

Your subscription payment failed. [Update payment method](#)

# From vibe coding to agentic engineering (abridged)

How AI is reshaping software engineering in the Singapore government, and the real-world stories that prove it's already happening

12 min read · 3 hours ago



Sau Sheong



Following



Listen



Share



More

*[ Thank you readers for taking time to read my previous articles [From vibe coding to agentic engineering](#) as well as [Agentic engineering in the wild](#). A lot of time, is the pointed feedback I get. I have heard you loud and clear. They are lengthy pieces and you just want to get to the points. Here's an abridged version of the 2 articles, combined. ]*



Generated by Nano Banana 2 (yes even the images are combined)

I've been having a lot of conversations lately. With engineers, with leaders in the industry, with people building tools that didn't exist six months ago. And one thing has become very clear to me. AI isn't just changing software engineering. It's reshaping the whole thing, from how we write code to how we think about what code even is.

This isn't a future thing. It's happening right now, and it's moving faster than most of us can keep up with.

At GovTech Singapore, we've been thinking hard about this. We put together an AI Strategy for Software Engineering earlier this year, and I've been engaging with technologists locally and around the world. What I've seen is a field in profound transition. Tools are evolving faster than organisations can adapt. The role of the engineer is being redefined. Old certainties about how software is built, reviewed and governed? They don't hold anymore.

I've organised this article in two parts. The first covers the broader industry landscape and how we're responding to it. The second shares real stories from GovTech teams who are already living this shift.

## The five levels of AI engineering

One of the more useful frameworks I've come across is Dan Shapiro's five-level model for understanding where AI sits in software development.

- *Level 1* is spicy autocomplete, AI suggesting code completions within the developer's context.
- *Level 2* is AI coding assistants like Claude Code or Cursor, executing multi-step tasks across files and tools.
- *Level 3* is autonomous development agents that independently take tickets from backlog to deployment.
- *Level 4* is collaborative agent networks, multiple specialised agents working together.
- *Level 5* is the software factory, where organisations describe desired outcomes and entire systems emerge.

Different parts of an organisation can and probably should operate at different levels simultaneously. Low-risk systems might accelerate toward Level 3 or 4, while critical national infrastructure should stay at Level 2 with stronger governance. Each level jump requires fundamentally different governance, risk management and quality assurance. I've seen organisations try to use Level 1 governance frameworks with Level 3 capabilities. It doesn't end well.

## Three shifts that go beyond engineering

Before diving into the detail, three changes stood out from every conversation I had.

First, the bottleneck is inverting. Engineering capacity is no longer the constraint. Decision speed is. Governance, procurement and compliance designed for human-speed delivery will become the primary brake on value.

Second, code may not be the durable artefact anymore. Specifications, domain models and decision history may matter more than source code, which becomes regenerable and ephemeral.

Third, team structures are about to shrink dramatically. If one person with AI can reach proof-of-concept before any engineering resource is engaged, everything we

assume about team composition, vendor engagement and project economics needs re-examination.

These three shifts recur throughout what follows. And as you'll see in the real-world stories later, they're not theoretical.

## **The bottleneck has shifted**

For decades, the common complaint has been that business stakeholders outpace the engineers who build for them. Backlogs grow. Priorities shift. Engineering becomes the bottleneck everyone plans around. That dynamic is about to reverse.

When agents can generate working code in hours rather than weeks, engineering capacity stops being the scarce resource. What becomes scarce is the ability to decide what should be built, to review whether it's right, and to absorb the consequences of shipping it. An agent can produce a working prototype overnight, but the approval to deploy it might take months. The organisations that will move fastest aren't those with the best engineering teams. They're those with the shortest distance between a decision and an action.

## **What is the artefact, if not code?**

If AI can regenerate code on demand, is code even the right thing to preserve? I heard arguments for specifications, domain models and decision history as more durable artefacts. The story of why decisions were made may matter more than the final implementation. Code, in this view, becomes ephemeral, regenerable, and potentially not even worth reading.

