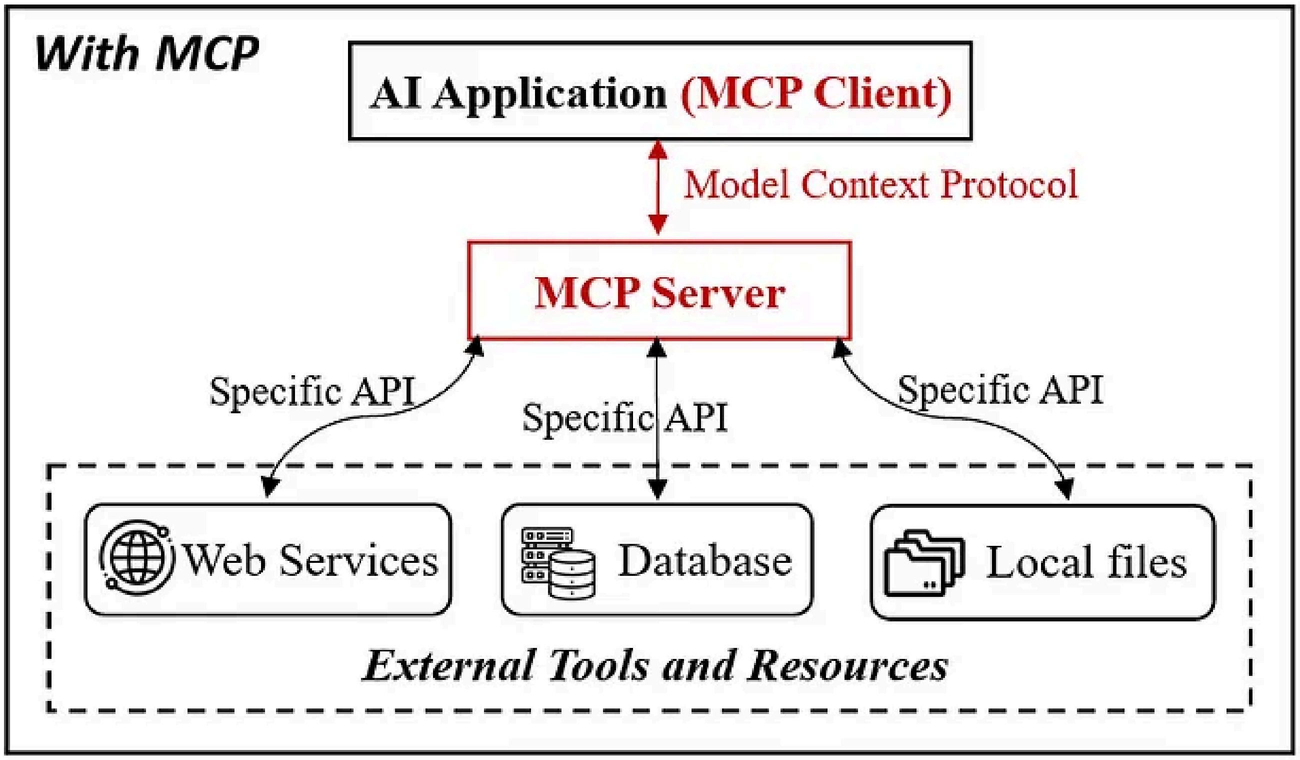
Level Up Coding

Member-only story

What are Model Context Protocols (MCPs) and Should you use them?

AI is growing up — and MCPs might be the common language it needs to talk to tools, orchestrate tasks, and work smarter across platforms.

