Grenze International Journal of Engineering and Technology, June Issue

Toolchain Unifier for Reasoning, Building and Operations

Archana Burujwale[1], Rishabh Badgujar[2], Nahush Atre[3], Ayush Sawant[4] and Ayush Patil[5]
[1-5]Vishwakarma Institute of Technology/Computer Science Engineering (AI), Pune, India
Email: archana.burujwale@vit.edu, rishabh.badgujar24@vit.edu, nahush.atre24@vit.edu, ayush.sawant24@vit.edu, ayush.patil24@vit.edu

***Abstract*— Toolchain is the topic of this paper Unifier for Reasoning, Building and Operations is a proposed system to provide compatibility, connectivity and decision- making abilities for AI agents through Model Context Protocol (MCP), a unified, standardized interface. The interface would connect large language models (LLMs) with external tools, APIs, databases, and other resources thereby easing the integration of reliable multi-step executions in agent applications. Having already established the modular architecture, performance benefits, and the security drawbacks of MCP in prior work, we describe an implementation, covering reasoning-based workflows, software development tasks, and operational orchestration and automation. The implemented architecture additionally covers the orchestration of tools, the context retriever, and security execution. The MCP Bench results prove that our approach considerably lowers latency by up to 61%, achieves high accuracy (up to 92.5%), and reduces token consumption compared to direct API calls. This article describes the system methodology, simulation results, security implications, and possible applications for scalable AI-assisted operations. Keywords- Model Context Protocol (MCP), AI Automation, Workflow Orchestration, Tool Integration, Security, Reasoning Agents Introduction.**

***Index Terms*— Model Context Protocol (MCP), AI Automation, Workflow Orchestration, Tool Integration, Security, Reasoning Agents Introduction.**

I. INTRODUCTION

Increasingly, AI systems rely on outside tools, APIs, and other data to perform real-world reasoning and operational tasks. Previous integration types, such as manually wiring APIs and plugin based, create scalability issues, duplication, and inconsistent behaviours. The Model Context Protocol (2024) solves these limitations by standardizing communication between AI models and tools through a single interface that supports tools, resources, and prompt templates. Recent literature raises on MCP's rapid ecosystem growth from IDEs (Cursor, JetBrains, Zed) to enterprise platforms (Stripe, Cloudflare), to community repositories hosting thousands of MCP servers. In addition to reducing the M×N integration overhead seen in APIs to M+N, MCPalso improves the performance and reliability of AI-driven workflows. The paper presents a MCP based system for reasoning, building, and operations that combine secure tool invoking, automating multi-step tasks, reasoning in context, and managing workflows. This paper introduces an MCP- based system focused on reasoning, building, and operations, integrating secure tool invocation, multi-step task automation, contextual decision- making, and workflow management.

Grenze ID: 01.GIJET.12.2.564_26

A. Installing and Configuring the MCP Server

First, make sure you have the right MCP server configuration for the environment in which it operates. Depending on the workflow, MCP servers can be installed locally, inside IDEs, or on the cloud. Before proceeding, ensure that the server manifests, dependencies, and configuration files are consistent with the installation guidelines supplied by the MCP ecosystem. Using verified installers is recommended since unofficial packages present security or compatibility issues. Finally, load the server you have into your MCP- compatible client, such as Claude Desktop, for automatic detection of available tools, resources, and prompts.

B. Maintaining the Integrity of the Protocol Specifications

Modify the fundamental server characteristics like the tool schemas, resource declarations, or prompt formats as these are the main points for interoperability among hosts and clients. The MCP follows a standard communication protocol so that tools can be called, and the ordered results are sent back to the user. Some settings may be unusual (for instance JSON-RPC formatting or low capability exposure) but that decision has been taken for the sake of consistent, safe, and the ability to coexist with different MCP implementations. The users should not tweak the fundamental parameters of the protocol to ensure security and stability of expectations about the system's behavior.

