hanzo.ai vs. claude.ai: Owned, Self-Hostable AI versus a Centralized Cloud

Zach Kelling — Hanzo Team

Version 1.0 2026

Abstract

We compare two structurally different ways to obtain frontier AI: a closed, centrally hosted model accessed as a metered cloud service (typified by Anthropic's Claude), and a sovereign, self-hostable stack with owned model weights (Hanzo). The distinction is not primarily one of model quality—closed frontier models are excellent—but of *deployment model and control*. For regulated industries, critical infrastructure, and government (including defense), the binding constraints are data residency, operation in disconnected and critical environments, supply-chain provenance, post-quantum security, auditability, and cost at scale. A closed cloud service cannot satisfy several of these by construction: it cannot run air-gapped, its weights cannot be inspected or owned, and sensitive data must leave the enclave to be processed. We argue that for these workloads the relevant choice is between foreign open-weight models and an American open-source alternative, and we position Hanzo as the latter—owning both the model (the Zen family) and the cryptography (a production post-quantum stack). We give a dimension-by-dimension comparison and a decision framework for when each model of consumption fits. These constraints are most acute in financial services, healthcare, critical infrastructure, and government systems—defense among them—but the framework applies to any organization that must control where its data and models live. We also make the productivity case concrete: on a regulated capital-markets build where every engineer had the same frontier chat model, the client's own engineering audit found that one engineer working through Hanzo's agentic stack out-produced entire contractor teams by the audit's own unit-cost metrics—the differentiator being the owned agentic *stack*, not the model.

Executive Summary. This is not a claim that our model is smarter than Claude. It is a claim about *where* and *how* you can run it. Claude is an excellent frontier model delivered as a managed cloud service: you call it over the internet, your data goes to the vendor, you cannot see or own the weights, and it cannot run on a disconnected device. That is the right product for many commercial uses and the wrong one for a workload that cannot leave a controlled environment. Hanzo is the opposite shape: open, owned-weight models you run on your own hardware—in a data center, at the edge, or air-gapped—with post-quantum security built in, your data never leaving, and cost driven down because you are not paying a per-token markup or a hyperscaler margin. America's frontier AI has narrowed to two closed labs; the only openly available frontier weights today are largely foreign. For regulated industries, critical infrastructure, and government buyers—defense included—who need open, sovereign, controllable AI, that makes Hanzo the American alternative to both the closed duopoly and foreign open weights.

Contents

1 The Real Axis of Comparison

Discussions of "which AI is better" usually compare benchmark scores. For commercial chat and coding that is a reasonable proxy. For regulated and sovereign deployment it is the wrong axis. A model that scores two points higher on a reasoning benchmark is of no use inside a controlled environment if it can only be reached by sending the data outside that environment.

The decisive axis is the *deployment and trust model*: who holds the weights, where inference physically runs, whether the data leaves the enclave, whether the system works without connectivity, what cryptography protects it, and what it costs at scale. On this axis a closed cloud service and a sovereign self-hosted stack are not competitors with different scores—they are different categories of thing. Claude is the leading example of the first category; Hanzo is built as the second.

We state plainly what closed cloud AI does well: it is operationally simple (no infrastructure to run), it tracks the frontier of raw capability, and its provider invests heavily in alignment and safety research. None of that is in dispute. The argument here is narrow and structural: several requirements that are mandatory for regulated, critical-infrastructure, and government workloads (defense among them) cannot be met by a model you do not hold and cannot run yourself.

2 Case Study: One Engineer on the Hanzo Stack vs. a Team on Claude

The cleanest evidence for this paper's thesis is not a benchmark—it is a controlled natural experiment that already ran in production, on a real regulated system, and was measured by that system's own engineering audit.

