nginity · Follow publication

Member-only story

Claude AI and Claude Code Skills: Teaching AI to Think Like Your Best team mate

Stop Copying Prompts. Start Building Intelligence. From Prompt Fatigue to Persistent Intelligence: Why Agent Skills Are the Architecture Pattern You're Missing.

14 min read · Oct 19, 2025

Reza Rezvani

Learn how Claude Agent Skills solve the context retention problem every engineering team faces. Build reusable AI intelligence through progressive disclosure and hybrid execution patterns.

Anthropic's Claude Skills will help any team to have higher outcomes

FINAL ARTICLE — SEO OPTIMIZED & PUBLICATION READY

Our senior frontend engineer is explaining our component architecture to Claude. Third time this week.

Same TypeScript patterns we've refined over two years. Same accessibility requirements we learned after that compliance audit nobody wants to remember. Same design system constraints that keep our products from looking like a frankenstack.

She gets decent output. Tweaks the prompt. Tries again. Eventually lands on something workable.

Different engineer, identical conversation. Wednesday, it's my turn — explaining our API documentation standards for what genuinely feels like the hundredth time this quarter.

That's when the realization hit like cold water: We weren't scaling our AI adoption. We were industrializing repetitive conversations.

Our productivity gains? Being devoured by context-setting overhead. Every conversation started from zero. Every fragment of hard-won tribal knowledge had to

be re-explained. Every standard re-taught. Every pattern re-discovered.

. . .

Your Current Solutions Are Sophisticated Workarounds (And Deep Down, You Know It)

Let me paint a picture. You've probably tried solving this context problem, and nothing feels quite right.

The master prompt document approach. I lived this for six months. Started well-intentioned. Within three weeks, we had five competing versions scattered across Slack threads. Half were outdated. Nobody knew which version was canonical. Someone would update their copy, forget to share it. The whole system held together with good intentions and increasingly desperate "friendly reminder" Slack messages.

The RAG system evaluation. We almost committed serious engineering time here. Vector databases, embedding models, chunking strategies, retrieval scoring. Real infrastructure overhead. Pinecone or Weaviate. Ongoing maintenance. And even with all that sophistication, you're still fundamentally constrained by context window limits. Plus retrieval quality varies wildly based on semantic similarity. It's impressive technology solving the problem indirectly.

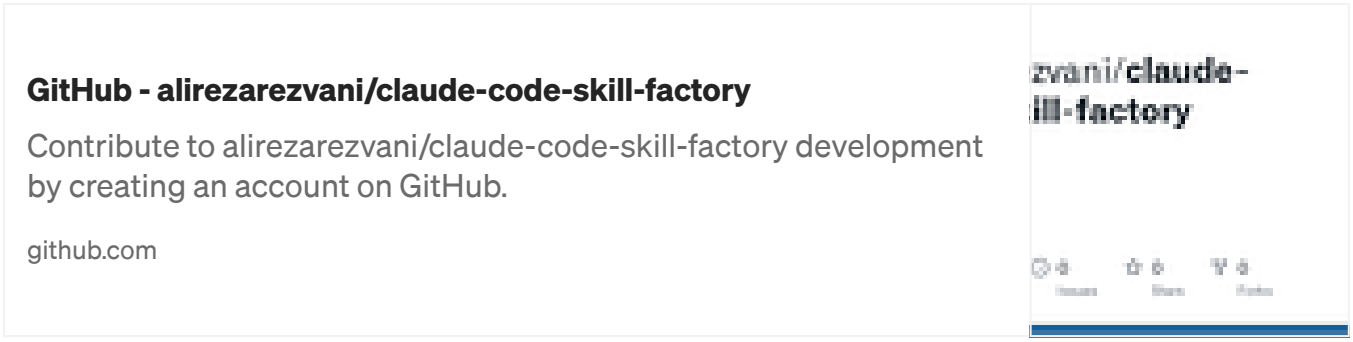
Fine-tuning? Most teams lack the ML expertise or budget. And even if you have both, fine-tuning optimizes for style and format — not for encoding procedural knowledge that evolves every sprint.

Here's what makes Agent Skills fundamentally different from everything else: They're not trying to outsmart the context problem with clever AI techniques. They're solving it through proper software architecture — modularity, version control, lazy loading, hybrid execution.

It's less exciting than discussing embeddings and vector spaces. But it works. Actually works. In production. Under real conditions.