Cristian Leo · Following
Published in Level Up Coding · 10 min read · 1 hour ago



Agent Tool Invocation with MCPs — Image extracted from Research Paper "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions"

AI is facing a bit of a communication problem. Imagine a workshop filled with brilliant craftspeople, each speaking a different language. That's kind of where we are with AI and external tools right now. We have incredibly capable AI models, and a vast universe of specialized tools and data sources out there — APIs, databases, cloud services — but getting them to talk to each other, to work **together** smoothly? Well, that's been quite the puzzle. As the researchers in "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions" point out, despite all the advancements, integrating these tools still feels… **fragmented.** Developers are often stuck manually defining interfaces, wrestling with authentication for each service, and basically re-inventing the wheel every time they want their AI to use a new tool.

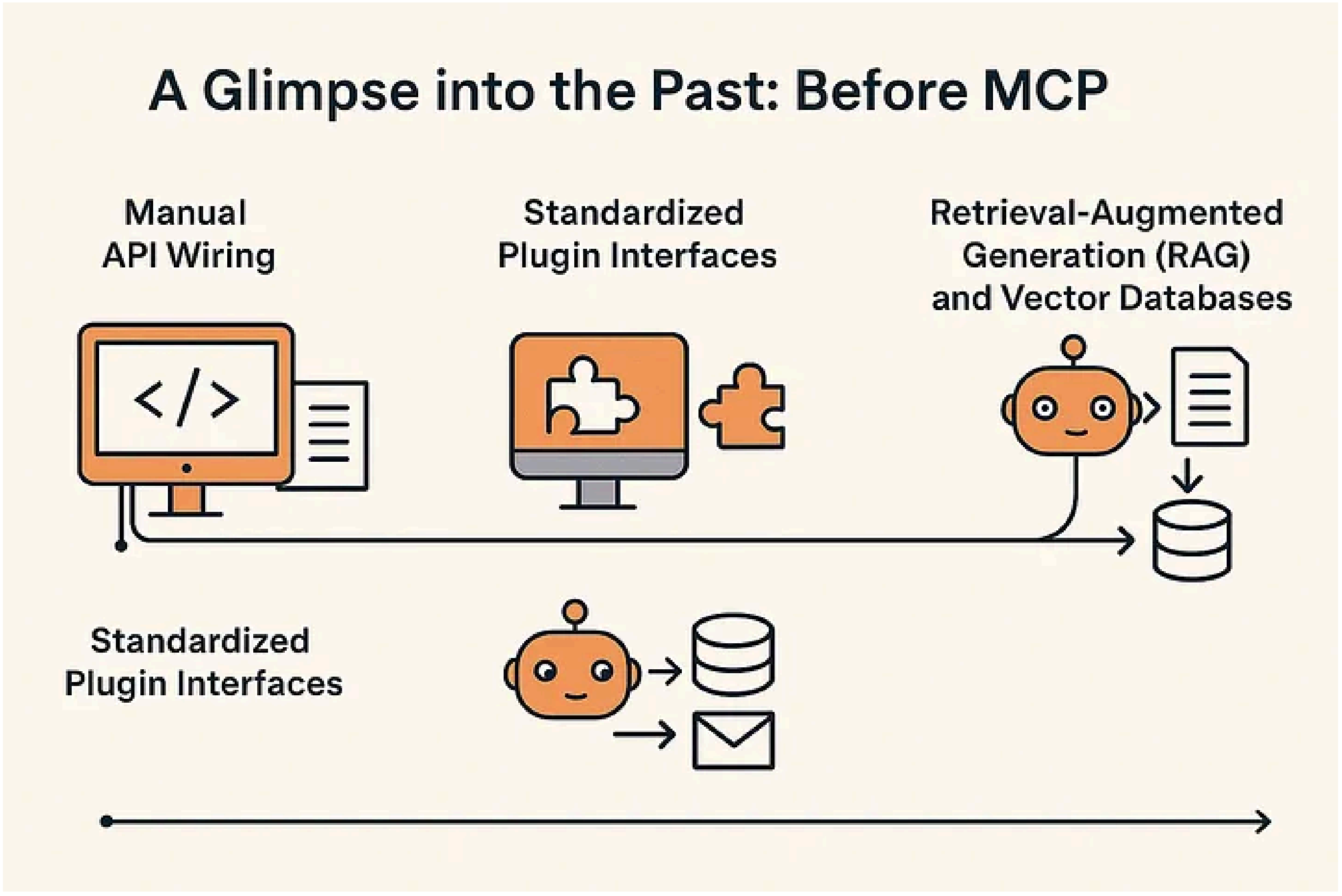
We've seen some attempts to smooth things over, of course. Plugin architectures, for instance, offered a more structured way, and frameworks like LangChain have helped with orchestration. And Retrieval-Augmented Generation (RAG) has been brilliant in giving models access to knowledge. Yet, if you ask me, these felt like steps in the right direction, but maybe not quite the leap we needed. They often still relied on pre-defined pathways, limiting the dynamic, on-the-fly tool selection that truly autonomous agents would require. It's like having a translator who only knows a few phrases — helpful in simple situations, but not for complex, nuanced conversations.