This sits uneasily with me. In government, we maintain systems for decades. We need to explain why things work the way they do, sometimes years after the people who built them have moved on. If code becomes disposable, what exactly are we auditing?

There is a counter-argument worth taking seriously. If AI can generate specifications from code just as easily as it can generate code from specifications, why not keep code as the primary artefact? Code is unambiguous. It compiles, it runs, it can be tested. Specifications are prone to drift and incompleteness. In practice, we may need both, maintained in tandem, with AI keeping them in sync.

## **Trust, care, and what's lost in abstraction**



As engineers increasingly rely on AI to navigate codebases they can't fully understand, questions of trust become acute. My own principle is that fully trusting an LLM's answer is folly, but using LLMs to navigate toward an answer is wise. The value is in the navigation, not the destination it hands you.

There's a deeper concern I keep coming back to. Part of the value of pair programming comes from explaining things to your pair. That act solidifies your own understanding. With AI as the pair, that forcing function disappears. Active knowledge decays without practice.

I noticed this in myself recently. I was using Claude Code to work through a codebase I hadn't touched in months. The agent navigated it faster than I could have, found the right files, made the changes, ran the tests. Impressive. But afterwards I realised I had no real understanding of what had changed or why it worked. If something broke a week later, I'd have to start from scratch. The speed is real, but so is the erosion of understanding. We need to design deliberately for the knowledge we want to retain.

## **Platforms, agents, and organisational readiness**

Platforms matter because they make the right thing the easy thing. If you make compliance a manual checklist, teams will skip it under deadline pressure. If you embed compliance into the deployment pipeline, teams comply by default. In an AI-augmented world, the platform is the primary mechanism through which an organisation exerts quality control, maintains security posture, and ensures governance at scale.

How we treat agents also matters. Should we treat them as non-deterministic technical systems, or as team members requiring onboarding and mentoring? Most organisations will land somewhere between both. The "technical system" framing leads you to invest in guardrails and monitoring. The "team member" framing leads you to invest in context engineering and progressive trust. Doing both simultaneously is genuinely hard to design for.

And AI amplifies existing conditions. It doesn't create them. Strong teams become faster. Dysfunctional teams become more chaotic. A team with clear coding standards and well-organised repositories can hand an AI agent meaningful context and get meaningful output. A team with inconsistent practices and undocumented

tribal knowledge gets output that's confidently wrong and wrong in ways that are hard to detect.

## **The human role is being redefined**

The shift is from traditional software engineering to supervisory engineering. Directing, evaluating and correcting AI outputs. Writing specifications and acceptance criteria. Exercising judgment about what to build. This requires new skills that many current engineers don't have, and there are no clear pathways to acquire them yet.

Counter to the prevailing narrative about AI eliminating entry-level roles, I'm quite positive about junior engineers. AI-native juniors carry potential advantages that are easy to underestimate. They need to prove themselves, which creates motivation to master new tools. They're willing to make mistakes, which is essential in stochastic environments. And they lack organisational baggage, no preconceptions about the "right way" to build software.

Conversely, the broader sentiment around senior engineers has been more negative than I expected. The resistance doesn't manifest as outright refusal but as using new tools to do familiar things. Many years ago, I spent a long time persuading a C programmer to write in Java. When I finally managed to convince him, he simply continued writing C, but in Java syntax. He took none of the advantages of Java, only the disadvantages. This is the *deja vu* I'm feeling now.

The value of agentic coding tools doesn't come from doing the same things slightly faster. It comes from doing fundamentally different things. Working at a higher level of abstraction, delegating implementation to agents, spending more time on specification and review and less on typing. That requires a willingness to let go of direct control that many senior engineers find deeply uncomfortable. Their expertise, their identity, their career trajectory. All of it has been built on the ability to implement. Asking them to stop implementing and start supervising feels like asking them to give up the thing that makes them valuable.