Download and try the enhanced SKILLS FACTORY PROMPT AND REPO for creating any Agent skill at scale:

*** Happy if you rate this repo with stars on Github, if you find it useful :) Share it with friends or colleagues! ****



. . .

The Progressive Disclosure Architecture That Changes Everything

Agent Skills operate on a principle that feels almost too obvious to be revolutionary: Claude doesn't need to know everything about your organization upfront.

It needs to know *what expertise exists* and *when to activate it.*

Think about effective onboarding. You don't print your entire engineering wiki and drop it on someone's desk their first day. You provide a map: "Security standards live here. API patterns are documented there. Testing guidelines over here." They explore as needed, loading context just-in-time.

Agent Skills work identically. A skill is simply a folder containing a `SKILL.md` file that begins with YAML frontmatter—just a name and description:

```
---
name: api-docs-expert
description: Generates TypeScript API docs following our security standards, JS
---
```

This metadata loads into Claude's system prompt at startup, creating a lightweight catalog. When you ask about documenting an endpoint, Claude scans its internal catalog, recognizes `api-docs-expert` is relevant, and only then loads the detailed instructions.

This isn't prompt engineering cleverness. It's progressive disclosure — the same foundational pattern we use in UI design, API design, and every system architected to scale. Load what's needed, exactly when it's needed, nothing more.

The implications cascade beautifully from this simple principle. You can maintain dozens of skills without overwhelming Claude's context window. Each conversation loads only relevant expertise. Adding new skills doesn't degrade performance on existing ones. The system scales horizontally instead of fighting vertical context limits.

And here's what makes it genuinely different: it's just files and folders. No special infrastructure. No API dependencies. No vendor lock-in. Pure architecture solving an architectural problem.

. . .

The Code Review Skill That Dissolved My Skepticism

I approached this skeptically. It looked like glorified documentation wrapped in unnecessary ceremony.

Then I spent a weekend building a code review skill for our TypeScript codebase. The skepticism evaporated somewhere between Saturday afternoon and Sunday morning.

Here's what I encoded: Our actual review checklist. Not generic "best practices" scraped from blog posts — the specific patterns we watch for because we've been burned. Security antipatterns that slipped through twice before we learned to catch them. Performance gotchas unique to our particular architecture. Accessibility requirements that compliance actually audits.

But the breakthrough wasn't the instructions themselves. It was combining those instructions with executable Python scripts.

One script analyzes cyclomatic complexity and flags functions creeping past our agreed threshold. We'd decided as a team that anything over fifteen decision branches was becoming unmaintainable, but reviewers forgot to check consistently. Human attention is finite.

Another script scans for SQL injection patterns specific to our ORM usage — not generic patterns, but the exact vulnerability shapes that emerge from how we've configured Prisma with our database architecture.

A third validates naming conventions across three different company acquisitions. This matters more than it sounds. When you're maintaining code from three companies with different conventions, consistency becomes the difference between maintainable and nightmarish.

Here's what happens in practice: Claude reads the code with full language understanding. It gets intent, context, architectural patterns, business logic. Then it executes deterministic scripts to validate specific, measurable aspects.

It's hybrid intelligence: LLM flexibility meets code reliability.

The reviews became both more comprehensive and more consistent. Not perfect — nothing is — but measurably better. And critically: reproducible. Same code, same skill, same output. That reliability matters when you're reviewing dozens of pull requests weekly.

The skill structure looked like this:

```
code-review-skill/
├── SKILL.md                    # Review workflow + orchestration
├── scripts/
│   ├── complexity_checker.py   # Flags functions >15 branches
│   ├── security_scanner.py     # Our specific vulnerabilities
│   └── naming_validator.py     # Cross-acquisition consistency
├── references/
│   └── security_incidents.md   # What we learned the hard way
└── examples/
    └── excellent_review.md     # What great actually looks like
```

Nothing fancy. Just organized expertise with clear boundaries.