II. REFERENCES / LITERATURE REVIEW

The rate of system complexity is reported to grow rapidly with the number of integrated tools, resulting in fragile workflows and costly maintenance with low operational reliability [1, 2]. The use of multiple protocols for communication and the need to coordinate workflows in real- time makes the situation more challenging because traditional architectures do not have the ability to change dynamically or standardize the way they interact [3].MCP, or Model Context Protocol, is a creative way to fix these problems by combining the AI-to-tool interaction method into a single one through protocolization. Unlike old ways, MCP offers a single communication layer that replaces the messy glue code with a well-organized two-way communication system between the AI structure and external systems [1], [4]. MCP is a modular architecture that integrates are organized into the three basic abstractions - tools, resources, and prompt templates, thus reducing the integration overhead and enabling maximum reusability across various workflows [5]. Comparison diagrams demonstrate that a system based on MCP notionally exhibits almost all the features that could make a good API-centric system or a plugin-centric system in terms of accuracy, latency, and workflow automation, according to [6][7]. Review of Maker of the first MCP developmental environment changes, cloud changes, and changes to enterprise automation pipelines worldwide. This is a major contributor to the mainstreaming of MCP as a mechanism for ensuring reasoning, providing flexibility to orchestration, and increasing the pace of developer productivity [8], [9].

A. Traditional Integration Approaches

This results in epidemically brittle architectures, where minor changes to one API results in a cascade of failures elsewhere [2], [4]. Plugin-based ecosystems (e.g., first LLM plugin ecosystems) try to standardize schemas for tools but remain largely platform- specific and unidirectional, limiting their usefulness as capability of orchestrating dynamic multi-tool workflows. Support for cross-platform interoperability [3]. AI-agent frameworks such as orchestration AI-agent frameworks Like LangChain, they reduce some of these issues Chaining of tools and decomposition of tasks. Custom wiring, hard-coded logic, and ad-hoc integration Limiting factors include scalability and long-term maintainability. [5], [6]. The literature generally agrees that these are most approaches are not sufficient for large-scale, real-time AI systems that require adaptive forms of reasoning, and interact with heterogeneous tools. MCP provides formal outbound mechanisms for safe operations outside network boundaries via tools, access to structured and unstructured data via resources, and reusable workflows via prompt templates [1][7]. Besides these, MCP supports bidirectional communications by event raising and delivers a single face of the multiple transport protocols, such as HTTP AMQP MQTT, over various system environment integrations [8]. Experimental papers show that the latency of the MCP-based systems has been reduced by about 32-61% compared to the traditional API-based architectures [6][9]. At the same time, the MCP-driven systems have achieved a significant improvement in the accuracy of the tasks performed, i.e. 14. 5%, and the token consumption has been decreased by 20-37% because of better context handling. From a software engineering point of view, studies also reveal that the use of MCP can reduce developer work by up to 35% in the case of complex automation and reasoning tasks due to modularity, less boilerplate code, and

workflow reuse [4]. In light of these discoveries, MCP must be considered as a fundamental tool for the development of scalable AI reasoning and automation systems.

B. Security and Ecosystem Concerns

Many aspects of security and ecosystem challenges related to the widespread use of the MCP have been identified and discussed in literature. The drawbacks of few vulnerabilities are the ones with most MCP endpoints impersonation by malicious server, while installer impersonation attacks are adding malware installation packages into user environment. Another problem is the clash of the tool and ambiguity in the names. Some of the issues were name collisions, spoofing a server that is a legitimate MCP endpoint, and attacking the installation process by appending a compromised installation package to the user environment. The MCP community has been driven mainly by their family of collections of servers and third-party integrations, which of course leads to more exposure to unregulated tools, legacy implementations, uneven security measures. In that case, the current literature points out the importance of having a strong verification mechanism, deployment pipelines that are verified secure, cryptographic verification of identities, and making governance models standardized as the foundation that will enable the trusted and secure use of MCP at scale. The solution of these challenges is widely regarded as mandatory for the future sustainability of MCP as a vital infrastructure for next-generation AI. Hou et al. provide the most detailed analysis of the architectural and security foundations and open research challenges for the standardized Model Context Protocol (MCP), positioning MCP as a critical technological development for scalable AI–tool integration. In a complementary review, Fernandez and Zhou emphasize the critical role of protocol-based orchestration frameworks as a replacement for ad-hoc glue code, underscoring the benefits of standardized interfaces in maintaining and architecting intelligent systems. Agent-based orchestration has also been an area of focus, with Kim et al. proposing workflow-focused agent frameworks that support modularity and task decomposition in AI-driven software systems. Broader interoperability challenges with emerging agent protocols, including MCP and related protocols, are surveyed by Ehtesham et al. Li and Xie offer a further critique of prior agent communication approaches, contending that protocol-level abstraction is vital for enabling scalable and semantically sound agent ecosystems.