Then, along comes the Model Context Protocol, or MCP. Think of it as a common language, a standardized protocol that aims to make AI-to-tool interactions genuinely seamless and flexible. Instead of pre-set paths, MCP is designed to let AI agents dynamically discover, choose, and orchestrate tools based on the task at hand. It even considers the human element, allowing for human-in-the-loop interventions when needed. Could MCP be the key to unlocking truly interconnected AI ecosystems? It's certainly generating a lot of buzz, and as the paper highlights, it's rapidly gaining traction in the industry.

So, let's dive in and explore what MCP is all about, how it works, and why it might just be something you should be paying attention to. Let's get started.

A Glimpse into the Past: Before MCP

Before MCP, if an AI needed to use an external tool, developers had a few main paths to choose from, each with its own set of trade-offs.


A Glimpse into the Past: Before MCP
Manual API Wiring
Standardized Plugin Interfaces
Retrieval-Augmented Generation (RAG) and Vector Databases
Standardized Plugin Interfaces

An evolution of specialized LLM usage — Image by Author

The Era of Manual API Wiring

The most fundamental approach, and frankly, still quite common, is what we might call "manual API wiring." Imagine you want your AI to fetch the weather forecast. You'd typically go directly to a weather API provider, study their documentation, and then write custom code to make requests, handle authentication, parse the data, and deal with any errors. And if you wanted to add another tool, say, to send emails? You'd repeat the process, writing more custom code, essentially building a unique connection for each and every tool your AI needed.

Now, while this approach offers maximum control — you're literally crafting each interaction by hand — it can quickly become… well, a bit of a coding jungle, wouldn't you agree? Think about the complexity scaling with each new tool you add. It's like building a house brick by brick, where every brick is slightly different and needs a custom-mixed mortar. While you get a house, the process isn't exactly scalable or easy to maintain in the long run. And if something breaks? Troubleshooting and updating these bespoke connections can become a real headache, leading to systems that are, as the paper describes, "tightly coupled and fragile"

Standardized Plugin Interfaces

Recognizing the pain points of manual wiring, the industry naturally started looking for ways to standardize things. Plugin-based interfaces, like those pioneered by OpenAI with ChatGPT Plugins, emerged as a promising step. The idea was to define standardized API schemas, like OpenAPI, so that developers could create tools that "plug in" to AI models in a more uniform way. Think of it like standardized electrical outlets — finally, a bit of consistency!

For example, within the OpenAI Plugin ecosystem, you could find plugins like Zapier that allowed models to perform predefined actions like sending emails or updating CRM records. This was a definite improvement, offering a degree of reusability and simplifying some aspects of integration. However, these interactions were "often one-directional and could not maintain state or coordinate multiple steps in a task".

