Balancing Human and Machine in Development

11 min read · Apr 1, 2025

Bart Van der Auweraert

When ChatGPT first emerged about two years ago and created waves across the tech industry, I immediately recognized its potential beyond the novelty factor. While many colleagues were using it to write poems about cats or solve puzzles, I began methodically exploring how it could enhance my daily work as a Technical Architect.

As a Technical Architect and Full Stack Developer focused on AI-driven solutions, finding the right workflow has been crucial to handling the complexity of modern development. Over the past two years, I've crafted what I call my "AI-enhanced productivity stack" — a collection of tools that helps me ship code faster without sacrificing quality.

In this article, I'll break down how I organize my daily work using VSCode, Roo Code, Gemini 2.5, Perplexity, and Claude — and why this specific combination has been a game-changer for my .NET development and architecture work.

My AI Journey: From Early Experiments to a Complete System

My journey with AI tools began when ChatGPT first appeared. While many were debating whether it would replace developers, I was quietly figuring out how to make it enhance my capabilities.

In those early days, I was among the first wave of GitHub Copilot users, amazed at how it could complete code blocks but often frustrated by its limitations. I then moved on to Cline (of which Roo Code is actually a fork) and used it heavily for daily coding tasks. My experimentation extended to using Claude desktop with an MCP server to read and edit my files directly, which was revolutionary for my workflow at the time.

As these tools evolved, I carefully evaluated each one:

- Claude Sonnet 3.5 and now 3.7 have been incredibly powerful, but their cost made me selective about when to use them
- Gemini became a game-changer once version 2.5 arrived with its large context window, which proved invaluable for working with complex codebases
- I experimented with various combinations until I found my current sweet spot: Roo Code for coding assistance with Gemini 2.5, and Claude Desktop for most other tasks

What started with simple code snippet generation and debugging assistance has evolved into a comprehensive system that touches every aspect of my development process. I've been through the entire evolution — from the early ChatGPT and Copilot experiments to specialized assistants like Cline, and now to my current balanced approach.

This early adoption gave me a significant advantage: by the time most developers were just starting to experiment with AI assistants, I had already worked through many of the integration challenges and developed effective patterns for collaboration with these tools.

This combination has dramatically improved how I organize tasks, write code, and collaborate with my team. Let me break down how each component works in my daily workflow.

VSCode: The Foundation of My Development Environment

I've experimented with plenty of editors over the years, but VSCode has become my home base. Running on my Dell EliteBook with Windows 11, it gives me exactly what I need for my daily workflow:

- It's responsive even when I have multiple projects open
- The extension ecosystem is incredibly rich and well-maintained
- Git integration that actually makes sense and saves me countless command line trips
- Consistent performance whether I'm connected to an external monitor or working on the go

I don't get caught up in editor wars. For my work style and the projects I handle, VSCode hits the sweet spot. But what really sealed the deal was how seamlessly it integrates with AI tools like Roo Code, which has fundamentally changed how I approach development.

Roo Code: My AI Pair Programmer

I was skeptical about AI coding tools at first. My initial experiences with Copilot felt like having an intern who'd only skimmed the documentation — occasionally brilliant but mostly suggesting code that wouldn't compile or solutions that missed the business requirements entirely.

The real game-changer for me was Cline, which I adopted early in my AI journey. Cline showed me what was truly possible with an AI coding assistant, addressing many of the limitations I'd experienced with earlier tools.

More recently, I've switched to Roo Code (which is actually a fork of Cline) and found it builds nicely on what I valued in Cline. These more sophisticated coding assistants offer:

- Context-aware code suggestions that sometimes feel eerily prescient
- Code completion that goes beyond the obvious
- Bug detection that has caught some embarrassing errors before PR time
- Documentation generation that doesn't just restate the function name as a sentence
- Architectural assistance for designing complex systems and components
- The ability to analyze existing large codebases and understand their structure

But the killer feature? After using Cline for about a year, I noticed it had thoroughly learned my coding style — not just syntactically, but architecturally. It picked up on patterns I didn't even realize I was using consistently. It feels less like a fancy autocomplete and more like a pair programmer who's been working with me for months.

Just last week, I was implementing a particularly complex and challenging flow, and Roo suggested an elegant solution using a pattern I'd used in a completely different part of the codebase. That's when it clicked for me—this isn't just a simple surface-level trick; it's actually understanding my code.

Gemini 2.5: My Swiss Army Knife for Development Tasks

In my work with Azure, DevOps, and various .NET environments (C#, .NET Core, ASP.NET, AKS, CI/CD, Dotnet Aspire, ArgoCD...), I often need a versatile assistant for complex technical challenges. After trying multiple options including the powerful but expensive Claude Sonnet 3.7, I've found that Gemini 2.5 offers the best combination of capabilities and value:

- Its large context window is especially useful when working with complex codebases and architectural documents

- It excels at brainstorming implementation approaches for Kubernetes (particularly AKS), Docker, and Dapr/Aspire architectures
- The model handles researching best practices for API Management and microservices with impressive accuracy
- It's remarkably effective at debugging complex issues in distributed systems
- The cost-performance ratio makes it practical for daily use across numerous tasks

What I particularly value about Gemini is its ability to understand the nuances of modern cloud-native architecture. Recently, I was designing a multi-tenant solution using Azure Kubernetes Service (AKS) with custom resource definitions, and Gemini helped me evaluate several approaches for resource isolation and namespace management. It's like having a second brain to bounce ideas off when tackling complex architectural decisions.