2.1 Background: a regulated platform, a sprawling estate

The setting (anonymized) is a U.S. regulated securities venue—an SEC-registered alternative trading system with broker-dealer and transfer-agent functions, among the most compliance-heavy software domains that exists. By the time of the engagement it had grown into a sprawling legacy estate: **200+ microservices** across **334 repositories**, roughly **$35,000/month** of public-cloud spend, accumulated security debt, and a hard regulatory bar to clear. The engineering organization was a set of contractor teams, and every engineer on the program had access to the same frontier chat model (Anthropic's Claude, via `claude.ai`).

2.2 The intervention: same model, different stack

The variable that differed was the *tooling around* the model. One engineer worked through Hanzo's owned, agentic stack (`hanzo.ai`)—an orchestration layer that turns the model into an autonomous build–test–ship loop across hundreds of repositories—while the contractor teams used the model the conventional way, as an interactive assistant alongside hand-driven development. This is *not* a claim that one group had a smarter model: everyone held the same frontier weights. What separated the outcomes was whether the model was wrapped in an agentic stack or merely chatted with.

2.3 The measured comparison

The platform's own engineering audit covered 334 repositories and 31,058 commits over an 81-day window, attributing every commit from raw version-control history and normalizing cost against each contributor's monthly spend. We report the three metrics the audit treats as defensible: total

commits shipped, unit cost per line of code ($/LoC), and shipping rhythm per dollar of monthly spend (commits per $1,000). The contractor pods are anonymized as Team A, Team B, and Team C.

Group (same frontier model)	**Commits**	**$ / LoC**	**Commits / $1k**
One engineer on the Hanzo agentic stack	15,294	$0.0006	588
Team A (contractor pod, conventional tooling)	2,112	$0.179	105
Team B (contractor pod, conventional tooling)	498	$0.560	10
Team C (contractor pod, conventional tooling)	1,415	$0.153	32

Table 1: The client's own engineering audit, anonymized. All contributors used the same frontier chat model; only the single engineer's work was multiplied by Hanzo's agentic stack. By the audit's own unit-cost metric, that engineer's cost per line of code was roughly 250× to 900× lower than the contractor pods', and that engineer alone shipped more commits than the three contractor pods combined.

Reading the numbers honestly. The audit is explicit that the raw lines-of-code figure for the agentic-stack engineer is inflated by framework-wide refactors, bulk dependency upgrades, and automation identities operating under the same attribution. We therefore do *not* headline raw lines authored; the defensible signals are (i) the *commit count*, which reflects discrete units of shipped, reviewed work; (ii) the *cost per line*; and (iii) the *relative ratio between groups*, since every group's lines were counted by the identical method, so whatever inflation exists applies uniformly and the 250×–900× unit-cost gap survives it. Even read at its most conservative, one engineer plus an agentic stack delivered a team's worth of audited, shipped output at a fraction of the cost per line.

2.4 The outcome: it shipped

The comparison is not about commit velocity in the abstract—the same engagement *shipped the platform*, and cleared the bar that matters. Over the window the estate was consolidated from **200+ microservices into 3 compiled binaries**; the codebase went from **3.07M lines to 1.06M active** (∼3.5 M lines of dead code deleted); a source audit of **93 findings** (22 critical) plus **∼780 dependency vulnerabilities** was driven to **zero actionable**; and cloud spend fell **∼90%** (∼$35K/mo → ∼$3.5K/mo, ∼$380K/yr). The system cleared **FINRA/SEC approval** and **SOC 2**, in record time—with the bulk of that work carried by the single agentic-stack engineer; a focused **10-day remediation sprint** alone produced **3,694 commits** and **100% of the critical- and high-severity security fixes**. The full cost-and-consolidation breakdown is its own companion case study; the point here is narrower but decisive: the productivity gap above was *not* bought by cutting corners—it cleared a regulated, externally audited production bar that the contractor teams, on the same model, did not.