Furthermore, these plugin ecosystems often became platform-specific. LLM app stores, as they are called, like ByteDance Coze and Tencent Yuanqi, sprung up, each offering their own plugin stores. While expanding the available tool options, this created "isolated ecosystems where plugins are platform-specific, limiting cross-platform compatibility". We moved from a jungle of custom wiring to a collection of walled gardens — better organized, perhaps, but still not truly interconnected.

AI Agent Tool Integration

As AI agents grew in sophistication, so did the need to orchestrate more complex workflows involving multiple tools. AI agent frameworks, like LangChain and LlamaIndex, emerged to address this. These frameworks provided structured ways for models to invoke external tools through predefined interfaces, improving automation and adaptability.

These frameworks certainly streamlined the process, offering a more organized way to manage tool interactions. However, the paper rightly points out that "integrating and maintaining these tools remained largely manual, requiring custom implementations". It's like conducting an orchestra where you still have to hand-tune each instrument before every performance. While the orchestration is improved, the underlying effort of setting up and managing the tools remained significant and grew with the number of tools involved. We were getting closer to a symphony, but still with a lot of manual setup behind the scenes.

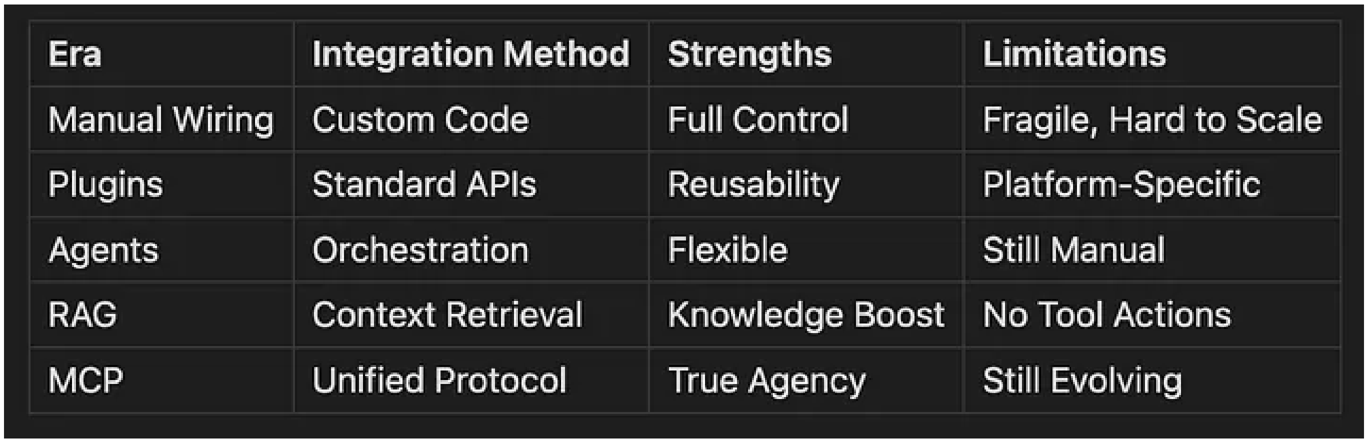
Retrieval-Augmented Generation (RAG) and Vector Databases

Another important development in AI tooling has been Retrieval-Augmented Generation (RAG), often coupled with vector databases. RAG allows models to access and incorporate external knowledge into their responses, overcoming the limitations of their training data. Vector databases made it efficient to retrieve relevant information from vast datasets. This was a significant step forward in making AI more knowledgeable and context-aware.

RAG systems could, for instance, retrieve relevant sections from a product documentation database to assist a customer support AI. This addressed the "knowledge cut-off" problem and improved accuracy. However, it did not inherently allow models to perform active operations. If our customer support AI needed to update records or escalate an issue, RAG alone couldn't bridge that gap.