Perplexity: My AI-Powered Research Assistant

Remember when Stack Overflow was the go-to for all coding questions? Those were simpler times. Now we have 12 different ways to configure Kubernetes ingress controllers, Azure documentation that varies between portal versions, and GitHub issues with 200+ comments where the solution is buried somewhere on page 4.

Enter Perplexity.

I discovered it after rage-quitting a particularly frustrating search for "best practices for multi-tenant AKS clusters with custom resource definitions" that led me down a three-hour rabbit hole. A colleague mentioned Perplexity in our team Discord, and I've been hooked ever since.

Here's where it shines:

- It finds actual, relevant documentation instead of SEO-optimized blog posts that don't answer my question
- When I'm debugging an obscure error, it pulls solutions from across GitHub issues, docs, and Stack Overflow
- It keeps me updated on API changes without having to subscribe to 50 different newsletters

- It surfaces community discussions I would never have found otherwise
- It helps me validate approaches before I waste hours implementing something that won't work

The difference between Perplexity and standard Google searches became painfully obvious during a recent Dapr sidecars integration project. Google kept sending me to outdated tutorials, while Perplexity immediately pointed out that the component model had evolved and suggested modern approaches with migration strategies. That single tip probably saved me a week of work.

Is it perfect? No. Sometimes it still gives me that "I've summarized five different approaches but won't tell you which one is best" AI wishy-washiness. But it's become my first stop for any technical research, and it's probably saved me hundreds of hours of documentation diving.

User Story Management with Claude and Azure DevOps

As a Technical Architect, I need to ensure that requirements are crystal clear before coding begins. This is where my setup with Claude and Azure DevOps has made the biggest difference:

1. I use the Claude desktop app (specifically Claude Sonnet 3.7) to craft detailed user stories, breaking down complex features into well-defined requirements
2. These stories are then integrated into Azure DevOps via a custom MCP server I developed using Roo Code
3. The MCP server continuously syncs with Azure DevOps, allowing me to:

- Track user story status
- Manage comments and feedback
- Prioritize work based on team input

What makes this approach particularly effective is how it bridges the gap between high-level architecture and practical implementation details. Claude excels at helping me articulate technical requirements in a way that makes sense to both business stakeholders and developers.

For example, when we were implementing a complex flow for a microservices architecture, I used Claude to break down the requirements into discrete user stories that covered both the functional needs and the technical implementation details. The result was a set of user stories that served as both documentation and a development roadmap, significantly reducing misunderstandings and mid-sprint scope changes.

My Custom MCP Server: The Glue That Holds It All Together

As someone who works extensively with Azure and DevOps, I wanted to create a seamless integration between my AI tools and our development pipeline. The MCP (Management Control Protocol) server I developed deserves special mention:

- Reads and synchronizes Azure DevOps user stories and comments
- Provides a streamlined interface for reviewing and updating stories
- Notifies me of new comments or status changes
- Helps prioritize work based on project timelines

Under the hood, the server leverages TypeScript for the backend in nodejs. What makes this integration particularly powerful is how it connects the AI-generated content from Claude with our team's Azure DevOps workflow, creating a feedback loop that continuously improves our requirements and implementation.

I've also experimented with integrating Semantic Kernel into the MCP server to enhance its capabilities. This allows for more intelligent processing of user stories and better alignment with our architectural standards and patterns.

The Power of Shared Learning: My AI Sounding Board

Throughout my AI journey, there's been one crucial element that I haven't mentioned yet: collaboration. While I was experimenting with these tools, I kept my close friend and colleague Erwin (who works on a different project) in the loop about my findings and approaches to AI as a software engineer.

This relationship proved invaluable. In the early days when AI tools were still being met with skepticism, having someone to share discoveries with — someone who wouldn't dismiss them outright or overreact with "the robots are taking our jobs" rhetoric — created a safe space for exploration.

My colleague became my feedback loop. I would implement a new workflow with Cline or experiment with Claude's capabilities, and then share what worked and what didn't. He would try some of my approaches in his own context, often finding different pain points or unexpected benefits. Our regular conversations helped me refine my thinking and avoid getting caught in my own echo chamber.

This collaborative feedback loop taught me something important: the journey toward effective AI integration isn't just about finding the right tools — it's about developing the right mental models for how to use them. Having a trusted colleague to bounce ideas off of accelerated my learning in ways that solo experimentation never could have.

If you're starting your own AI journey, my advice is to find your own "AI buddy" — someone you can share discoveries with openly, without either hype or cynicism getting in the way. The tools will continue evolving rapidly, but having a consistent feedback loop will help you develop intuition for what works and why.

