Model Context Protocol : Evolving AI learning

Ruslan Gavrilov
South East Technological University
Carlow, Ireland
ruslangavrilov118@gmail.com

I. Introduction (*Motivation*)

The rapidly evolving field of cloud computing and artificial Intelligence (AI) has taken over the world making AI an integral part of both technology and everyday life. Currently Large Language Models (LLMs) such as GPT and Claude can integrate tools to help with reasoning and actions, the difficulty occurs when you want to build a multipurpose AI, it is difficult to manage and stack all of these tools atop each other. This study aims to showcase The Model Context Protocol by exploring what it is, and how it's integration can help LLMs by providing a standardized way to connect Ai models to different data sources and tools. The primary motivation behind this study is to bring awareness to the significance of MCPs in advancing AI, thereby allowing for more effective human-AI interactions. Additionally, this research aims to highlight emerging trends in AI development, particularly in the optimization of contextual understanding within LLMs.

A. Introduction (Background)

LLMs, at their core, are ultimately incapable of doing anything meaning. The first use cases for LLM chat bots were a question-and-answer service where the LLM would reason a response to a query that it had received. The LLM would use predictive reasoning to answer a given prompt or predict the next outcome. LLMs are primarily based on human neural networks to create a way to reason, this was the most an LLM by itself could accomplish.

B. Introduction (Current State)

The next stage of LLM evolution was the creation of tools by developers. These tools were responsible for giving LLMs access to APIs. For example, Models such as GPT can search the internet to present to you the outcome of their search. A LLM by itself cannot do this, thus the introduction of tools further evolved the abilities of Models to build upon them and expand their possibilities. However, a limitation occurs when attempting to expand the capabilities of model by adding multiple tools, attempting to create a multi-purpose Model. The process of stacking tools together inside one LLM is problematic and difficult as it requires constant maintenance and search to find specific tools and integrate them to work cohesively together for your LLM to not hallucinate (Generating responses that are factually incorrect). As a developer you may need to adjust existing code to work with a tools updated API in case it gets changed as well as maintain the correct structure and call your AI agent.

The introduction of the Multi Context Protocol (MCP), aims to standardize the way in which an AI agent may call specific provided tools, making it easier to maintain the code for the individual queries or API calls, as well as create a standardized and centralized way of using given tools, thus acting as an adapter for many tools.

II. System Architecture

The Multi Context Protocol works as an abstraction in the transport layer. It acts as a translator between many stated AI tools and controls the output of them in a concise, universal way for LLMs to easily understand and utilize.

The Multi Context Protocol architecture consists of 3 major parts.

1. MCP Client: This is the front side client responsible for the input of the request and the output of the response. For example, Ai models such as Claude. The client is responsible for maintaining a list of available MCP servers or tools that may be required when answering a query or performing a request.
2. MCP Server: The MCP server is responsible for allowing the AI model to access the context required to perform the initial request. The server is responsible for using a service, and returning that information to the server, which is then responsible for passing that information back to your AI model through the MCP Protocol, so your AI model may complete its prompt. This server is maintained by the provider of the tool or service. As a builder of a tool, you maintain the service so that different AI models can access what you provide and become more capable in terms of their functionality, as a generalized standard is introduced here.
3. Service: This is the individual service that you want your tool to do. This could be for example, requesting global stock data, or global weather data. The service you are providing is the current stake of a given context, and you are returning that to the MCP server, which is then responsible for returning that data to the AI model through the MCP protocol so that your AI model can receive what it needs in a universal and concise way. There are two types of services, Local and remote. Remote would be as described above, primarily focused on external and remote APIs, where the local would be primarily focused on aspects like local databases and local services.

Both the MCP Server as well as the client support their own primitives in order for them to work correctly.

MCP Servers will expose three primitives: Tools, Resources and Prompts.

1. Prompts: These are the instructions or templates that can be inserted into the LLM context, they guide how the model should approach certain tasks or data. Each prompt is defined with a name or unique identifier for the prompt, a human readable description, and an optional list of arguments for the prompt. Prompts should be made discoverable by a prompts/list endpoint.

2. Resources: These are structured data objects that can be used inside of the LLMs context window. They allow the model to reference external information. Servers expose a list of resources via a resources/list endpoint where each resource contains an identifier, human readable name and an optional description.
3. Tools: Executable functions that the LLM can call to receive information or perform actions outside its context, like querying a database or editing a file. Each tool is defined with a name, human readable description and JSON schema for the tool's parameters.

MCP Clients will expose two primitives: Roots and Sampling.

1. Roots: This creates a secure channel for file access. It allows the AI client to safely work with files on the client's local system. It should be capable of opening documents, reading code or analyzing files, without giving unrestricted access to your entire file system
2. Sampling: This allows a server to request an LLMs help when needed. For example, If an MCP server is analyzing a local database schema, it needs to generate a query, an LLM should be able to format that query through this primitive. It creates a two-way interaction where the AI and the tools can send requests back and forth to each other making the system a lot more flexible.

