We Spent $18 Benchmarking Self-Hosted LLMs. The Cheapest APIs Still Won — Except When They Didn't.

The same GPU-hour, the same model, the same task: 122,733 requests one afternoon, 56,987 the next. Nothing about the infrastructure changed. Only the traffic did.

The question everyone prices wrong

Every team building on open models eventually faces the same fork: pay a provider per token, or rent a GPU by the hour and serve the model yourself. The debate usually runs on the wrong variables — model quality, platform features, vendor preference. So we stopped debating and measured it.

Our setup was deliberately simple. Modal's new Auto Endpoints deploy an open model behind an OpenAI-compatible API in one command, billed per second of GPU uptime, scaled to zero when idle. We deployed two models at opposite ends of the scale — Gemma 4 E2B (a 2B-class model on one A100, ~$2.50/hour) and OpenAI's GPT-OSS-120B (117B parameters, on one B200 at $9/hour) — and ran each through an identical one-hour protocol: five minutes at concurrency 1 to measure what a single user feels, fifteen at concurrency 16 to simulate a realistic pipeline, forty at concurrency 48 to find the ceiling. The task was sentiment classification with strict JSON-schema output — intentionally easy, because we were testing the infrastructure and its economics, not the model's intelligence. One variable at a time.

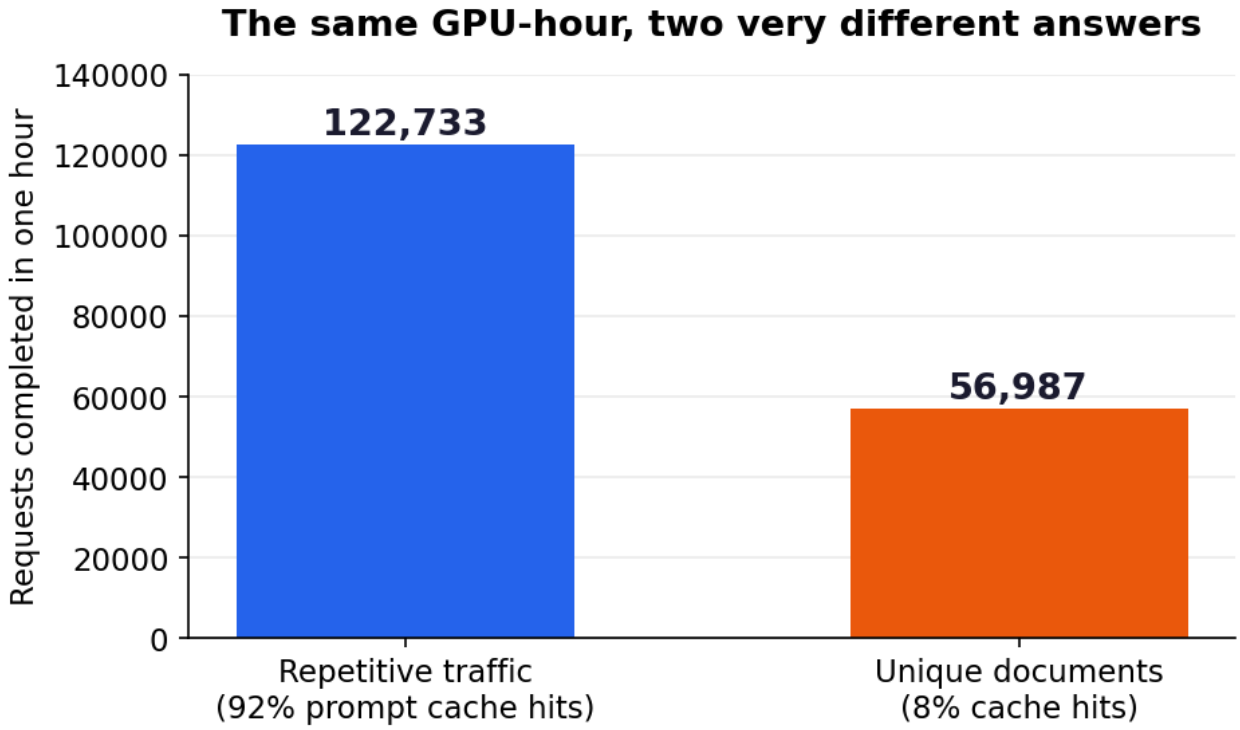
Five experiments. Roughly 345,000 requests. Total spend: about $18.

What we found: whether self-hosting beats the API is not a model question or a platform question. It is a traffic question — how repetitive your prompts are, how heavy

your volume is, and how steady it runs.

The 2.2x nobody puts on a pricing page

Our first run cycled a fixed set of 48 texts for an hour and completed 122,733 requests. Then we reran the identical hour with one change: every request carried a unique document. Throughput fell to 56,987 — a 2.2x drop, on the same GPU, the same model, the same configuration.