The Promise of MCP

So, tracing this evolution, we see a clear progression: from the complexities of manual wiring, to the structured but siloed plugin interfaces, then to orchestration frameworks that still required significant manual effort, and finally, to knowledge retrieval systems that lacked active operational capabilities. Each step addressed certain limitations but also revealed new challenges. It's in this landscape that MCP emerges, promising to take us to the next level. MCP offers "a standardized protocol that enables AI agents to seamlessly invoke, interact with, and chain multiple tools through a unified interface". This is the key — a unified interface, designed for dynamic interaction and orchestration, going beyond passive information retrieval to enable true AI agency. But how does it actually work? That's what we'll explore next.



Era	Integration Method	Strengths	Limitations
Manual Wiring	Custom Code	Full Control	Fragile, Hard to Scale
Plugins	Standard APIs	Reusability	Platform-Specific
Agents	Orchestration	Flexible	Still Manual
RAG	Context Retrieval	Knowledge Boost	No Tool Actions
MCP	Unified Protocol	True Agency	Still Evolving

Agentic Tool Usage Evolution — Image by Author

Decoding MCP

Alright, so we've journeyed through the evolution of AI tooling and seen why MCP is generating so much excitement. But how does this protocol actually work?

Host, Client, and Server

The MCP architecture, at its heart, is built around three core components, which I like to think of as a collaborative trio: the **MCP Host**, the **MCP Client**, and the **MCP Server**. These three work together to enable that seamless communication we've been talking about. Let's break down each of their roles.

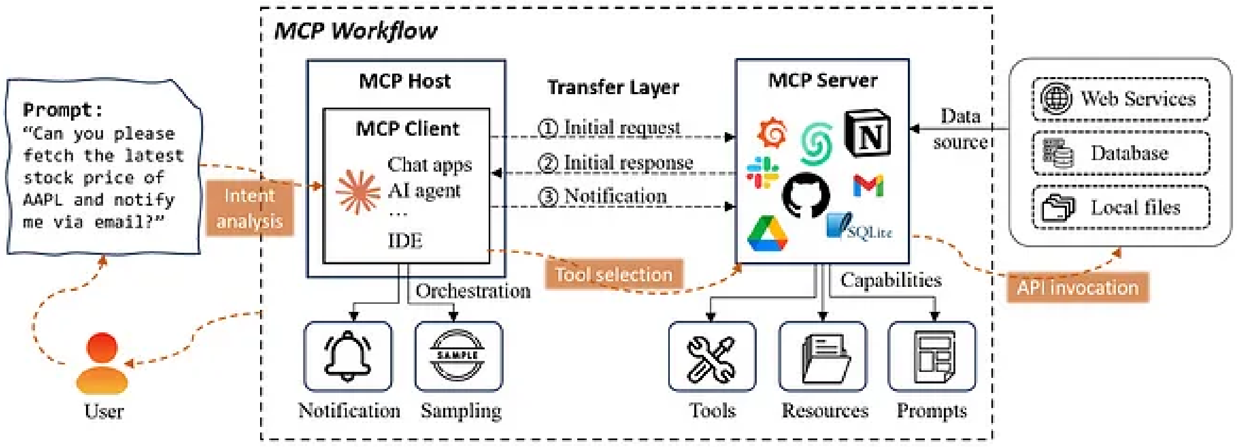
First, we have the **MCP Host**. This, in essence, is the AI application itself — the entity that's driving the whole process. Think of it as the conductor of our orchestra. Examples of MCP Hosts could be applications like Claude Desktop, Cursor, or even a custom-built AI agent designed for a specific task. The key is that the Host provides the environment where the AI-driven tasks are executed and where the MCP Client resides. It's the home base for the whole operation.

Next up is the **MCP Client.** This is the intermediary, the communication manager, if you will, operating within the Host environment. Its main job is to handle the dialogue between the MCP Host and one or more MCP Servers. The Client is responsible for initiating requests, asking MCP Servers about their capabilities, and then retrieving the responses. It's like the translator in our Tower of Babel analogy, ensuring smooth communication between different languages. Crucially, the MCP Client also manages notifications from the Servers, keeping the Host informed about the progress of tasks and the status of the system. And, interestingly, it even performs "sampling" — collecting data on tool usage and performance.