Dimension	**Before**	**After**
Microservices	200+	3 compiled binaries
Codebase (active)	3.07M lines	1.06M lines (−70%)
Cloud spend	∼$35K/mo	∼$3.5K/mo (∼90%)
Open critical/high findings	93 (22 critical)	0 actionable
Dependency vulnerabilities	∼780	0 actionable
Regulatory status	legacy, unapproved	FINRA/SEC approved, SOC 2

Table 2: The engagement outcome, anonymized. The productivity comparison was delivered *into* this result—a regulated, audited production system—not alongside a prototype.

Why the stack wins (the mechanism). The audit is one measured data point; the reason to expect it to reproduce is structural, not anecdotal. Interactive chat is human-paced and serial: a prompt is written, an answer is read, code is pasted, the loop turns once, and the engineer is the bottleneck on every turn. An agentic stack inverts the bottleneck—the model is wrapped in an autonomous build–test–ship loop that reads the repository, makes the change, runs the tests, reads the failures, and iterates, across many repositories at once, while the engineer supervises rather than types. Throughput then scales with the orchestration and the available compute, not with one person's typing speed. Three structural factors compound:

- **The model is one component, not the product.** The stack supplies the memory, tool use, sandboxes, retrieval, identity/secrets, and CI integration that turn a good answer into a merged, tested commit. Chat supplies only the answer; a human supplies the rest, by hand.
- **Parallelism over serialism.** Many agent loops run concurrently across the codebase; a chat session is one conversation at a time. The productivity gap is the gap between a fleet and a single operator.
- **Owned and unmetered.** Self-hosted on owned hardware, the loop runs at full throttle without a per-token meter deciding how much iteration is affordable—and iteration is exactly what produces tested, reviewed work.

The same frontier weights, accessed two ways, land in two different productivity regimes. The model is a commodity input; the agentic stack is the durable advantage—which is why the gap is a property of the *method*, reproducible by any team that adopts the stack, not a property of one exceptional engineer. The measured 250×–900× is the proven floor; the theoretical case says it travels.

What switching gets you. For an engineering organization the practical read is direct: adopting Hanzo's stack force-multiplies the engineers already on payroll—same people, same frontier model, now wrapped in an owned orchestration layer—so legacy modernization, regulated rebuilds, and net-new work all ship at a fraction of the cost per line, on infrastructure the organization owns and can audit. The switch is low-friction (the stack wraps the model access a team already has) and the gain is structural (it is the method, not the model). That is the case for moving from renting a chat window to owning an agentic stack.

What this is, in one line. This is the `hanzo.ai`-versus-`claude.ai` comparison made concrete. It is not "our model is smarter"—everyone had Claude. It is that an *owned agentic stack* turns one engineer into a team's worth of shipped, audited output, at a fraction of the cost per line. The same ownership that buys sovereignty in the rest of this paper—self-hostable weights, post-quantum

security, data that never leaves the enclave—is what buys this productivity edge. You own the stack; you do not rent the chat. A buyer adopting Hanzo gets both halves of that sentence at once.

3 Dimension-by-Dimension

Dimension	**Closed cloud AI (e.g. Claude)**	**Hanzo (sovereign OSS)**
Model weights	Not disclosed; not transferable	Owned (Zen family, 0.8B–1T+); inspectable
Where inference runs	Vendor cloud only	Your hardware: data center, edge, air-gapped
Your data	Leaves the enclave to the vendor	Never leaves; stays where it lives
Disconnected / offline	Not possible (requires connectivity)	First-class (offline on embedded ARM)
Provenance	Vendor-controlled	American-owned, full provenance
Post-quantum security	Not a product property	Native (ML-KEM-768 + ML-DSA-65)
Auditability	Black box	Open stack; cryptographically signed actions
Customization	Prompting / limited fine-tune via API	Full fine-tune on your data, in your enclave
Cost model	Per-token metered	Owned + self-hosted; no markup or cloud margin
Lock-in	High (API + data gravity)	Low (you hold the stack)

Table 3: Closed cloud AI versus a sovereign self-hostable stack. The comparison is of deployment model, not of raw benchmark quality.