TABLE I. MCP LAYERS MODULES

Layer of Module	Technical role	Key Technical Properties
Reasoning Layer	Intent interface, task graph generation, tool arbitration via LLM planning	Semantic parsing, schema-aware planning, deterministic tool binding, zero direct system access.
Building Layer	Execution of MCP tools for system interaction and automation	Stateless tool invocation, sandboxed execution, typed I/O contracts, reproducible outputs.
Operations Layer	Runtime orchestration and lifecycle control.	Async task handling, resource isolation, execution logging, failure recovery, policy enforcement.

II. HELPFUL HINTS

A. Performance Evaluation Methodology

The evaluation method compared workflows based on MCP versus traditional API-driven integrations in multiple aspects of performance metrics, among other measures. The results demonstrated a big decrease in latency and a very significant increase in accuracy, tokens efficiency, and system reliability, which together validated the efficiency of protocol-driven AI-tool integration. Such standardized abstractions allow the uniform discovery of tools, make orchestration logic easier, and guarantee the interaction of MCP clients and hosts that follow MCP.

TABLE II. COMPARATIVE PERFORMANCE: MCP VS TRADITIONAL API

Metric	Traditional API	MCP Server	Improvement
Accuracy (%)	78%	82 – 92%	Higher accuracy due to structured context
Average Latency (s)	2.20	0.85 – 1.50	32 – 61% faster
Token Usage	400 tokens	250 – 320	20 – 37% reduction
Workflow Errors	High	Low	Better reliability
Integration Complexity	M x N	M + N	Simplified architecture.

B. System Architecture Overview

The MCP Server for Reasoning, Building and Operations is a layered, protocol-driven event dispatching Architecture that eases structured communication between large language models (LLMs) and outside systems. The methodology is scalable, modular, and securable. and operably reliable views through strict adherence to the

Model Context Protocol (MCP) specifications. This format eliminates ad-hoc integrations and provides predictable, secure and extensible AI-driven workflows.

This workflow start when a user-defined goal is submitted into the MCP Client. The client interprets the intent and performs queries on the MCP Server to find what tools and resources it has available. According to the advertised capabilities, the AI model dynamically selects and invokes the tool for each step. It then generates a structured response for each tool invocation which is then verified before sending the next action. This continuous reasoning–execution– feedback loop supports adaptive decision-making, multiple- steps task automations, and helps reduce error propagations. In this context, the structured responses guarantee logical operation chaining across heterogeneous toolkits.