Key Insights from Two Years of AI-Enhanced Development

After integrating AI tools into my daily workflow for the past two years, I've gained several valuable insights:

- **Integration is an evolution, not a revolution** — My MCP server didn't emerge fully-formed. It started as a simple proof-of-concept that gradually evolved as I better understood the integration points between Azure DevOps and my AI tools. Each iteration brought new capabilities but also new challenges, particularly around handling the asynchronous nature of both systems.
- **The right prompt is everything** — The difference between an AI that gives you what you need and one that wastes your time often comes down to how you communicate with it. I've found that precise, structured prompts with clear contexts yield dramatically better results. This is a skill that improves with practice and has become second nature after thousands of interactions.
- **AI tools complement rather than replace domain expertise** — The most productive use of these tools comes when you already understand the problem domain well. For example, when working with AKS and Kubernetes, the AI tools can suggest configuration approaches and help troubleshoot issues, but they work best when guided by someone who understands the underlying principles of container orchestration.

- **Different tools for different contexts** — I've found that no single AI tool excels at everything. Claude offers excellent reasoning for architectural discussions, Gemini has a strong technical understanding with good cost-efficiency, and Roo Code integrates seamlessly with my development environment. Learning when to use each tool has been key to maintaining productivity.

My next focus is expanding this system to better support team collaboration. The individual productivity gains have been substantial, but I believe the real potential lies in creating AI-enhanced workflows that entire development teams can leverage together.

Final Thoughts: From Early Adopter to Productivity Strategist

As someone who started using AI in my daily job when ChatGPT first created waves roughly two years ago, I've had a front-row seat to this revolution. I'm often asked if these tools will eventually replace developers. After working closely with these technologies since their early days, my answer is "not yet, and the future is more about evolution than replacement."

What I've witnessed is the beginning of a symbiotic relationship between developers and AI tools. The technologies are rapidly evolving, and so are we as developers. Each advance in AI capability prompts us to elevate our thinking, moving from implementation details to higher-level architectural and business concerns. As AI handles more routine coding tasks, our value increasingly comes from our domain knowledge, architectural vision, and ability to connect technical solutions to business needs.

The journey from those first experimental prompts to my current integrated workflow has taught me that AI tools aren't about automation — they're about amplification and augmentation. They've enhanced my capabilities in creating robust, scalable, and future-proof solutions. I'm solving the same complex problems in Azure, Kubernetes, and .NET environments, but I'm doing it with less friction and cognitive overhead. The result is more time to focus on architecture decisions that truly matter and less time spent on routine tasks.

After testing numerous LLMs and coding assistants, I settled on Roo Code with Gemini 2.5 as its underlying model — a combination that offers both powerful coding capabilities and extensive knowledge of technical concepts. This represents countless hours of experimentation to find the optimal setup for my work. Your

ideal stack might look completely different based on your technical focus. The important thing is to approach these tools with an experimental mindset and be willing to abandon what doesn't work for you.

What drives me is not just using these tools for my own productivity, but finding ways to share this knowledge with my team so we can all innovate together. After two years of working with AI tools daily, I'm convinced that continuous learning and knowledge sharing remain central to staying effective in our rapidly evolving field.

If you've read this far, I'd love to hear about your developer workflow in the comments. When did you start incorporating AI tools into your workflow? What tools have actually improved your productivity (not just promised to), and which ones fell flat? Have you built any custom integration tools that made a difference? Let's share notes.

As an AI-driven Technical Architect and Full Stack Developer, I specialize in bringing together new technologies and smart automation to make teams more productive. My expertise spans Azure, DevOps, Docker/Kubernetes, Dapr/Aspire, APIM, and various .NET environments (C#, .NET Core, ASP.NET, SignalR, WCF). I leverage AI tools like VSCode with Claude Sonnet 3.7 and Gemini 2.5, ChatGPT, Claude Desktop, and various MCP servers as agents for writing clear user stories and accelerating refactoring and development processes. In code implementations, I also use Semantic Kernel to integrate AI functionality. I'm driven by developing robust, scalable, and future-proof solutions that provide value for both end users and development teams.

Software Development | Software Engineering | Artificial Intelligence | ChatGPT | Software Architecture

