AI Efficiency Is Knowledge and Skills, Not Access

Everyone's talking about cheaper models and local models to lower AI costs. Almost nobody's asking whether we're using LLMs efficiently, or whether the people using them were ever taught how.

Daniel Rodriguez • Founder & CEO, Infragil • September 2026

I've been having some interesting conversations lately, because the same questions keep coming up, each with a different "correct" answer, and there's been surprisingly little discussion about what's underneath them. Let me start with two I got recently.

"How do I reduce my token usage and cost?"

The common answers: use a cheaper, more cost-effective model for some of the tasks. Use local models to start the work and hand off to a frontier model to expand or finish it. Clear your context regularly, because a 1M-token context window adds up fast.

"How do I restrict my agent's permissions so it can only do exactly what it's supposed to?"

This one came from someone using an agent to onboard new employees at their company. The standard answer: create a restricted identity for the agent with very limited access. Better than nothing. Still not great.

> **Different people, different jobs, different problems. Or are they?**

All of those are real answers. I'd even give some of them myself. But they're bandaids on the same wound, and none of them meaningfully move the needle on cost or security. The honest answer to both questions is the same: the LLM shouldn't be doing that work in the first place. The person asking doesn't fully understand what they're delegating, so they're delegating everything, and both the token bill and the permission problem are symptoms of that.

The industry conversation about AI cost right now is all about models. Switch to a cheaper one. Run a local one. Quantize it, distill it, route to something smaller. Model, model, model. Almost nobody is asking the questions that actually matter:

Should AI be doing this work at all?

Why is a reasoning model doing deterministic, scriptable work?

Because here's what gets missed: most of this work is scriptable. Reusable code that runs in milliseconds instead of minutes. Code that's testable. Code that's predictable, producing the same result every single time, under the same identity, with a log to prove it. Paying an LLM to perform these tasks directly costs dramatically more per execution and gives you less reliability in return.

So why does everyone reach for the LLM? Because it's easier to plug in. No development, no testing, no deployment pipeline. You type what you want and it happens. That convenience is real, and it's exactly the trap: the easiest tool to start with is the most expensive tool to keep running. And when the person plugging it in doesn't have the experience, training, or guidance to know the difference, the LLM quietly becomes the answer to everything.

Let me walk through those two examples properly.

"How do I restrict my agent so it can safely onboard users?"

I gave the textbook answer above: a dedicated identity, scoped to minimum permissions, with the agent impersonating it to do the work. Least privilege, service accounts, the stuff my profession has preached forever. That answer is correct.

But it's the answer to the wrong problem.

Step back and look at what's actually happening. User onboarding is one of the most repetitive, deterministic, well-understood processes in all of IT. Same steps, same systems, same permissions, every single time. People who've been in this field for years didn't need an LLM for this. They wrote scripts. They built integrations. They automated it once and moved on.

So the real answer isn't "restrict the agent." It's this: have the LLM write the script. A tested, reusable script that the agent can call, or the user can call, running under its own scoped identity. Better yet, build an integration between the HR system and the IT system so that when a new hire shows up in one, the right permissions appear in the other. Automatically. No LLM in the loop at all.

That version is faster. It's more reliable. It's auditable. Your permission problem shrinks to a single service account doing a single job. And your token cost for onboarding drops to zero.

"How do I reduce my token count?"

Different person, this time a developer, tokens burning too fast, bill too high.

The textbook answers are the ones up top: smaller models, local models, cheaper models, trimmed context. Those help around the edges, and sure, do them. But notice they all treat the symptom. The workflow stays exactly as wasteful; you just pay a lower rate for the waste.

Watch what's actually consuming the tokens. A lot of development work with LLMs is repetition. Tell an agent to check in code and it will generate the command-line commands fresh, every time. If something errors, it'll reason through it, retry, maybe generate the commands two or three more times until it gets it right. And it will do that whole dance again tomorrow. And the day after. Every developer, every session, every check-in.

Versus a script. Written once, tested once, runs under a separate identity, does the same work in milliseconds for zero tokens, forever.