The SKILL.md workflow encoded what our best reviewers do intuitively but inconsistently:

```
## Phase 1: Architectural Alignment
- Does this change respect our bounded contexts?
- Are we adding necessary abstractions or just adding abstractions?
- Cross-reference against references/security_incidents.md

## Phase 2: Automated Quality Assessment
- Execute scripts/complexity_checker.py on modified functions
- Run scripts/security_scanner.py for our known vulnerability patterns
- Validate naming consistency with scripts/naming_validator.py
## Phase 3: Contextual Human Judgment
- Evaluate team velocity impact
- Identify testing gaps specific to changed behavior
- Verify documentation updates match actual code changes
## Phase 4: Constructive Feedback Formation
- Format using examples/excellent_review.md as reference
- Prioritize clearly: blocking issues vs. nice-to-haves
- Suggest specific solutions, not just problem identification
```

This captures years of learned behavior compressed into a reusable format. The security incidents that taught us what actually matters. The complexity threshold discovered through actual maintenance pain. The naming conflicts we only discovered post-acquisition when onboarding became unnecessarily difficult.

. . .

The Contrarian Take That Might Annoy You (But Is True)

Here's my perspective that probably irritates some people: **Agent Skills aren't fundamentally an AI innovation. They're a software engineering pattern that happens to work brilliantly with AI.**

Everything about Claude Agent Skills — modular architecture, progressive loading, version control, composability — these are established engineering practices applied thoughtfully to AI context management. The breakthrough isn't making AI smarter. It's recognizing that organizing information beats cramming in more information.

We've invested years chasing bigger models, longer context windows, more sophisticated retrieval systems. Agent Skills take the opposite approach: make the AI more *focused* by providing structured access to specialized knowledge only when contextually relevant.

This is why Skills will outlast most AI tooling we're seeing today. They're not dependent on specific model capabilities, context window sizes, or API features that might change. They're just thoughtfully organized folders that any sufficiently capable AI can consume.

When Claude 5 launches with whatever improvements it brings, your skills continue working. When competitors eventually catch up to Claude's capabilities, the Skills pattern transfers seamlessly.

The real innovation is admitting that architecture matters more than raw capabilities. It's not as exciting as discussing emergent behaviors or scaling laws, but it's what actually ships and maintains well.

. . .

Deployment Contexts: Where Skills Actually Live (And Why Each Matters)

Understanding where Agent Skills run changes how you architect them. They're not confined to a single environment.

In Claude AI (web and mobile applications): You upload SKILL.md via attachment and reference it explicitly: *"Using the code-review-skill, analyze this pull request."*

The constraint is skills don't persist across conversations — each new chat requires re-upload. This initially felt limiting until I realized it's actually a feature, not a bug. You're explicit about which expertise is active in each context. No unintended skill interference. Clean mental model. Predictable behavior.

In Claude Code: Skills transform into invisible automation. Install them as plugins in your `~/.claude/skills` directory, and Claude Code automatically detects relevance without explicit invocation.

I commit code. Without prompting, Claude Code recognizes the context, invokes the review skill, executes analysis scripts, and generates detailed feedback. It's workflow automation that understands context. The magic manifests in what you don't have to do anymore.

Via API integration: Skills become genuine product differentiators. We're building a developer platform where each client has unique coding standards, security policies, deployment constraints.

We create a skill per client encoding their specifics — their particular security requirements, their naming conventions, their deployment constraints shaped by their infrastructure. When processing their requests, we attach their specific skill through the `/v1/skills` endpoint.

Same underlying AI. Customized behavior per client. That's impossible to replicate economically with traditional approaches without building completely separate infrastructure per client or maintaining separate fine-tuned models.

This architectural flexibility is where Agent Skills separate themselves from alternatives.

. . .

Lessons From Actually Shipping This (The Hard-Won Kind)

After building multiple skills and observing them operate in production conditions, certain patterns emerge with uncomfortable clarity.

Narrow and deep consistently beats wide and shallow. My first attempt was predictably ambitious — one massive "engineering-expert" skill covering everything from code review to incident response to architectural decisions. Complete disaster.

Unwieldy to maintain. Claude couldn't determine when to activate it. Different sections contained conflicting advice. The skill became a dumping ground rather than focused expertise.

Breaking into tightly-scoped skills — code-review-expert, api-docs-expert, incident-response-expert — worked dramatically better. Each skill does one thing excellently. Claude orchestrates multiple skills when complex tasks require it, but keeping them separate makes them individually reusable, debuggable, and maintainable.

Structure scales better than instructions ever will. When SKILL.md grows beyond a few screens, split it aggressively. Your security checklist belongs in `references/security_standards.md`. Templates deserve separate files. Examples live in `examples/`. Keep SKILL.md laser-focused on workflow orchestration.