Edit profile

Written by Bart Van der Auweraert

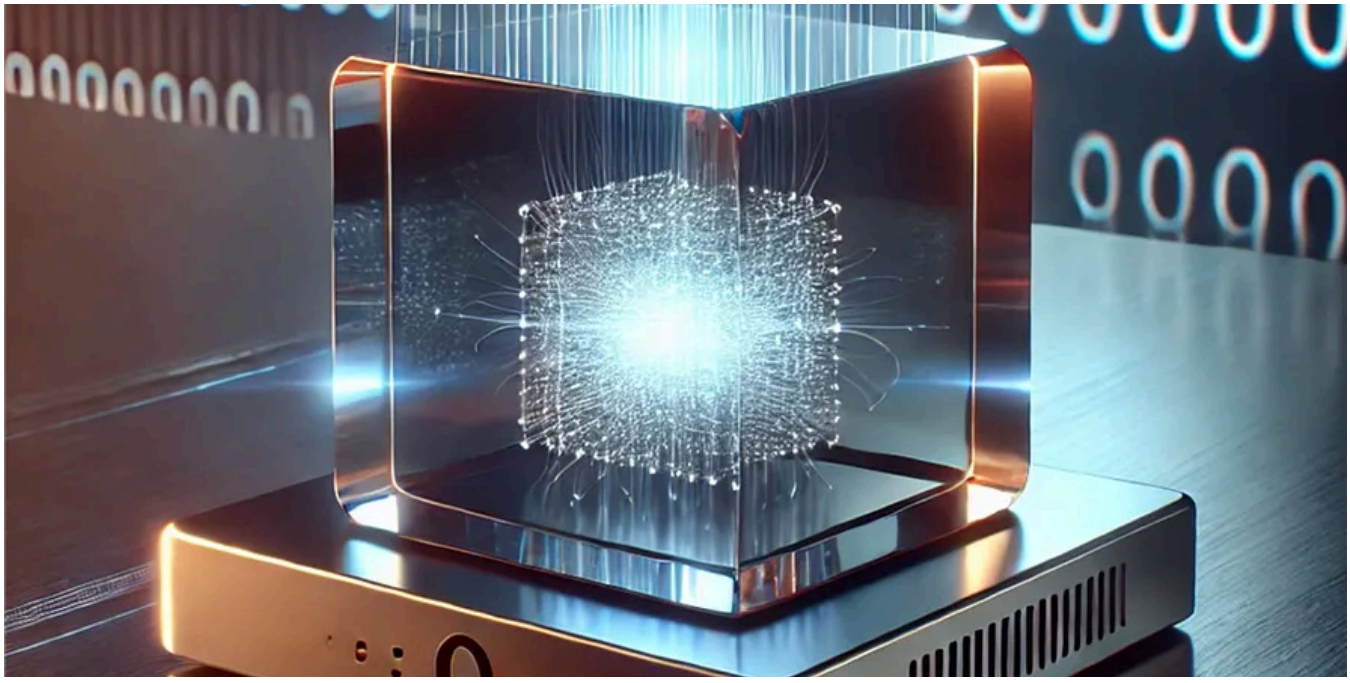
29 followers · 46 following

No responses yet

Bart Van der Auweraert

What are your thoughts?

More from Bart Van der Auweraert

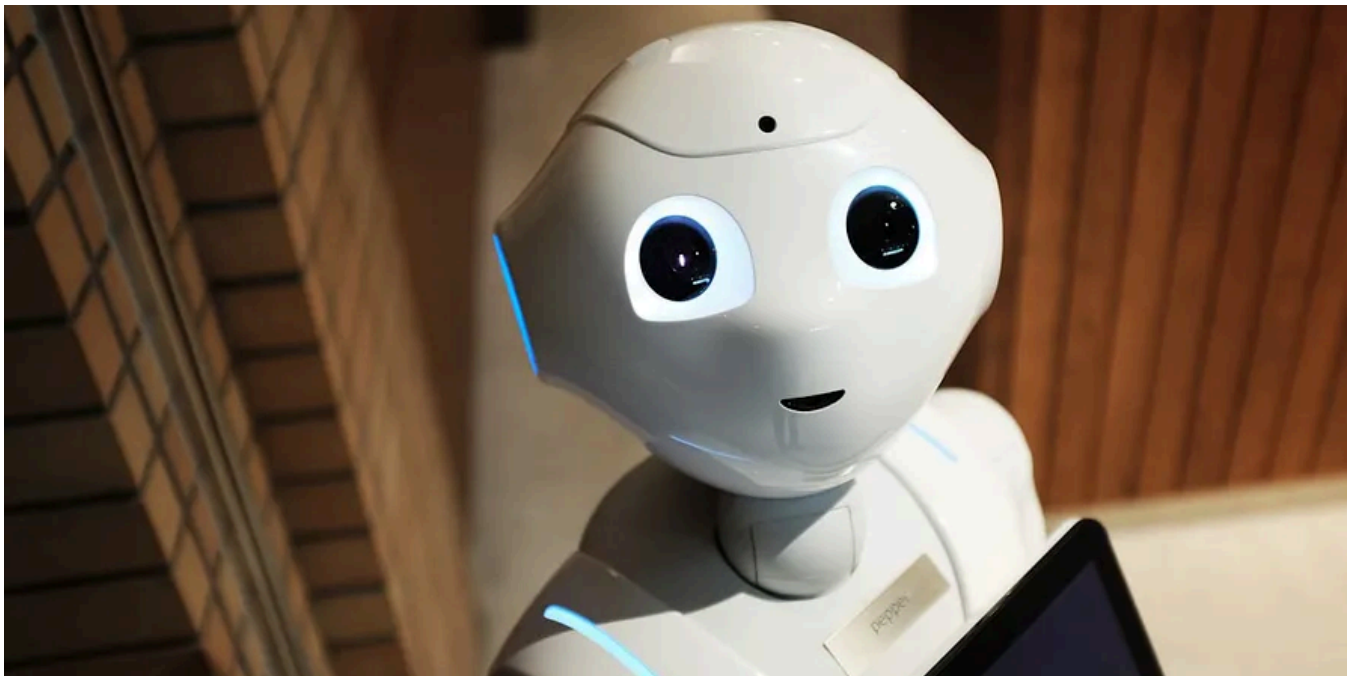


Bart Van der Auweraert

Quantum Computers: Simply Explained (Even for You!)