## **Vendor and SaaS disruption**

Traditional engagement models for outsourced vendors and SaaS products are under pressure. If agents can generate code at scale, the labour intensity drops. If AI tools make individual engineers dramatically more productive, you need fewer of them. The vendor's value proposition of "we'll bring you twenty engineers" loses its force when your internal team of five, equipped with AI, can match that output.

SaaS faces a similar challenge. Most organisations use only a fraction of what any SaaS product offers. If AI can build that fraction as a custom tool tailored to their specific workflow, the value proposition of paying for the rest becomes hard to defend. Nobody should be vibe coding their own identity management system. But internal dashboards, workflow tools, lightweight project tracking? These are all candidates for AI-built alternatives.

## How we're responding

Our strategy is being built across several workstreams. We're deploying AI coding tools across the organisation, focusing on *Copilot* and *Claude Code* as standard offerings.

We've created the *Agent Prime Directives*, a centrally managed tool providing shared context, skills and tools that make AI coding assistants effective and consistent across teams. Some engineers have already stopped writing code by hand entirely and are using only Claude Code with the Agent Prime Directives. That's not a mandate. It's organic adoption driven by genuine productivity gains.

We've built *Prelude*, an in-house AI code review tool. We've developed *Graphiqode* for legacy modernisation, analysing codebases to build visual dependency graphs and extract business rules. And we're building the enabling infrastructure through *Backstage* for service cataloguing, *GovPaas* for secure hosting, and *Astrolabe* for engineering metrics.

What ties it all together is a coherent idea-to-product pipeline. A non-technical officer prototypes using a vibe coding tool. The application gets registered and tracked. Prelude runs automated security scans. When the prototype shows enough promise, engineers come in with agentic coding assistants. The entire journey is tracked, auditable and on approved platforms. No step requires the builder to go outside the guardrails.

## Agentic engineering in the wild

Those were the arguments. Now for the proof.

I sat down with several people across my teams at GovTech Singapore to understand what AI-assisted coding actually looks like in practice. The numbers were so dramatic I had to double-check them. But these aren't projections. These are things that have already happened, with real users and real production systems.

## The smallest possible team

“I have really limited Android development experience. Like, pretty limited,” the software engineer began. “But within 2 weeks I personally delivered 2 fully functional projects as the sole developer on both.”

The first was an Android app with an on-device LLM that scans messages for scams in real time. Team of three. The second was an autonomous AI agent for detecting and disrupting scam enablers across online platforms. Team of two.

“If you do the maths, the old way would be 16 to 36 person-weeks. What I did was 2 person-weeks. That’s an 8 to 12x reduction.”

But what surprised me more was what came next. PMs on his team now code simple features and verify changes themselves. Data scientists troubleshoot deep model issues with Claude. One PM is acting as a data scientist on another project entirely.

“The floor has risen dramatically,” he said. “People who couldn’t build things before can now build things.”

The fences between roles are lower, not gone. The cost of crossing into adjacent territory has dropped dramatically, not that the territory no longer exists.

### **The data scientists who built a platform**

“No software engineering background at all,” the data scientist said. “And I built what is now a core part of our go-to-market strategy for vision and multimodal AI.”

Engineers told him it would take a few sprints for a working prototype. Too slow. He built it himself with Claude Code. The bottleneck inversion in action. The constraint wasn’t the building. It was the waiting.

Two data scientists, no software engineers. Over 30 full-stack showcases, 12 capability playgrounds, 150+ users across 20+ agencies. What used to take 5 to 7 days per showcase now takes 1 to 2, thanks to a monorepo with reusable patterns that Claude Code follows. That’s roughly a 30x multiplier per showcase.

His colleague had zero web development experience. Zero. She now independently delivers full-stack showcases. The monorepo plus Claude Code turned their codebase into a curriculum.

The ripple effects went further. A PM vibecoded features over the December holidays, now in client engagements. A UX designer builds Figma plugins. An

engineering manager polishes slides. All with Claude Code, all without formal sanction.

And this is where I felt a knot in my stomach. That PM's holiday project is now in client engagements. Did it get a security review? Does anyone besides him know how it works? The capability is extraordinary, but without guardrails it creates risk at exactly the speed it creates value.