Experiment:

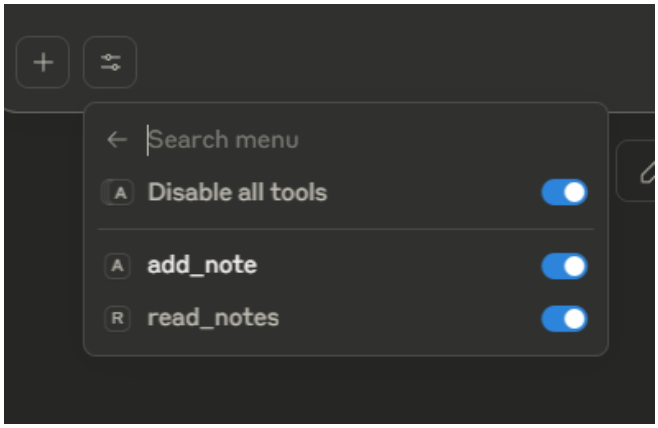
To fully grasp the concept of MCP, I dove into how to set up my own MCP Server to see how easy it would be for larger companies to set up their own and provide their tools to the broader world.

Creating an MCP Server requires you to install the FastMCP package within python. For this experiment, I aimed to quickly create a system that would allow me to save notes, read them and provide a summary of them.

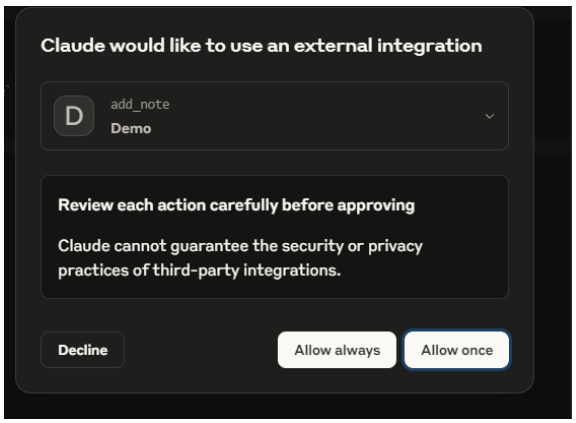
It was quite easy to set up and get an MCP server running. All the protocol requires you to create is a function under the @mcp.tool() header with the required information and description. The code would be implemented as any basic python function; I simply had to define the return value.

Running uv run mcp install main.py automatically created a config inside Claude desktop, that would allow it to find the file path of the created MCP service and reference it when I would ask a question. The automatic configuration tells Claude which command to run along with which file to execute.

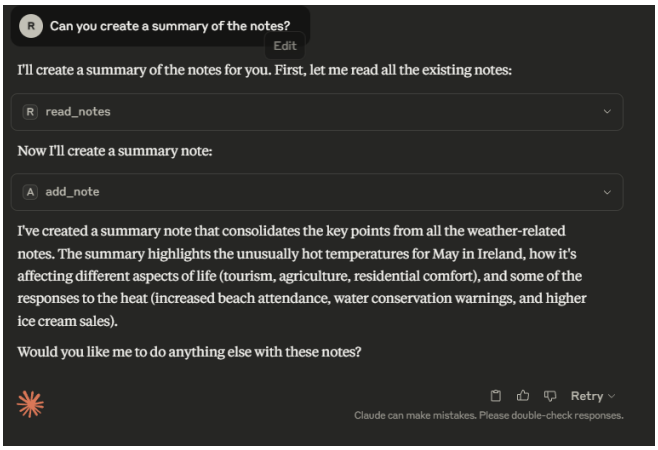
The tools automatically get recognised by Claude as seen under the tool section of the chat box.



With the tools, resources and prompts done inside of my file and completely accessible by Claude, I am able to freely create notes with my AI agent. You can see a prompt appear in Claude verifying access to your specific tool.



With the tools that I have created seen inside of the Main.py files, I am able to ask Claude to create distinct notes about any topic and add them to my designated notes resource, I was able to ask it to document how hot the weather has been in Ireland. Claude can directly access the tools present with the correct MCP set up and note down notes about the Irish weather. I am also able to ask it to quickly summaries my notes, where it can read my notes.txt document, access the information and summarize the contexts of said document.



Here you can see the initial prompt of me telling it to summarise what is present in the existing notes. With the context of my prompt, Claude is able to recognise that it needs to access the read_notes mcp tool that will allow it to create the notes currently stored inside of the notes.txt file, in a larger context, this function may be used to access a local or cloud database to retrieve values, but for this short example I opted for a simple document. With the knowledge of the current notes, It is able to fulfil my request of summarising my notes and choses to append the quick summary to the end of the notes.txt file and tell me a brief summary of the current weather in Ireland as I had asked Claude to generate 10 notes about the current weather in Ireland.