Finally, we arrive at the **MCP Server**. This is the bridge to the external world, the gateway to tools and resources. The MCP Server is what enables the Host and Client to actually access and execute operations on external systems. Each MCP Server is essentially in charge of offering three core capabilities: **Tools**, **Resources**, and **Prompts**.

- **Tools**: These are what enable external operations. They are the functions that the MCP Server can invoke on external services and APIs. When the Client requests an operation, the Server identifies the appropriate tool, uses it to interact with the external service, and then sends the result back. What's particularly neat, and different from traditional function calling, is that MCP Tools streamline the process — the AI model can autonomously select and invoke the right tool based on the context, without needing multiple separate steps. Once set up, these Tools follow a "supply-and-consume" model, making them modular and reusable — a significant step towards efficiency, wouldn't you agree?
- **Resources**: These are about data access. MCP Servers can expose various datasets to AI models — data from local storage, databases, cloud platforms, you name it. When an AI model needs specific data, the Server retrieves and processes it, enabling data-driven decisions. Imagine an MCP Server connected to a customer interaction log database — a recommendation system could use this resource to access customer history and make better suggestions.
- **Prompts**: This is where workflow optimization comes in. MCP Servers can offer pre-defined "Prompts" — templates and workflows that the Server generates and maintains. These are designed to optimize AI responses and streamline repetitive tasks. Think of them as reusable scripts or blueprints for common operations. For example, a customer support chatbot might use prompt templates to ensure consistent and accurate responses, or an annotation task could use them to maintain data labeling consistency.

The MCP Workflow



The MCP Workflow — Image extracted from Research Paper "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions"

The image above (Figure 2 from the research paper) illustrates a classic MCP workflow. Imagine a user gives a prompt to an AI application — something like, "Can you please fetch the latest stock price of AAPL and notify me via email?"

1. **Intent Analysis** (within the MCP Client): The process starts with the MCP Client receiving this user prompt from the MCP Host. The Client's first task is to analyze the intent behind the prompt. It needs to figure out what the user actually wants to do. In our example, the intent is twofold: get the AAPL stock price and send an email notification.
2. **Tool Selection** (via MCP Server): Next, the MCP Client communicates with MCP Servers to figure out which tools are available and appropriate for fulfilling the intent. The Client queries the Servers, asking about their

might select a "stock price API tool" and an "email sending tool."

3. **Orchestration and API Invocation**: Once the tools are selected, the MCP Client orchestrates the workflow. It sends requests to the appropriate MCP Servers to *invoke* the selected tools. This is where the actual API calls to external services happen. The Server executes the tools, interacting with external APIs (like a stock price API and an email service API), retrieves the information, and processes it.
4. **Response and Notification**: Finally, the MCP Server sends the results back to the MCP Client. The Client, in turn, delivers the processed information back to the MCP Host, and ultimately, to the user. In our example, the user gets notified of the AAPL stock price, likely via email as requested. Throughout this process, the MCP Client also handles notifications from the Servers, providing real-time updates about task progress.

This workflow allows for dynamic tool selection, automated orchestration, and seamless interaction, all thanks to the standardized MCP protocol. And it's all underpinned by a robust **Transport Layer.** This layer handles the transmission of requests, responses, and notifications, making sure everything happens reliably and securely.

Conclusion

Model Context Protocol is more than just another technical specification. It represents a significant shift in how we think about building and deploying AI systems..

MCP could be laying the foundation for the next era of AI — an era of interconnected AI. Imagine a future where AI agents can fluidly navigate across different platforms, dynamically select the best tools for the job, and orchestrate complex tasks with minimal human intervention.