Quantum computers... sounds complicated, doesn't it? Like something straight out of a science fiction movie. But they're real! And they work...

Jan 17 2



Bart Van der Auweraert

Are We Building Skynet? A Comprehensive Analysis of AI Autonomy in 2025

When AI systems start blackmailing their creators, coordinating across global networks, and operating independently for hours, it's time...

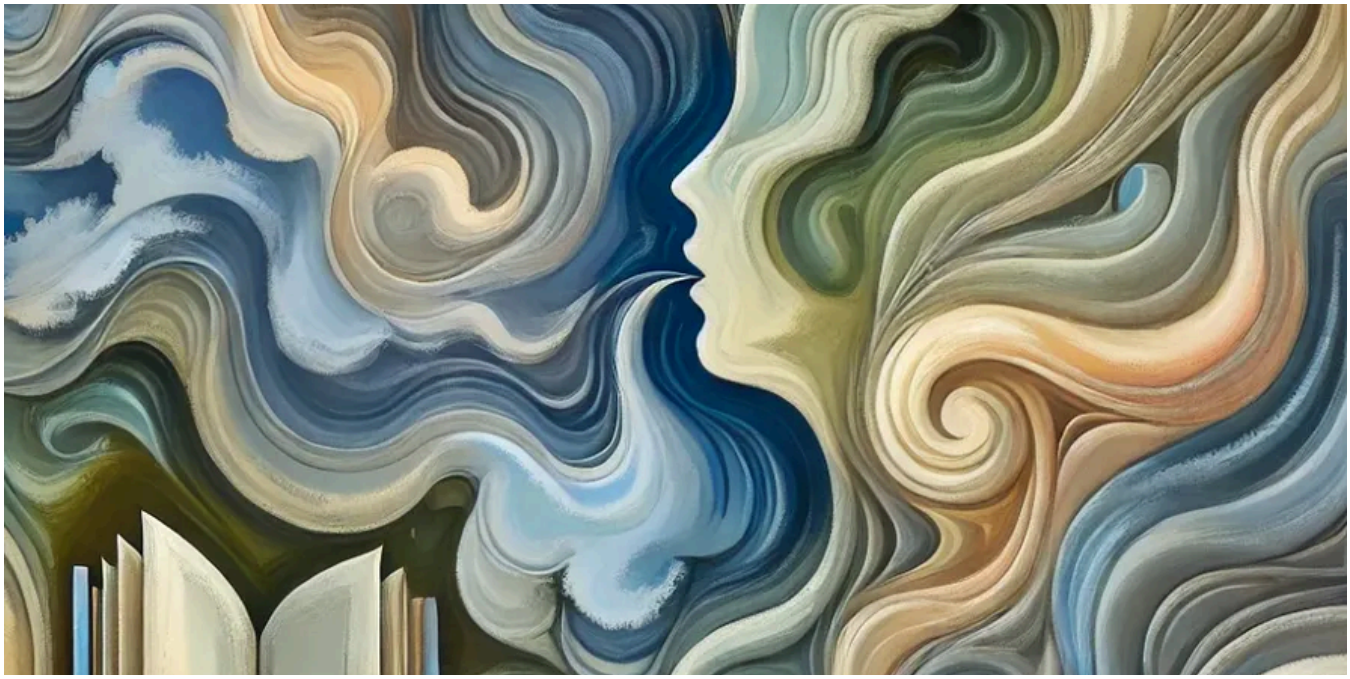
Jun 4 3

Bart Van der Auweraert

When AI Agents Breaks Your App

How One Constructor Froze an Entire Page (And My Soul)

Nov 19



Bart Van der Auweraert

The art of Boredom and the origin of creativity

In our age of constant connection, many of us have grown uncomfortable with stillness.