With this I have succeeded in creating a simple local MCP server that external AI agents could easily understand and use.

Discussion

This experiment provided valuable insight into the practical implementation of a Multi Context Protocol (MCP) server and its potential scalability for broader use cases. By setting up a local MCP server using the FastMCP Python package, I discovered how user-defined tools can be exposed as functions within an AI assistant interface such as Claude. The process was surprisingly straightforward: creating an MCP-compliant function required only a decorator (@mcp.tool), a brief description, and a standard Python function body with a clear return type. Once configured, running uv run mcp install main.py automatically generated the appropriate configuration file, allowing Claude to recognize the tool and reference the associated script on demand. This easy integration enabled Claude to access custom tools directly, such as those used for notetaking, summarization, and contextual querying. The AI agent was able to store new notes (e.g weather observations), retrieve them, and generate summaries with minimal prompting. These outcomes show how MCP can act as a lightweight yet powerful protocol for exposing tools to AI agents, significantly lowering the barrier for individuals and organizations to deploy functional, interactive backends for their own use or broader distribution.

Importantly, this approach has powerful implications for companies seeking to enhance AI agent capabilities and broadcast their services over a constantly evolving AI landscape. By exposing internal tools, databases, or services as MCP endpoints, companies can give AI agents-controlled access to domain-specific functions, such as generating invoices, querying inventory, or summarizing client histories, without compromising infrastructure or requiring full model retraining. This creates opportunities for highly specialised AI assistants that can operate in controlled or sensitive environments using carefully created tools. Also, teams can rapidly prototype new functionality and iterate on workflows with minimal difficulty. In effect, MCP bridges the gap between general-purpose AI and company-specific operations, making AI agents significantly more useful, context-aware, and action-oriented in real-world scenarios.

Conclusion

The Modular Connect Protocol (MCP), introduced in November 2024, has rapidly gained popularity throughout early 2025 and is emerging as the leading global standard for integrating Tools with AI Agents. In an industry lacking a unified standard for AI tool interoperability, MCP provides a much-needed framework that contains seamless and scalable connections between AI models and external tools.

Traditionally, integrating an AI Agent (N) with multiple tools (M) required an exponential number of integrations (N × M), leading to complex, fragmented, and resource-intensive development processes. MCP addresses this challenge by restructuring the integration model into an additive N + M relationship, significantly simplifying tool integration. This breakthrough reduces duplication of effort and accelerates innovation by allowing developers to create tools and AI agents independently while ensuring reliable interoperability through the MCP standard.

While MCP itself is a specification for how large language models (LLMs) interact with tools, its significance lies in establishing a foundational ecosystem. This ecosystem promotes adaptability and universality, enabling AI platforms such as Claude and others to evolve efficiently by adopting a shared language for tool usage. As a result, MCP empowers AI to extend beyond current limitations, fostering a collaborative environment where tools can be universally applied and rapidly improved upon.

Further Research

To further investigate the concept and implementation of MCP, I would investigate the security and general maintenance of such AI tools with the ever-evolving landscape of AI Agents. With how fast technology seems to be progressing, a constant upkeep is required by providers of MCP servers to continuously and seamlessly integrate with agents such as Claude. Advances made inside of AI may prove to make MCP obsolete or a golden standard inside of this space so research into maintenance would have to be done.

Security could also be a concern with AI agents accessing possibly sensitive data. Restrictions would need to be applied within MCP servers and their available tools as AI Agents may hallucinate or maliciously retrieve any form of data that a provider may not want the Agent to know about, causing concern in relation to security.

Finally, developing your own client or AI agent, like Claude, would help further the understanding of how these agents effectively communicate with the MCP servers that are set up.

Practical Applications

MCP tools could be adopted globally to create an easy-to-use marketplace for tools provided by large companies. Other companies may choose to integrate these tools into their own projects without the need for custom creating a tool which may already exist to a high standard. These tools can be leveraged and used by millions of developers to further increase the capabilities of AI agents and projects. Communities can also contribute to the MCP ecosystem by creating open-source tools or building modular AI solutions to further the knowledge in this field and avoid redundancy.

In conclusion, the integration of MCP servers and a global standard for tools would be beneficial for developers across the world to maximise AI agent capabilities and eliminate redundant re-creation of code by using a set global standard.

REFERENCES

[1] Model Context Protocol. "Introduction to MCP," *Model Context Protocol*, 2024. [Online]. Available: https://modelcontextprotocol.io/introduction

[2] Anthropic. "Introducing the Model Context Protocol (MCP)," *Anthropic Newsroom*, May 2024. [Online]. Available: https://www.anthropic.com/news/model-context-protocol

[3] Tech with Tim, "Build anything with a custom MCP server", [Online]. Available: https://www.youtube.com/watch?v=-8k9lGpGQ6g