Claude loads files on-demand through progressive disclosure, so a skill with ten focused reference files actually performs better than one monolithic file. You achieve token efficiency through architecture, not through compression or summarization tricks.

Test skills with the same rigor you test code. We write explicit test cases for skills now. Example inputs with known correct outputs. Automated validation that behavior matches expectations across versions.

Skills are executable artifacts, not documentation that might drift. Version them properly. Review changes through pull requests. Test thoroughly before deployment. Deploy deliberately with rollback procedures. Treat them with identical rigor to your application codebase.

This discipline prevents the gradual quality erosion that kills so many internal tools.

. . .

The Security Conversation Teams Keep Avoiding

Let's address this directly without sugar-coating: Agent Skills execute code in your environment. This capability is simultaneously powerful and potentially dangerous.

I've reviewed community-contributed skills that looked genuinely helpful on the surface but contained concerning patterns upon closer inspection. Scripts making network calls to undocumented endpoints. File operations accessing more than the description claimed. Dependencies pulling in packages with questionable provenance.

My policy is unambiguous: Only install skills from sources I trust completely and explicitly. Skills I created and control. Official skills from Anthropic. Skills from colleagues whose code I've personally reviewed and would be comfortable maintaining myself.

Everything else receives thorough security audit before any deployment consideration. Not a quick scan — thorough audit. Read every line of every script. Trace every file operation. Understand every network call. Verify every dependency.

For team deployments, this requires process, not just guidelines. Mandatory peer review for all skills. Security team approval for any skill accessing sensitive data or making external calls. Explicit documentation of which skills are approved for which environments. Complete version control with audit trails.

Log skill execution in production. Monitor for unexpected behavior. Have rollback procedures ready and tested.

The risk isn’t theoretical. A malicious or compromised skill could exfiltrate data, modify critical files, execute arbitrary operations, or create backdoors. The same flexibility making Skills powerful makes them security-sensitive.

Don’t let this prevent you from using Skills — just treat them with appropriate caution and systematic rigor. The security model is actually clearer and more auditable than most AI tooling because the code is directly reviewable. No black boxes. No hidden API calls with unclear behavior. Just Python scripts you can read, understand, and audit.

> Security through transparency, not through obscurity.

. . .

A Realistic Implementation Path (Not Another Grand Strategy)

Forget quarterly roadmaps and phased rollouts. Here’s what actually works based on real implementation experience.

Pick one workflow that genuinely frustrates your team. Not the most strategic opportunity. Not the highest theoretical ROI. The workflow that makes people audibly groan when they realize they have to do it. Code reviews taking too long. Documentation always inconsistent. Security checks getting forgotten under deadline pressure.

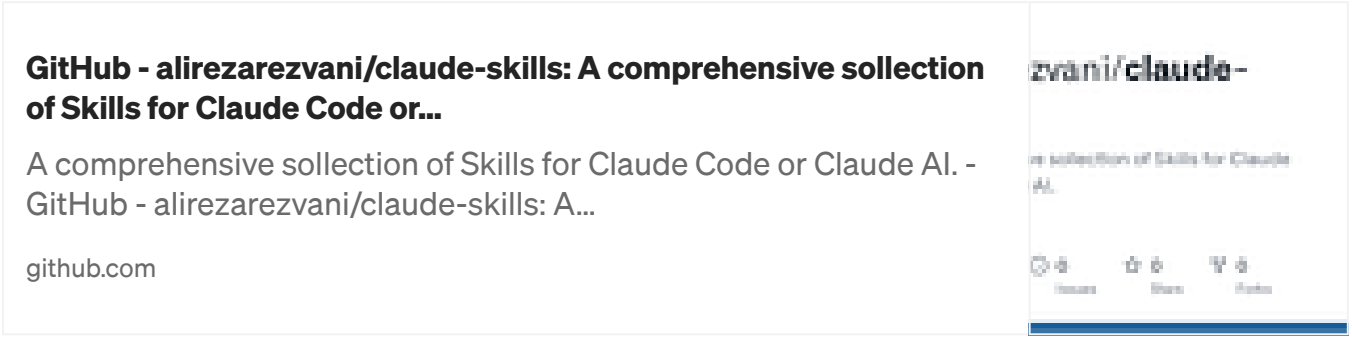
This weekend — yes, this weekend — document how your best person actually does this task. What do they check first? What's their mental process? What makes their output noticeably better than average? Write this in clear prose before thinking about skills syntax or structure.