3.1 Data residency and the enclave boundary

The single most consequential difference is the enclave boundary. To use a closed cloud model, the data being reasoned over must cross the boundary to the vendor. For protected health information, customer financial records, regulated controlled data (CUI included), engineering files, or operational data this is often simply prohibited. Hanzo runs the model inside the boundary, so the data never crosses it. This is not a policy preference that can be waived with a contract clause; it is a property of where computation physically happens.

3.2 Disconnected and offline operation

A cloud service is unavailable the moment the link drops. On a remote industrial site, aboard a vessel, in an air-gapped facility, or at any field location where connectivity is intermittent or denied—disconnected defense operations included—there is no link to call. Hanzo's models are quantized to run fully offline on hardware ranging from a laptop to an embedded processor, so capability persists in exactly the environments where a cloud service cannot operate at all.

3.3 Provenance and the supply chain

For an open, controllable alternative to a closed cloud model, today's practical options are foreign open-weight models or Hanzo. Regulated enterprises and government buyers (defense among them) care about who produced the weights and whether they can be trusted and inspected. Hanzo's

position is specific: an American-owned, open-source frontier stack with owned weights, so an organization can have open, auditable AI without depending on either the closed domestic duopoly or foreign-origin weights.

3.4 Post-quantum security

Security against a future quantum adversary is a property of the whole stack, not a feature of a chat API. Hanzo's transport, identity, and key management are post-quantum by construction (all three NIST standards in production, formally verified). A closed chat service does not expose this as something the customer controls.

3.5 Cost at scale

A metered API charges per token on every inference, forever, with the vendor's margin baked in. Owning the model and self-hosting removes both the per-token markup and the hyperscaler margin; combined with an efficient transport layer (the ZAP protocol's 91% memory and 96% energy reductions) this is the mechanism behind roughly an order-of-magnitude lower infrastructure cost at scale. A companion case study documents a regulated client whose cloud spend fell approximately 97% after migrating to this stack.

3.6 Agentic throughput, restated as cost

The productivity comparison in Section 2 is also a cost argument. Hanzo's founding CEO/CTO, Zach Kelling, was—by public usage tracking (`claudecount.com`)—the **#1 user of Anthropic by token count worldwide in 2025**; the audited output behind that throughput, one engineer outproducing entire contractor pods on the same frontier model, is documented above. The point here is what changed afterward: that same agentic throughput now runs on Hanzo's *owned*, self-hostable stack rather than a metered cloud API. The capability that topped a closed provider's global usage is exactly the capability Hanzo packages as a sovereign, owned alternative, so a buyer gets the productivity without the per-token meter, the data egress, or the dependency—frontier-grade agentic output, owned instead of rented.

4 When Each Model Fits

This is a decision framework, not a verdict. Closed cloud AI is the right choice when the workload is non-sensitive, connectivity is reliable, operational simplicity outweighs control, and there is no requirement to own or inspect the model. A sovereign self-hosted stack is the right—often the only—choice when any of the following hold: the data cannot leave a controlled environment; the system must work disconnected or in degraded conditions; provenance and auditability are required; post-quantum protection is mandated; or per-token cost at scale is untenable. Regulated industries, critical infrastructure, and government workloads—defense included—concentrate in the second column. That is the gap Hanzo is built to fill.

5 Conclusion

The honest comparison is not "Hanzo's model beats Claude." It is that a closed cloud model and a sovereign self-hostable stack are different categories, and the requirements of regulated, critical-infrastructure, and government workloads (defense among them)—data residency, disconnected

operation, provenance, post-quantum security, auditability, and cost—select the second category by construction. Within that category the choice narrows to foreign open weights or an American open-source alternative. Hanzo is built to be that alternative: own the model, own the cryptography, run it where the data lives.