### **The engineer who became an entire team**

"I built a production-grade AI system actively used across over 150 projects, including Singpass, GovWallet, and TradeNet," the software engineer said. He paused. "By myself."

This is Prelude, the AI code review agent I mentioned earlier. When a developer opens a merge request, Prelude spins up an isolated sandbox with Claude Code inside, analyses the code against government classification policies, generates structured review comments with confidence scoring, and posts them on the merge request.

Six technical domains. Backend, AI/ML, frontend, data engineering, DevOps, security. One person. A conservative estimate without Claude Code would be 3 to 5 engineers over 6 to 12 months.

In January 2026 alone he hit 256 commits and 94 merge requests, up from a baseline of 60 to 70. While concurrently contributing to 6+ other projects.

"It's not magic," he said. "You still need to make architectural decisions, you still need to review what comes out. But the execution speed is in a completely different league."

He didn't say the AI did the thinking. He said it did the execution. Claude Code didn't replace his expertise. It amplified it. This is supervisory engineering in action.

### **What the stories tell us**

The numbers are frankly hard to believe. 8 to 12x. 30x. A data scientist with no web experience shipping full-stack apps. A single engineer maintaining a production system that would normally need a team of five. If someone had told me this five years ago, I would have smiled politely and changed the subject.



But nearly every theme from the strategy showed up in these conversations. The bottleneck inversion is real. The solo-to-scale model works, in government no less. Role boundaries are shifting faster than I expected. Platforms like Agent Prime Directives are the enabling layer. And every person I spoke to was clear that the AI didn't do the thinking, it did the execution. The human value was in the decisions.

There are hard questions embedded in the good news. When a PM builds a tool that's now in client engagements, who reviewed it? When one engineer is responsible for a system spanning six domains, what happens when that person is unavailable? These governance questions are more urgent, not less, because these tools aren't spreading at the pace of a managed rollout. They're spreading at the pace of people discovering they can build things.

Nobody I spoke to said the AI does the thinking for them. The software engineer still needed to understand on-device LLMs. The data scientist still made architectural decisions. Claude Code didn't replace the expertise. It amplified it.

The shift from vibe coding to agentic engineering isn't coming. It's already here. The question is whether we can build the governance, the platforms, and the organisational structures fast enough to match the pace of people who aren't waiting for permission.

That's the work ahead.

AI

Software Development

Claude Code

Vibe Coding

Agentic Ai



Following ▾

**Written by Sau Sheong** 

3.4K followers · 251 following

I write, code.



## No responses yet



Son Le Thanh (Son)

What are your thoughts?

## More from Sau Sheong



Sau Sheong

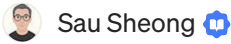
## From vibe coding to agentic engineering

How AI is reshaping software engineering in the Singapore government, and what comes next