> **The model should be creating the automation, not being the automation. Save the inference for the real work: the design decisions, the tricky debugging, the code review, the judgment. That's what reasoning models are actually good at, and it's the part a script can't do.**

A simple example showing 94% fewer tokens

Don't take my word for it. Here's a real example from an agentic workflow in my own stack.

The objective was simple: determine the current status of several application services. The first version did what most agent builds do. It had the agent inspect the environment directly:

Step	Approximate tokens
`docker ps` output	200
Scanner logs, 150 lines	2,500
Trading-service logs, 150 lines	2,000
Position-monitor logs, 100 lines	1,500
Reasoning and retries when names don't match	1,000
Total	**~7,200**

And that's a good run. On a busy day, with more activity, errors, extra scans, or a log tail that doesn't go back far enough, the agent needs several more rounds of retrieval and reasoning. The token count multiplies.

But look at what the agent is actually trying to accomplish. It doesn't need 400 lines of logs. It needs the answer to a much simpler question: **what is happening right now?**

So instead of repeatedly paying an LLM to retrieve raw operational data, interpret it, and reconstruct application state from scratch every time, we built a deterministic activity interface that does that work once. Plain, boring, conventional software. Now the same status check looks like this:

Step	Approximate tokens
One `/api/activity` request or database query	300–500
Typical total	**~400**

That's roughly a 94% reduction, about 18 times fewer tokens, for the same answer. If status gets checked three to five times in a working session, that one change avoids 20,000 to 34,000 tokens. Every session. Forever.

And there's a better outcome hiding behind it. We put the same structured data on a dashboard, and now the human often doesn't ask the agent at all. For those queries, the token cost isn't reduced. It's zero.

Notice what the solution was not. It wasn't a cheaper model, a shorter prompt, a smaller context window, or telling the agent to be more concise. Those might have shaved the cost a little. We changed the architecture instead. The LLM had been consuming piles of raw machine data to repeatedly infer something conventional software could calculate once and serve in a few hundred bytes. We moved the deterministic work out of the reasoning layer.

> **7,200 tokens became roughly 400. Not because the model got better. Because the system design got better.**

That's the optimization enterprises should be hunting for. Not "how do we make this prompt 20% cheaper?" but "why are we invoking a reasoning model for this operation at all?"

Another example: PDF file analysis

Here's an everyday version of the same mistake, one almost every team makes.

Someone needs to analyze a PDF, so they drop the whole document on the LLM. A 100-page contract, a vendor's SOC 2 report, a stack of invoices. The model ingests every page: the boilerplate, the headers and footers, the legal disclaimers, the table of contents, all of it, at tens of thousands of tokens per document. Do that across a team, across hundreds of documents a month, and you've built a very expensive PDF reader.

But extracting text from a PDF isn't reasoning. It's parsing. Scripts have done it reliably for decades. So put a script in front of the model: extract the text, pull out just the sections that matter for the question being asked, and hand the LLM a few hundred tokens of relevant content instead of the entire document. If you're processing the same document type over and over, the script can find the fields you care about, the renewal date, the liability cap, the audit findings, before the model ever sees a word.

The LLM still does what only it can do: interpret the clause, flag the risk, summarize the findings, answer the question. It just stops paying by the token to do the part a parser handles for free. Same answer, a fraction of the cost, and as a bonus, less irrelevant context means the model is less likely to get distracted or wander. Preprocessing isn't just cheaper. It usually makes the answers better.

> **Same lesson, third time: extraction is execution. Analysis is reasoning. Pay for reasoning.**

None of this is new

Programmers, DBAs, infrastructure engineers, and sysadmins have relied on scripting and reusable code for at least 30 years. Automate the repetitive thing, test it, run it under the right identity, log it, reuse it. That's not innovation. That's Tuesday.

What it does require is background. You have to know what's possible, how to implement it, and why it's worth doing. The people asking me these questions aren't dumb. They're smart people who were handed a powerful tool without the engineering context that makes the tool cheap and safe. So they use the LLM for everything, because the LLM is the interface they were given, and it's the one that plugs in without asking anything of them.