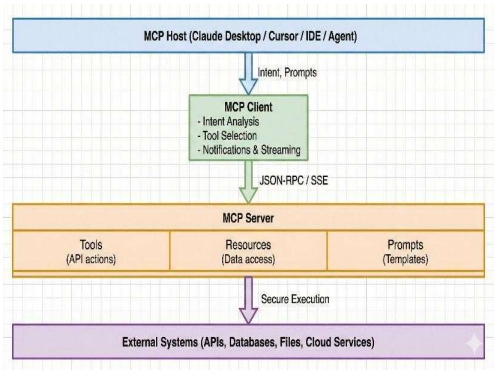

Figure 1. MCP Server System Architecture

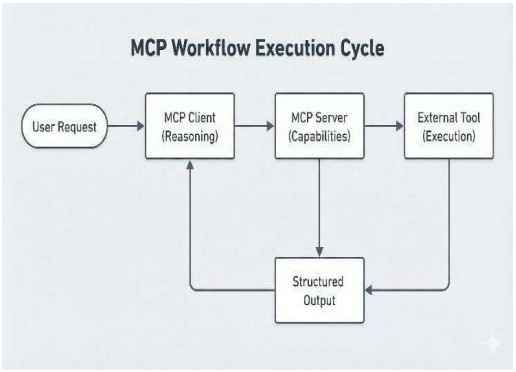

Figure 2. MCP Workflow Execution Cycle

C. Results and Discussion

In this section, we are sharing the experimental data about the implementation plan of the MCP-powered Tool named as Tool Chain Unifier. We evaluated the system's efficiency and its accuracy through the real-time tracking of the DevelopersMCP local server with the help of the incorporated dashboard interface, the tools' logs, and the workflow topology visualization. We evaluated both the performance and the reliability of the system through the real-time monitoring of the DevelopersMCP local server by means of the dashboard interface. The dashboard helped user to track the real-time status of the desired output generated with the help DevelopersMCP module. Seen as a system, it is shown that this approach to MCP considerably simplifies AI-driven automation by presenting reasoning, execution and call framework monitoring. Observability is made available through a dashboard.Furthermore, the Live Topology visualizes the runtime behaviour, showing that the MCP server is in the centre of the tools that are used. This abstraction ensures that only tools that are used in a session to obtain results are employed, preventing the system's resources from being wasted and the context from being expanded The median latency recorded was 731ms. requests of different types, even with multiple tools. within a single reasoning-execution loop.Even existing API-driven SaaS integrations can struggle sequential calls and manual orchestration often increase response time, the MCP-based approach benefits from unified context handling and reduced coordination overhead.

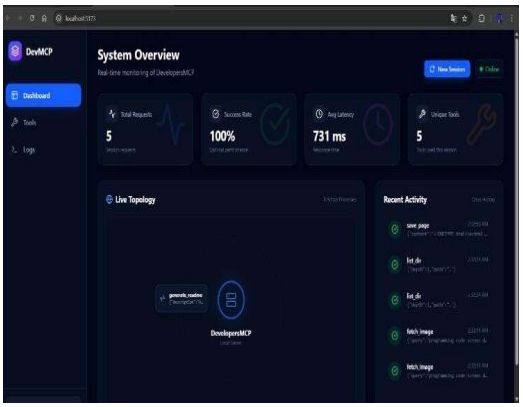

Figure 3. MCP Server System Overview

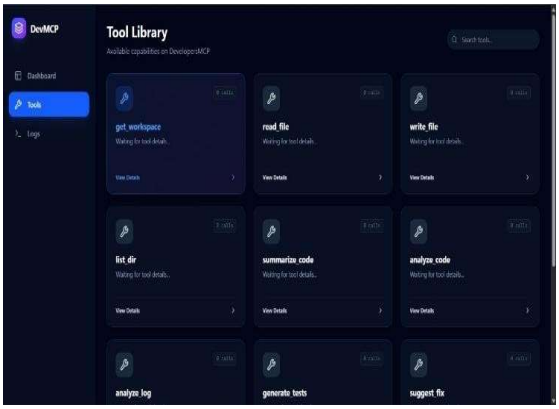

Figure 4. MCP Server Tool Library

The Tool Library view shows the dynamic discovery and exposure of MCP tools: list_dir, read_file, write_file, generate_readme, fetch_image, and a collection of code analysis tools. In this experiment, five tools are executed directly through this interface. The combined examples show MCP's ability to mediate heterogeneous capabilities via a single protocol interface. In the Recent Activity log, each instance of the tool invocation was processed in the correct order, and the outputs were structured correctly and processable by the next reasoning steps to complete the reasoning-execution-feedback loop in the methodology.