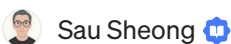
Feb 28 👍 376 💬 7







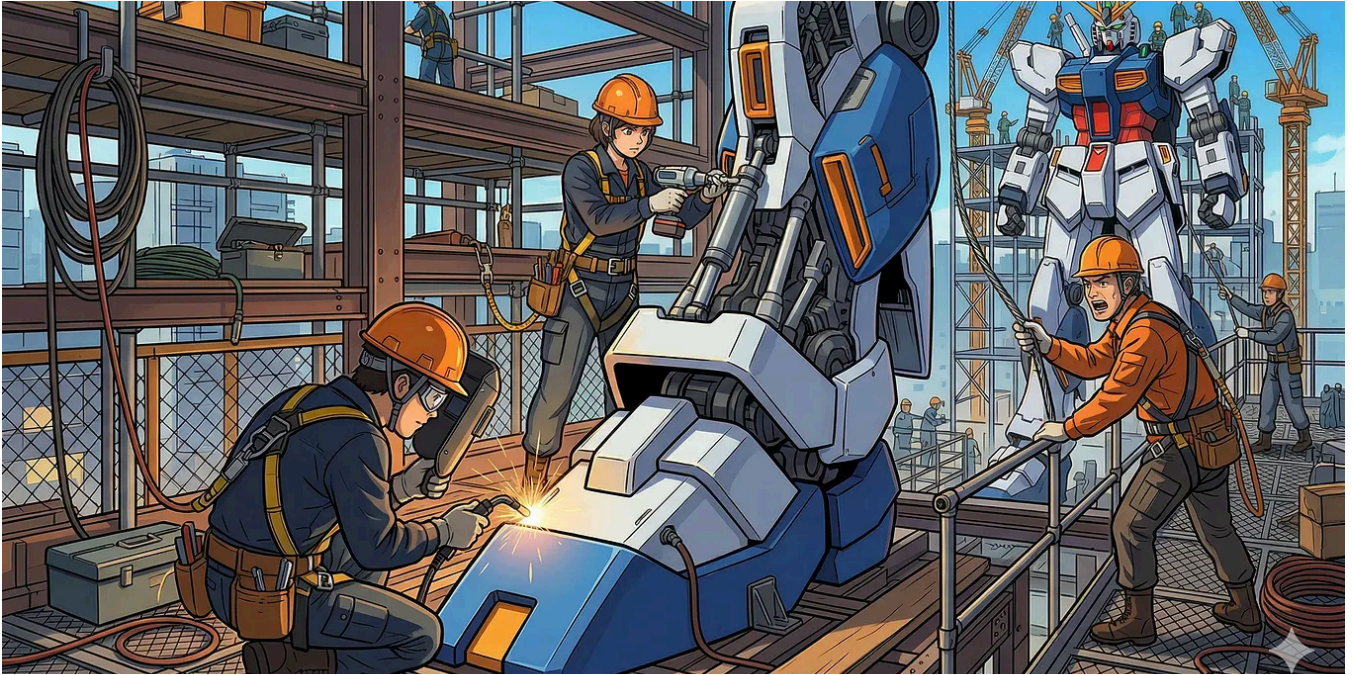
Real-world stories from early-adopting Singapore government teams experimenting with and delivering with AI



 

## How AI, vendor concentration, and geopolitical risk are changing organisational technology decisions



Mar 28  230  4



 Sau Sheong 

## The Context and the Harness

Or, Why the Model Is Just the Beginning

6d ago  65  1



See all from Sau Sheong

## Recommended from Medium





 Suleiman Tawil

## Qwen Just Quietly Became the Most Dangerous Open-Source AI Model

The most-downloaded AI model family on Earth was built by a small team with fewer resources than its competitors. Then Alibaba restructured...