So, what are your thoughts on the Model Context Protocol? Are you as intrigued by its possibilities? What challenges do you see as most critical to address? Let's continue this conversation in the comments below. Share your perspectives, your questions, and your insights. And if you found this exploration valuable, be sure to follow me for more deep dives into the ever-evolving world of Data Science and AI. Let's learn, explore, and build this future together.

Data Science Artificial Intelligence Large Language Models Software Development Technology

Published in Level Up Coding

Follow

222K Followers · Last published 1 hour ago

Coding tutorials and news. The developer homepage gitconnected.com && skilled.dev && levelup.dev

Written by Cristian Leo

Following

34K Followers · 13 Following

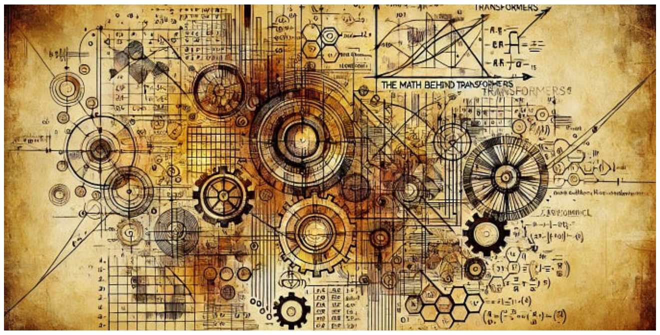
Data Scientist @ Amazon with a passion about recreating all the popular machine learning algorithm from scratch.

No responses yet

Alex Mylnikov

What are your thoughts?

More from Cristian Leo and Level Up Coding

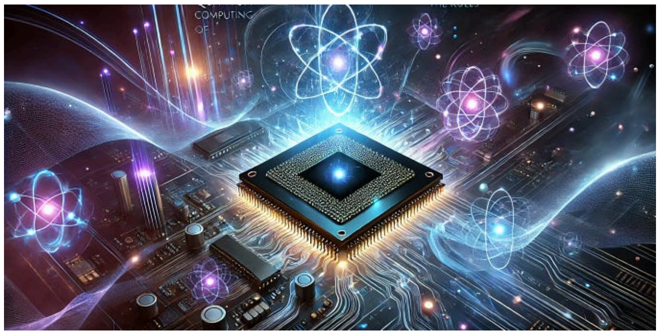


Cristian Leo

The Math Behind Transformers

Deep Dive into the Transformer Architecture, the key element of LLMs. Let's explore its...

Jul 25, 2024 1.1K 12

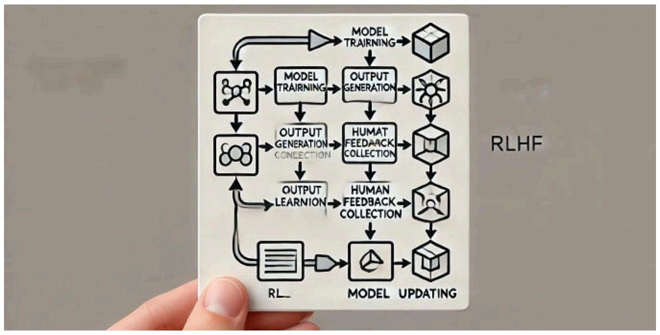


In Data Science Collective by Cristian Leo

How Qubits Are Rewriting the Rules of Computation

From Classical Certainty to Quantum Possibility: Exploring the Science, Math, and...

Mar 24 84



In TDS Archive by Cristian Leo

Reinforcement Learning 101: Building a RL Agent

Decoding the Math behind Reinforcement Learning, introducing the RL Framework, an...

Feb 19, 2024 906 10

In Level Up Coding by Cristian Leo

Can We Really Trust Human Feedback?

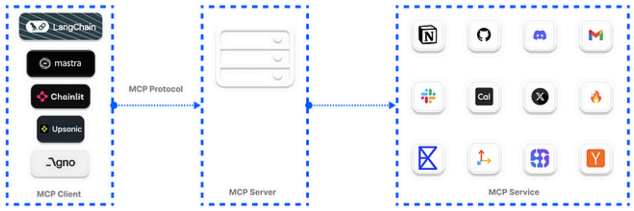
Exploring the Limits and Potential of Reinforcement Learning from Human...