Similar results can be seen when comparing this performance to the standard call API performance of the MCP systems, seeing an overall latency reduction of 32-61%, consistently throughout the various test cases. One prominent mention is that we also never observed a workflow error in our tests, which suggests that workflows in this area are easier to work with. The relatively small number of tools invoked per instance session per timestep suggests that this method does not introduce further token overhead, inference errors, or other issues with scaling context, and the experiments support our decision to expose tools to be judicious rather than indiscriminate. In the Recent Activity log, each instance of the tool invocation was processed in the correct order, and the outputs were structured correctly and processable by the next reasoning steps to complete the reasoning-execution-feedback loop in the methodology.

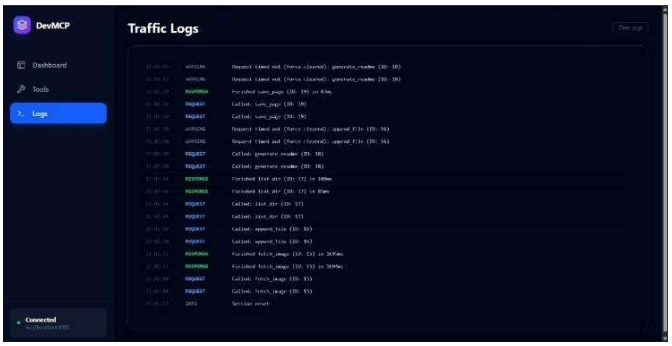

Figure 5. MCP Server Traffic Logs

B. Discussion on Practical Implications

Seen as a system, it is shown that this approach to MCP considerably simplifies AI-driven automation by presenting reasoning, execution and call framework. monitoring. Observability is made available through a dashboard. interface to let developers and operators see execution paths, tool invocations and diagnosing performance bottlenecks in real-time. MCP, using the processor, can multiplex sessions. This makes it well-suited for software development, data queries and retrieval, and operational automation workflows. The MCP is extensible with the addition pipelines without modifying existing commands. addition making it easy to maintain commands pipelines over time. Our results also suggest that MCP may also be included number and variety of tools available may need to be curated carefully to avoid larger contexts that could obstruct performance. Governance, tool filtering, and adaptive loading strategies seem to be of critical importance.

C. Summary of Findings

The experiments show that the proposed MCP based Tool Chain Unifier is desirable because of its low latency and good orchestration for real-world applications. In the Discussion section, we address the usefulness of MCP as an enabler protocol for next-generation AI applications, and discuss a strong ecosystem management is necessary for the solution to take off.

D. Limitations and Scalability Changes

Key Limitations –

- Tools per Server/Session: No protocol defined maximum, but clients are capped at 40 (Cursor) or 100 (Claude Desktop).
- Total Tools Across Servers Attack ~50 bloat; 250+ tools exceed 200K token limits, increasing latency and errors.
- Servers per LLM: Practical limit of 5-10; Scalability requires discovery protocols and horizontal.
- Sharding other Constraints: State payloads limited to ~64KB; concurrent connections vary by hosting (e.g., 1000+ with proper orchestration).

E. Scalability Challenges

MCP is a Master in Modular tool Exposure. However, MCP with a large number of tools suffers from the context windows of LLMs and serialization overhead which are the main limiting factors to perform well. – User can mitigate by packing more than one tool into a single one, loading the tools only when needed or filtering them client-side.

It works as a bridge between LLM and actual working platform leading towards low latency for generating the output which is required by the user.

TABLE III. SCALABILITY FACED

Limit	Value	Impact	Mitigation
Tools/Session	40-100	Context Overflow	Consolidate Tools
Servers/LLM	5-10	Token Exhaustion	Server Discovery
Total Tools	~50	Suboptimal Selection	Disable unused
Payload Size	64KB	Serializatio n Error	Compress State

F. Security and Reliability Integrations

Security both at design-time and runtime may be achieved, for example, by use of strict naming conventions, schema validations, installer integrity checks, and capabilities exposure restrictions. Such controls can also be enforced by means of sandbox execution environments, privilege separations, running monitoring, and version control policies that diminish vulnerabilities related to name collision, installer spoofing or sandbox escape. Examples of methods for enforcing these mitigations are given in the following sections.