🌟 Mar 31 🗣️ 1.2K 💬 34



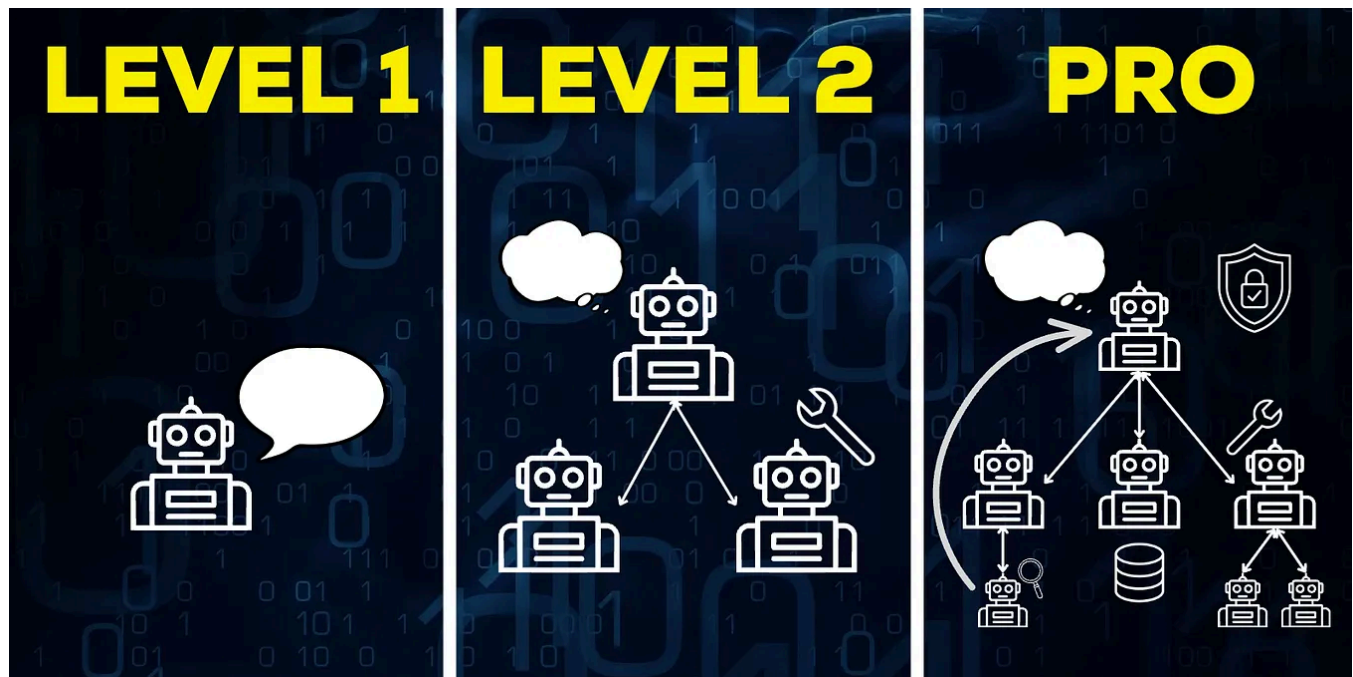
 unicodeveloper

## 10 Must-Have Skills for Claude (and Any Coding Agent) in 2026

The definitive guide to agent skills that change how Claude Code, Cursor, Gemini CLI, and other AI coding assistants perform in production.



Mar 9 1.2K 21



**DSC** In Data Science Collective by Marina Wyss

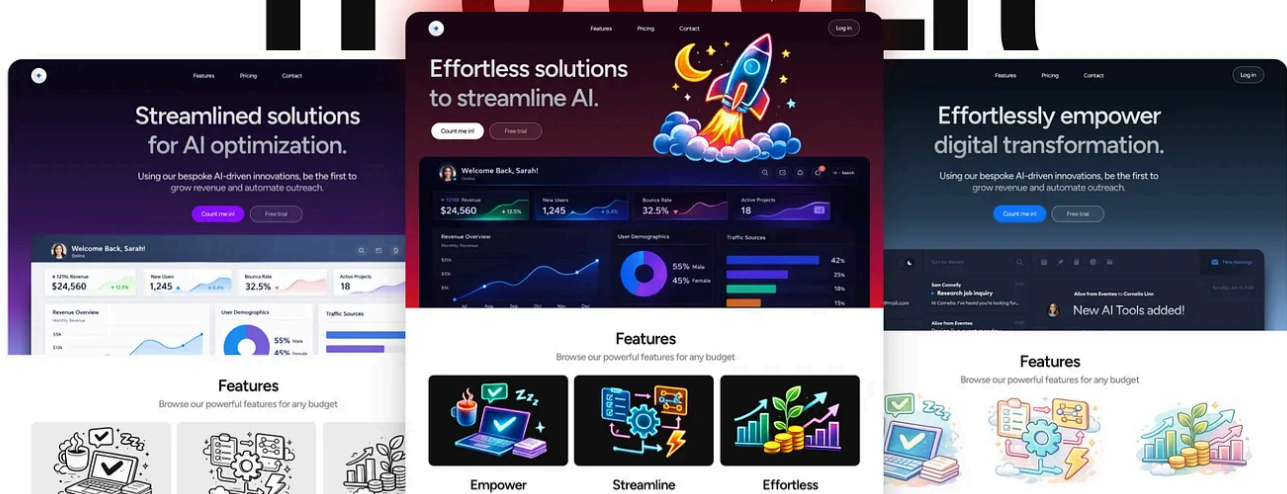
## AI Agents: Complete Course

From beginner to intermediate to production.