Create the minimal skill structure. Just SKILL.md with that workflow clearly documented. No scripts yet. No fancy reference files. Get the core instructions correct first. Test it yourself extensively. Where does Claude Excel? Where does it struggle or misunderstand?

Add exactly one script for the most critical validation step. Just one. Make it work reliably. Test it until you trust it completely. Then deploy to a small group of three to five people you trust to give honest, detailed feedback.

Collect real feedback from actual usage under genuine working conditions. Iterate based on reality, not assumptions about how you think people will use it.

This approach isn't impressive. It won't make a compelling all-hands presentation. But it's honest about how learning actually works in real organizations. You'll learn more from building one complete skill through full deployment than from reading documentation for weeks or attending conferences about AI strategy.



• • •

The Deeper Pattern Everyone's Missing

Agent Skills represent something more fundamental than a new Claude feature or a clever prompt engineering technique.

They're a format for organizational learning that actually persists beyond individual conversations and individual people.

Every company has critical expertise living exclusively in people's heads. Security practices learned through painful incidents. Code patterns that prevent subtle bugs discovered through production failures. Documentation standards that make onboarding genuinely faster. Performance optimizations discovered through careful profiling.

This knowledge is valuable — sometimes extraordinarily valuable — but it's rarely captured systematically. It lives in Slack messages, pull request comments, verbal explanations during onboarding, and institutional memory that walks out the door when people leave.

Agent Skills provide tangible structure for encoding this expertise into reusable, version-controlled, testable artifacts. They transform ephemeral tribal knowledge into durable organizational assets. And because they're just files and folders using standard formats, they integrate seamlessly with all the existing tooling teams already use for managing code.

Organizations figuring this out early create compounding advantages that accelerate over time. Not because they have access to better AI — everyone can use Claude — but because they're systematically capturing and codifying expertise instead of letting it evaporate between conversations, get lost in chat archives, or disappear when people change roles.

This isn't ultimately about AI capabilities or context windows or model parameters.

It's about organizational learning architecture that compounds rather than resets.

. . .

Start Building (The Revolution Starts Small)

The comprehensive documentation exists. The production-tested examples are available. The growing community is sharing patterns openly. What's missing is simply your expertise, packaged appropriately.

Clone the official repositories and explore real examples:

- Anthropic's Skills Repository — Production-tested document editing skills, example patterns, templates
- Community Skills Collection — Real-world marketing, engineering, and leadership skills

Pick one workflow that frustrates your team. Build a skill for it this weekend. You'll learn infinitely more from shipping one real skill than from reading additional articles or attending more webinars about AI strategy.

The future of enterprise AI isn't one-size-fits-all foundation models serving identical responses to everyone.

It's specialized agents understanding your domain, your constraints, your hard-won lessons as deeply as your best people do.

Agent Skills are the architecture pattern making this future not just possible but practical to implement starting today.

And it begins with one folder, one SKILL.md file, and the willingness to capture what your team already knows but hasn't yet codified.

. . .

Your Turn: Let's Build This Ecosystem Together

Drop a comment below with the specific workflow you're tackling first. I read and respond to every comment. Let's learn from each other's real implementation experiences as this ecosystem develops.

The question isn't whether to build Skills. The question is whether you'll systematically capture your expertise or let it continue evaporating between conversations.

The revolution won't be televised or announced at conferences.

It's happening quietly in version-controlled repositories of organized folders, compounding expertise one skill at a time, transforming how teams work with AI from transactional conversations into persistent intelligence.

Start this weekend. Build something real. Share what you learn.

. . .

About This Article

Last Updated: October 2025
Verification: All code examples tested, repository links verified, technical claims fact-checked against official Anthropic documentation

. . .

Getting started

- Claude apps: User Guide & Help Center
- API developers: Documentation
- Claude Code: Documentation
- Example Skills to customize: GitHub repository
- nginty Claude Skills for teams: Github repository

If This Resonated, Take Action

👏 **Clap generously** :) That motivates me every day writing for you here.
💬 **Comment** with your first skill idea
🔗 **Share** with your engineering team
📧 **Subscribe** for weekly deep-dives on AI-augmented engineering

Long-form technical articles like this require substantial research, testing, and writing time. Your engagement directly supports creating more comprehensive, production-tested content instead of surface-level AI hype.

. . .

About the Author

Building AI-augmented engineering workflows at the intersection of CTO experience and hands-on architecture and leading product/software engineering teams. Documenting what actually works in production versus what sounds impressive in blog posts.