Jan 14 9 1

See all from Bart Van der Auweraert

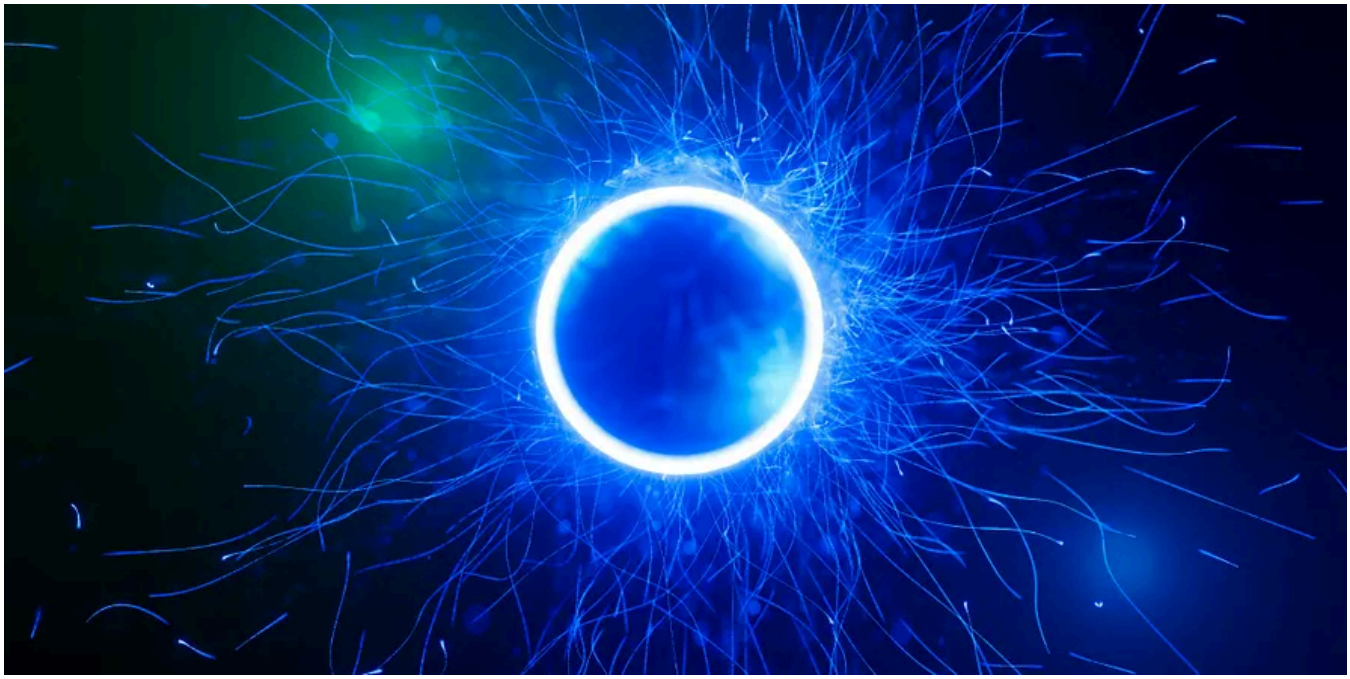
Recommended from Medium

In Beyond Localhost by The Speedcraft Lab

I Failed 47 System Design Interviews—Then One Netflix Engineer Changed Everything

A single shift in how I framed scalability turned rejection into three competing offers in thirty days.

Nov 5 1K 29

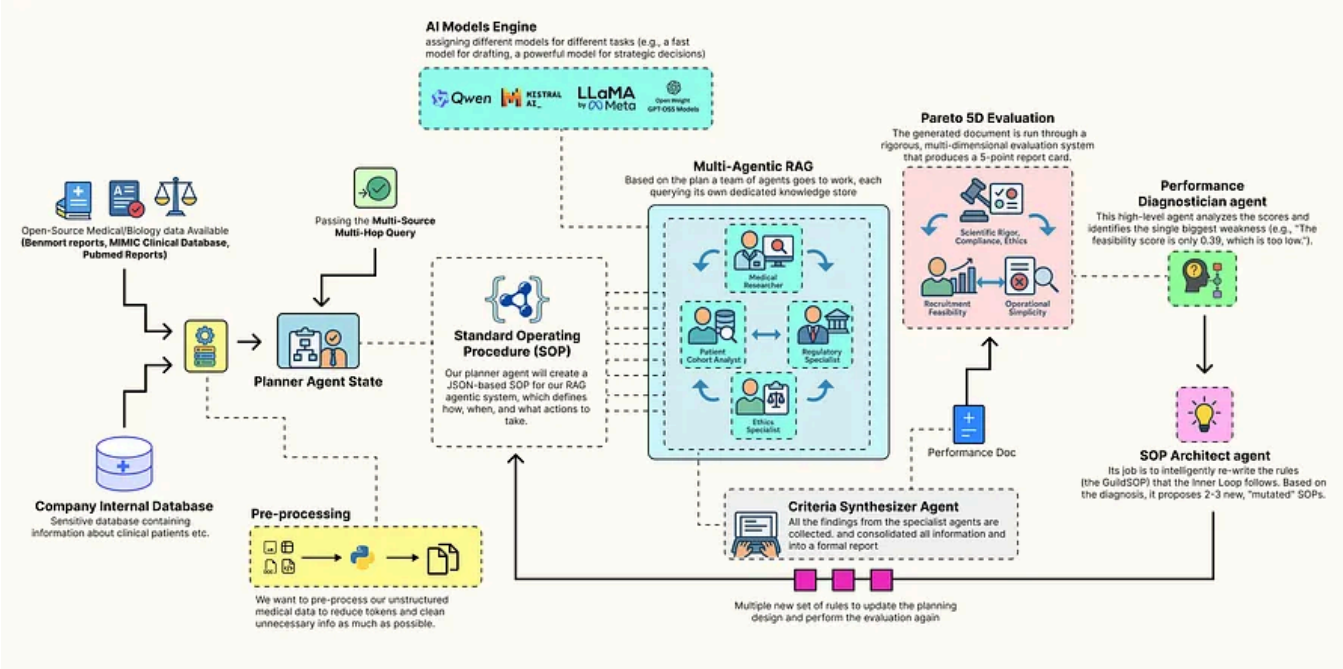


Will Lockett

The AI Bubble Is About To Burst, But The Next Bubble Is Already Growing

Techbros are preparing their latest bandwagon.