Think about what that means at scale. A person without automation experience will use the agent to run the same commands, re-read the same context, troubleshoot the same predictable errors, and reconstruct the same workflow, every single time. They may save an hour of manual work while the agent quietly consumes thousands of tokens doing something that should have been implemented once as a script. At small scale you won't notice. At enterprise scale, it's an operating-cost problem with your name on it.

And here's the uncomfortable connection: this is one of the reasons companies aren't seeing cost reductions after AI-driven layoffs. If the work people used to do gets transferred into poorly designed, token-hungry agent workflows, the expected savings disappear into inference costs, infrastructure, licensing, and overhead. In some cases the repetitive token burn approaches or exceeds what the people cost. That's not transformation. That's moving the inefficiency somewhere harder to see, and paying a metered rate for it.

Upskilling is part of the architecture

There's an organizational failure hiding under all of this, and most enterprise AI rollouts walk right into it.

> **Giving employees access to AI is not the same thing as teaching them how to use AI effectively. Access is not ability.**

If a company is serious about deploying AI at scale, it has to invest in upskilling right alongside it: training, practical guidance, coaching, examples of good architecture, hands-on workshops that show people when to use an LLM and when to automate the work another way. The goal isn't to make every employee better at prompting. The goal is to help employees recognize opportunities to redesign their work.

That means teaching people to ask questions like:

- Is this task repetitive?
- Is the output deterministic?
- Could this become a script, a function, an API call, or a workflow?
- Should the LLM build the automation instead of executing every individual step?
- Does this action require reasoning, or just execution?
- What identity should perform the work?
- What permissions does that identity actually need?
- Can the process be tested, logged, and reused?

Notice something about that list. The token question and the security question are on it together, because they were never two different problems. An agent that only handles the judgment work, while tested automation handles the execution under scoped identities, is simultaneously cheaper, faster, more reliable, and dramatically easier to secure. You don't have to choose.

Without that guidance, you end up with employees using LLMs as extremely expensive automation engines. With it, you get something genuinely new: a workforce that can automate its own work, safely.

For experienced developers, DBAs, and infrastructure people, this thinking has existed for decades. The challenge now is teaching those same engineering principles to a much broader group of employees who suddenly have the ability to automate their own jobs. That's not a

tooling problem. That's a mentoring, coaching, and training problem, and it's exactly where your experienced people earn their keep in an AI rollout.

Done well, AI improves automation and efficiency.

Used as an easy button, it creates technical debt, inefficiency, and security issues.

The problem was never that AI is inherently expensive. The problem is inefficient AI architecture in the hands of people nobody bothered to train. Replacing a repetitive human process with a repetitive LLM process isn't transformation. Use the model to understand the process, identify what can be automated, help build that automation, and then spend your inference on the parts that actually require thought.

The companies that get the most value from AI won't be the ones that buy the most tokens or deploy the most agents.

They'll be the ones that teach their workforce when not to use them.

At Infragil, we help companies use AI responsibly and effectively, and that includes teaching your people the difference between work that needs a reasoning model and work that needs a script. Training, architecture guidance, workshops, and the governance to hold it together. If your token bill is climbing and your agents have more permissions than your admins, let's talk.

Daniel Rodriguez is the Founder & CEO of Infragil, an AI advisory and cybersecurity practice, and creator of Skopos, an AI-native third-party risk management platform. He has spent 29 years in cybersecurity, including CISO and VP roles.

Sources

• Internal Infragil workflow measurements: token counts from a production agentic status check before and after the architectural change (~7,200 to ~400 tokens)
• Gartner (Mar 2026): agentic workflows consume 5 to 30 times more tokens per task than standard chatbot queries
• Gartner (Aug 2026): the "Inference Paradox"; AI inference costs per agentic workflow forecast to rise more than fivefold through 2028 despite falling per-token prices
• Gartner, 2026 AI Hype Cycle: 40% of AI agent projects forecast to be canceled by 2027, driven by cost overruns
• Forrester, Predictions 2026: 55% of employers regret AI-driven layoffs
• Careerminds (Feb 2026): two-thirds of companies that conducted AI-related layoffs are rehiring, over half within six months
• Orgvue: companies spend roughly $1.27 for every $1 saved through workforce reductions