners, retrievers, and validators) becomes more efficient when they can rely on a shared protocol for accessing tools and context. Instead of exchanging raw text or custom API calls, agents can communicate through structured, schema-defined interactions, reducing ambiguity, and enabling more reliable collaboration. This supports more advanced orchestration patterns commonly used in research and enterprise applications. Finally, the investigation of MCP's role within development environments like Cursor shows how MCP can improve real-world productivity. By exposing files, project metadata, and development utilities through a controlled MCP interface, AI models can assist with code editing, refactoring, and project navigation safely and accurately. This demonstrates how MCP not only supports large-scale agent systems but also enhances everyday developer workflows by providing AI with a secure, structured understanding of local environments.

5. Conclusion

The integration of the Model Context Protocol (MCP) with AI agent systems represents a significant step toward standardizing how intelligent models interact with external tools, data sources, and computational workflows. As demonstrated across the methodological evaluations, MCP reduces fragmentation in agent architectures by offering a unified interface for tool invocation, context exchange, and capability discovery. The analysis of MCP prompts, LangChain integration, multi-agent orchestration, and development environments such as Cursor collectively highlights how structured schemas and consistent protocols can improve reliability, maintainability, and scalability in real-world AI deployments.

MCP's standardization efforts address long-standing challenges in agent ecosystems, where custom APIs and framework-specific integrations often introduce inconsistencies and redundant engineering overhead. By enabling heterogeneous agents to access shared resources through a common protocol, MCP strengthens collaboration between specialized components such as planners, retrievers, and validators, leading to more stable and predictable multi-agent workflows. The findings also emphasize how MCP enhances contextual grounding and reduces operational errors, which are critical for applications requiring accurate reasoning and secure interaction with external systems.

Overall, MCP positions itself as a foundational layer for the next generation of agent-driven AI systems. Its capacity to unify tool interfaces, structure contextual information, and support scalable orchestration patterns shows strong potential for transforming how intelligent agents operate in complex, dynamic environments. Continued advancements in MCP research and ecosystem development are likely to play a central role in shaping future AI infrastructures built around interoperability, extensibility, and robust autonomous behavior.

REFERENCES

[1] **arXiv:2501.14249 [cs.LG],** Humanity's Last Exam
[2] Scale AI team, Humanity's Last Exam, Available: https://scale.com/leaderboard/humanitys_last_exam
[3] Model Context Protocol (MCP), Available: https://modelcontextprotocol.io
[4] **arXiv:2410.13360v3 [cs.CV]**, RAP: Retrieval-Augmented Personalisation for Multimodal Large Language Models
[5] **arXiv:2505.06416 [cs.CL],** ScaleMCP: Dynamic and Auto-Synchronizing Model Context Protocol Tools for LLM Agents
[6] arXiv:2502.05957 **[cs.AI],** AutoAgent: A Fully-Automated and Zero-Code Framework for LLM Agents
[7] **arXiv:2509.25292 [cs.CY]**, A Measurement Study of Model Context Protocol Ecosystem
[8] **arXiv:2506.13538 [cs.SE]**, Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers
[9] arXiv:2508.14704 **[cs.AI]**, MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers
[10] **arXiv:2503.23278 [cs.CR],** Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions
[11] **arXiv:2507.06250 [cs.CR]**, We Urgently Need Privilege Management in MCP: A Measurement of API Usage in MCP Ecosystems