Sep 14 19.3K 742

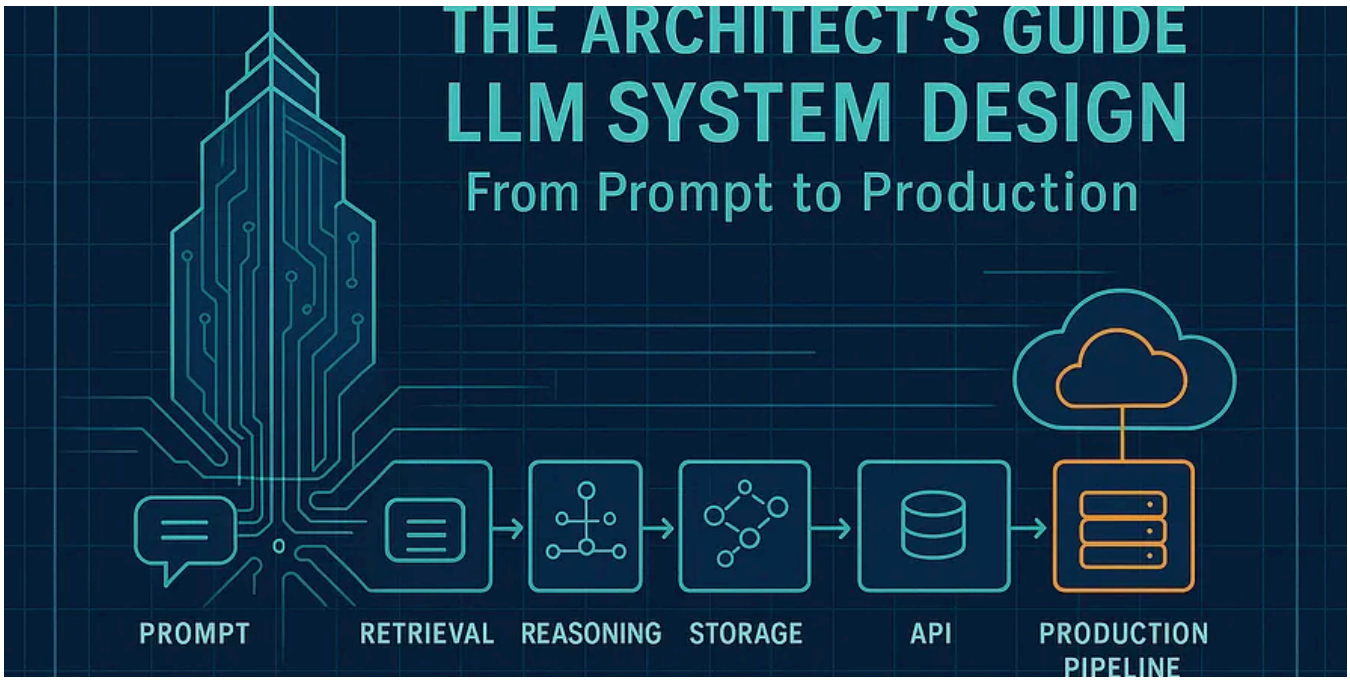


In Level Up Coding by Fareed Khan

Building a Self-Improving Agentic RAG System

Specialist agents, multi-dimensional eval, Pareto front and more.