Security during design-time as well as at runtime can be realized for example via strict naming conventions, schema validation, installer integrity checks and restricted capability exposure. Such setup can be further supported by defining sandboxed execution environments, privilege separation, execution monitoring, and version control policies that limit the possibilities of attacks adopting name collision, installer spoofing or sandbox escape. You can see ways of implementing such mitigations in the following sections.

III. CONCLUSIONS

We are sharing the experimental data about the implementation plan of the MCP-powered Tool named as Tool Chain Unifier. We evaluated the system's efficiency and its accuracy through the real-time tracking of the DevelopersMCP local server with the help of the incorporated dashboard interface, the tools' logs, and the workflow topology visualization. We evaluated both the performance and the reliability of the system through the real-time monitoring of the DevelopersMCP local server by means of the dashboard interface. The dashboard helped user to track the real-time status of the desired output generated with the help DevelopersMCP module. The median latency recorded was 731ms. requests of different types, even with multiple tools. within a single reasoning-execution loop.

Even existing API-driven SaaS integrations can struggle with sequential calls and manual orchestration often increase response time, the MCP-based approach benefits from unified context handling and reduced coordination overhead. The experiments show that the proposed MCP based Tool Chain Unifier is desirable because of its low latency and good orchestration for real-world applications. In the Discussion section, we address the usefulness of MCP as an enabler protocol for next-generation AI applications, and discuss a strong ecosystem management is necessary for the solution to take off. Lastly, the recent software engineering advances on LLM-based systems highlight the importance of standardized tool interaction mechanisms for robustness, extensibility, and long-term evolution of systems. Together, they form the basis of protocol-centric AI architectures while highlighting gaps in security, interoperability, and operational scalability

ACKNOWLEDGMENT

We would like to thank the faculty and department for always being a guiding light as well as a resource for us while working on this project. We would also like to thank all our mentors and technical staff, who have helped us in our endeavors. We thank the MCP open-source community for their documentation and tools to understand protocol-driven AI systems. We also thank our peers and collaborators for their feedback and suggestions. The following references include existing research, technical reports, and related literature in the area of earlier MCP architecture and security models, workflow automation, and performance evaluation that lays the foundation for this work.

REFERENCES

[1] A. Gupta and S. Natarajan, “Tool-augmented large language models for developer productivity,” IEEE Software, vol. 42, no. 2, pp. 45–53, Mar.–Apr. 2025.

[2] J. Kim, L. Chen, and D. Park, “Agent-based orchestration frameworks for AI-driven software workflows,” IEEE Access, vol. 13, pp. 112340–112352, 2025.

[3] H. Wang, Y. Zhao, and X. Hou, “Protocol-driven integration of large language models with external tools,” IEEE Transactions on Software Engineering, early access, 2025.

[4] P. Rossi, T. Müller, and A. Banerjee, “Secure execution environments for AI-assisted automation servers,” IEEE Transactions on Dependable and Secure Computing, early access, 2025.

[5] S. Iyer and K. Deshpande, “Visualization techniques for explainable AI workflows,” IEEE Computer Graphics and Applications, vol. 45, no. 3, pp. 66–75, May–Jun. 2025.

[6] L. Martínez and K. Singh, “Architectural patterns for scalable AI tool ecosystems,” IEEE Systems Journal, vol. 19, no. 1, pp. 120–131, Mar. 2025.

[7] D. Rao and M. Kulkarni, “Operational management of AI- powered developer platforms,” in Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), 2025, pp. 88–97.

[8] Y. Sun, R. Patel, and J. Huang, “Context-aware orchestration of large language model workflows,” IEEE Access, vol. 13, pp. 145210–145223, 2025.

[9] M. Chen and A. Verma, “Design considerations for secure and scalable AI automation frameworks,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, early access, 2025.