Mar 17 156 1

See all from Cristian Leo

See all from Level Up Coding

Recommended from Medium

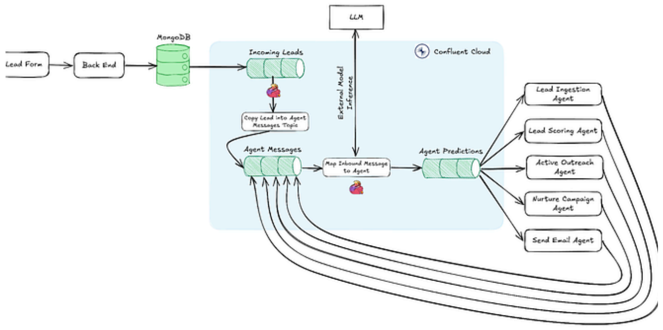


Amos Gyamfi

The Top 7 MCP-Supported AI Frameworks

Create AI apps with Python and Typescript frameworks that leverage MCP servers to...

6d ago 634 14

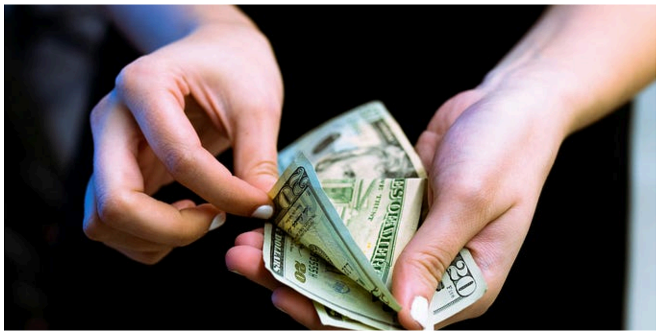


Sean Falconer

How to Build a Multi-Agent Orchestrator Using Flink and Kafka

Build a scalable, event-driven multi-agent orchestrator using Apache Flink and Kafka.

Mar 31 150 2



In Learn AI for Profit by Nipuna Maduranga

You Can Make Money With AI Without Quitting Your Job

I'm doing it, 2 hours a day

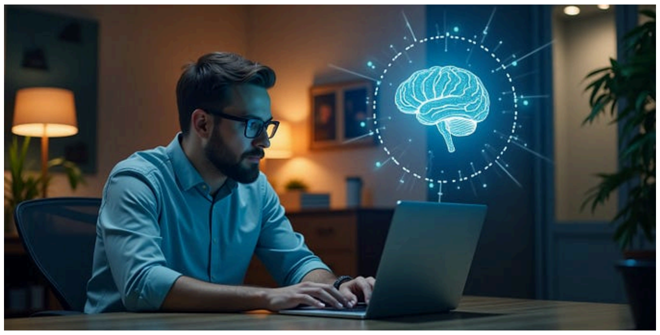
Mar 24 2.6K 120

In Binome by Craig Li, Ph.D

Introduction to Graphite — An Event Driven AI Agent Framework

Graphite is an open-source framework for building domain-specific AI assistants using...

6d ago 165 2



OPEN SOURCE
AI PROJECTS
TOP 10 OPEN-SOURCE AI PROJECTS THAT BLEW MY MIND AS A DEVELOPER

In Mr. Plan ₿ Publication by Ashen Thilakarathna

Build Smarter AI Agents in Minutes—For Less Than $0!

Discover the Secret Tool (MCP) That's Revolutionizing AI Development—No Codin...

Mar 30 1.5K 35

In Let's Code Future by Let's Code Future

Top 10 Open-Source AI Projects That Blew My Mind as a Developer

You need to checkout

Mar 29 490 11

See more recommendations