★ Dec 6, 2025 5.5K 220



# IT'S OVER



Michal Malewicz

**Vibe Coding is OVER.**

Here's What Comes Next.

★ Mar 25 🖱 4.6K 💬 159

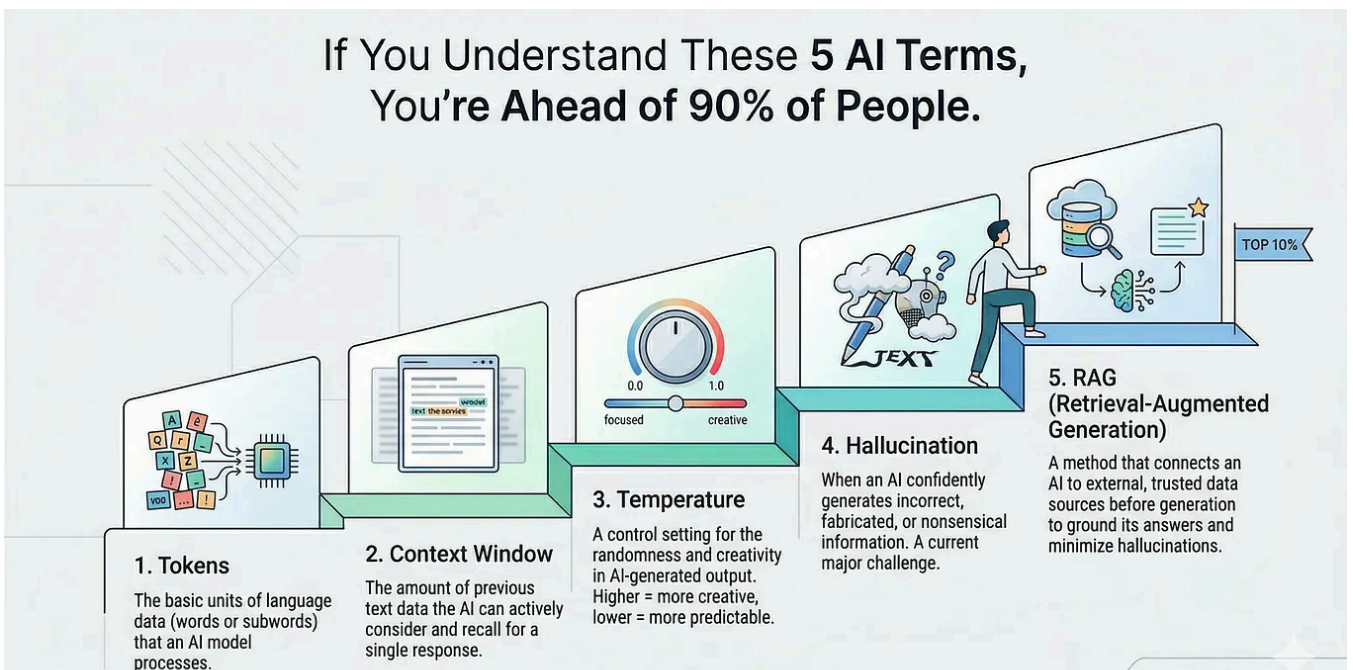


Joe Njenga

## Claude Code Ultraplan Launched: I Just Tested It (And It's Better Than It Looks)

Anthropic has added Claude Code ultraplan, and I was quick to test it. You might like it or hate it for one reason — I'll talk about that...

★ 4d ago 🖱 469 💬 19





In Towards AI by Shreyas Naphad

## If You Understand These 5 AI Terms, You're Ahead of 90% of People

Master the core ideas behind AI without getting lost



Mar 29



6.4K



128



See more recommendations