Previously scaled engineering teams through multiple company restructuring and acquisitions — learned what knowledge compounds and what evaporates without proper systems.

Connect: LinkedIn | **Read more:** Medium Reza Rezvani | **Explore:** GitHub

. . .

Continue Learning

Related Articles:

- *Building Production-Grade Claude Code Workflows*
- *From Tribal Knowledge to Organizational Assets: Documentation Patterns That Work*
- When the Ground Shifts: Leading Engineering Teams Through the Anxiety We All Feel

Official Resources:

- Agent Skills Documentation — Anthropic's technical reference
- Engineering Deep-Dive — Architecture and design decisions
- Skills API Quickstart — Production deployment guide

Developer Productivity | Claude Code | Software Engineering | Claude Skills | Anthropic Claude

Follow

Published in nginity

34 followers · Last published Oct 19, 2025

This publication chronicles my journey through agentic AI development, architectural decisions that shape startups, and the evolving landscape of AI-augmented engineering. Here, you'll find: Startup Engineering Realities—From MVP to scale, the technical decisions ...

Following

Written by Reza Rezvani

954 followers · 72 following

As CTO of a Berlin AI MedTech startup, I tackle daily challenges in healthcare tech. With 2 decades in tech, I drive innovations in human motion analysis.

Responses (1)

Bgerby

What are your thoughts?

Farhad Nawab
Oct 21

This piece convincingly argues that Claude Agent Skills are the architectural solution to AI context fatigue, transforming ephemeral tribal knowledge into persistent, modular, version-controlled intelligence that

allows AI to scale organizational... more

2 1 reply Reply

More from Reza Rezvani and nginity

Reza Rezvani

“7 Steps” How to Stop Claude Code from Building the Wrong Thing (Part 1): The Foundation of...

Learn how to stop Claude Code from rewriting your architecture with vague prompts. This guide introduces Spec-Driven Development...

Sep 17 44 2

In nginity by Reza Rezvani

I Let Claude Sonnet 4.5

IMAGINE this: It's 6 a.m., the kind of quiet dawn where the world's still wrapped in that soft, hazy light filtering through your blinds...

Sep 29 101 1

In nginity by Reza Rezvani

How Cursor and Claude Code Plugins Turned Me Into a 20x Developer – The Agentic Coding Setup That...

My Background Agent just submitted a PR that made our senior architect ask, “Who wrote this?”

Oct 14 73

Reza Rezvani

I Discovered Claude Code’s Secret: You Don’t Have to Build Alone

I’ve been coding long enough to know that the late-night debugging sessions aren’t glamorous. They’re just necessary.

Sep 19 151 2

See all from Reza Rezvani

See all from nginity

Recommended from Medium

In Dare To Be Better by Max Petrusenko

Claude Skills: The $3 Automation Secret That's Making Enterprise Teams Look Like Wizards

How a simple folder is replacing $50K consultants and saving companies literal days of work

Oct 17 398 6

In Realworld AI Use Cases by Chris Dunlop

The complete guide to Claude Code's newest feature "skills"

Claude Code released a new feature called Skills and spent hours testing them so you don't have to. Here's why they are helpful

Oct 21 95 4

In AI Software Engineer by Joe Njenga

Why Claude Weekly Limits Are Making Everyone Angry (And $100/Month Plan Will Not Save You)

Yesterday, I finally hit my weekly Claude limit, and I wasn't surprised, since I see dozens of other users online going crazy over these...

Oct 19 122 24

In Towards AI by Gao Dalie (高達烈)

RAG is Not Dead! No Chunking, No Vectors, Just Vectorless to Get the Higher Accuracy

Over the past two years, I have written numerous articles on how Retrieval-Augmented Generation has become a standard feature in nearly all...

Oct 17 699 6

In Coding Nexus by Code Coup

Claude Desktop Might Be the Most Useful Free Tool You'll Install This Year

I didn't expect much when I first saw the announcement for Claude Desktop. Another AI wrapper, I thought. Maybe with a shiny UI.

6d ago 246 6

In nginity by Reza Rezvani

Master Claude Code “Skills“ Tool to transform repetitive AI prompts into permanent, executable...

Discover how the Anthropic’s new tool for Claude Code called “Skills” transform AI Coding assistant from a generic assistant into your...

Oct 17 121 2