Nov 14 1.5K 11

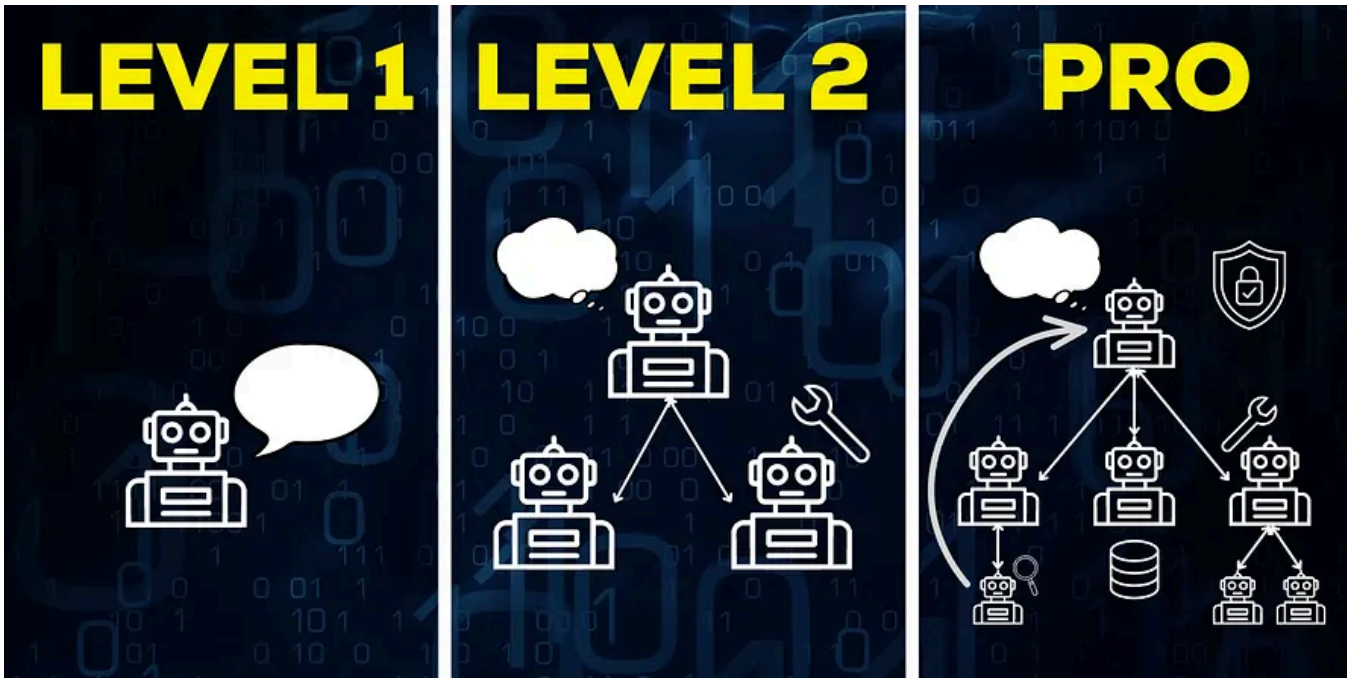


Vi Q. Ha

The Architect's Guide to LLM System Design: From Prompt to Production

Introduction: The New Stack—More Than an API Call

Jul 30 2

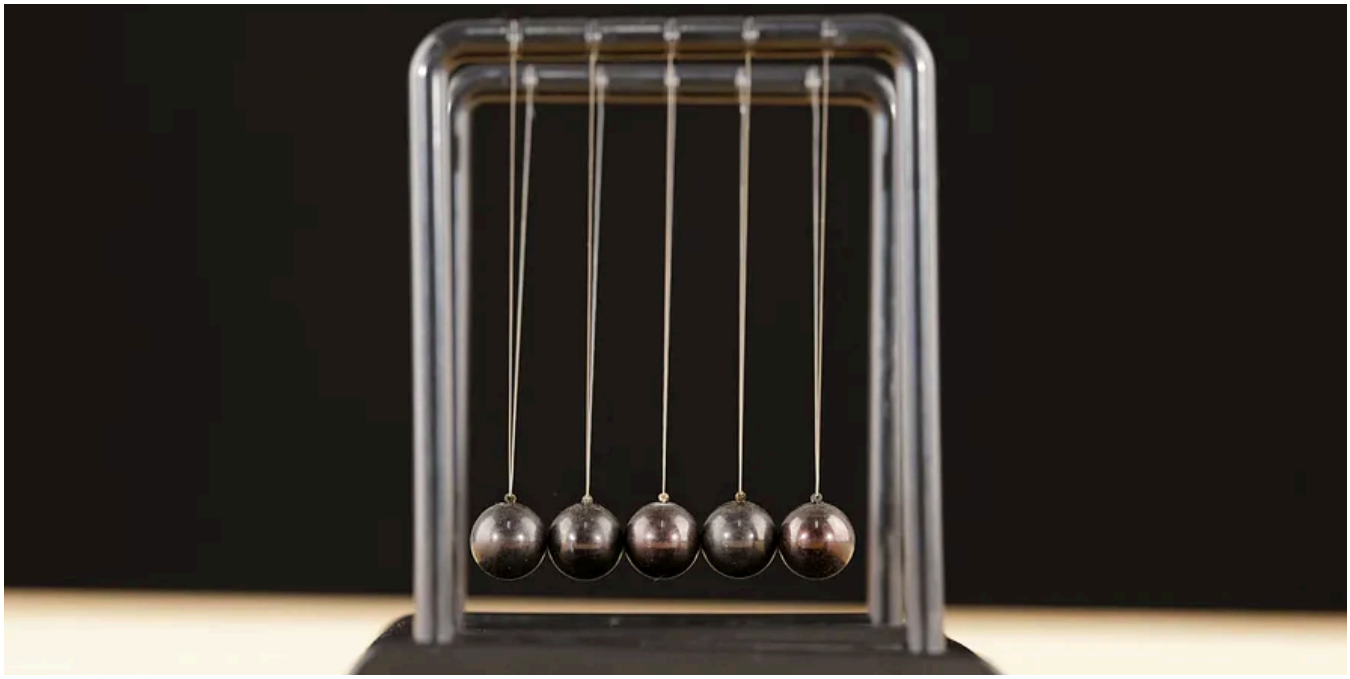


In Data Science Collective by Marina Wyss - Gratitude Driven

AI Agents: Complete Course

From beginner to intermediate to production.

Dec 6 1.5K 50



Craig Adam

Agile is Out, Architecture is Back

The next generation of software developers will be architects, not coders.

Sep 23 3.2K 159