Gemma 4 E2B-it, 1×A100-40GB ($2.50/hr), identical configuration — only the traffic changed. 2.2× difference.

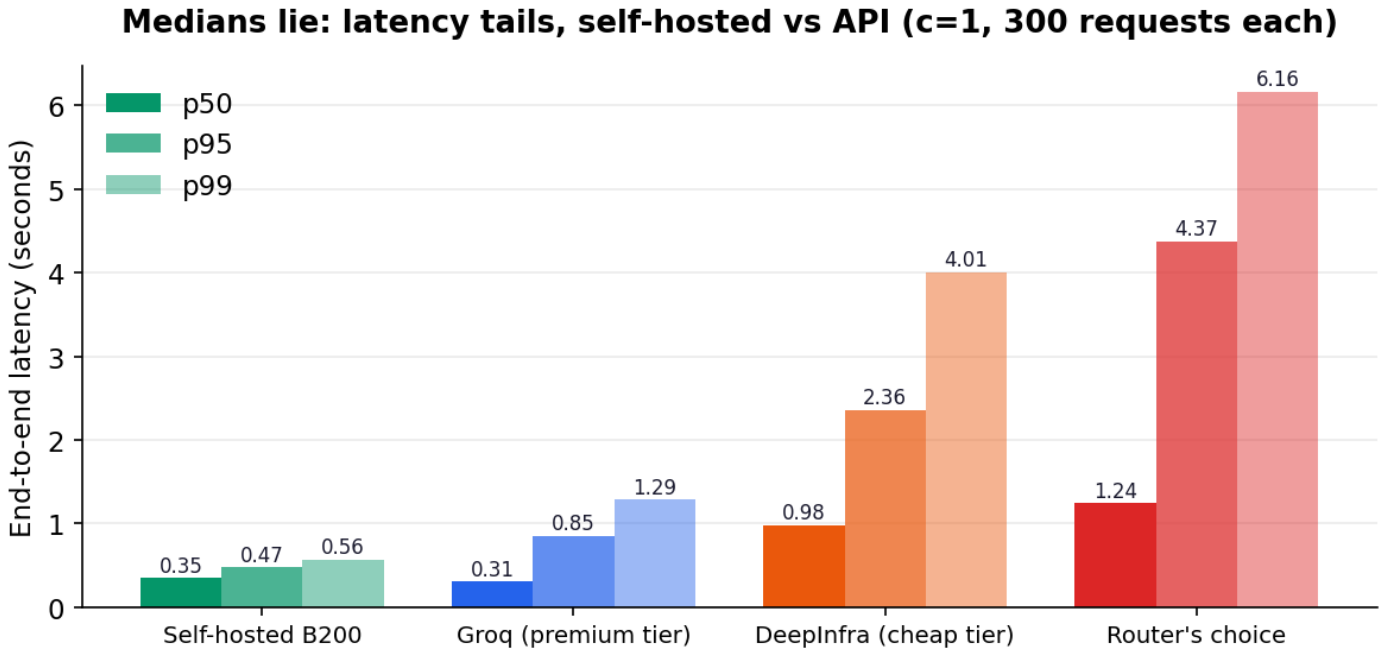
The mechanism is prefix caching. Modern inference engines cache the computation for repeated prompt prefixes; the engine's own metrics showed a 92% cache hit rate in the first run and 8–11% in the second (just the shared system prompt). When your traffic repeats — shared instructions, templated extraction, RAG over a stable corpus — most of your "input tokens" cost the GPU almost nothing.

Here is why that matters commercially: per-token APIs charge you the same for a cached token and a computed one. When you rent the GPU, the caching dividend is yours; when you pay per token, it is your provider's margin. A team whose traffic is 90% repetitive and a team processing unique documents are buying completely different quantities of compute — and only one pricing model lets them see it.

The scale surprise

Intuition says a 50x bigger model costs vastly more to run. Our invoices disagree. The 120B model on the B200 delivered 3.7x the throughput of the 2B on the A100 — 33,000 versus 8,900 tokens per second on unique documents — at 3.6x the hourly price. Cost per million tokens came out flat: $0.076 versus $0.078. Cost per thousand requests fell threefold, to about four and a half cents.

Two factors absorbed the entire scale jump: the model's mixture-of-experts design activates only 5.1B of its 117B parameters per token, and the newer GPU generation brings far more memory bandwidth per dollar. We state this as a measured result for this pairing, not a law — a dense 70B model would tell a different story. But it breaks a reflex worth breaking: do not price a model by its parameter count.



GPT-OSS-120B, same client, same payload, July 2026. Bars per target: p50 / p95 / p99.

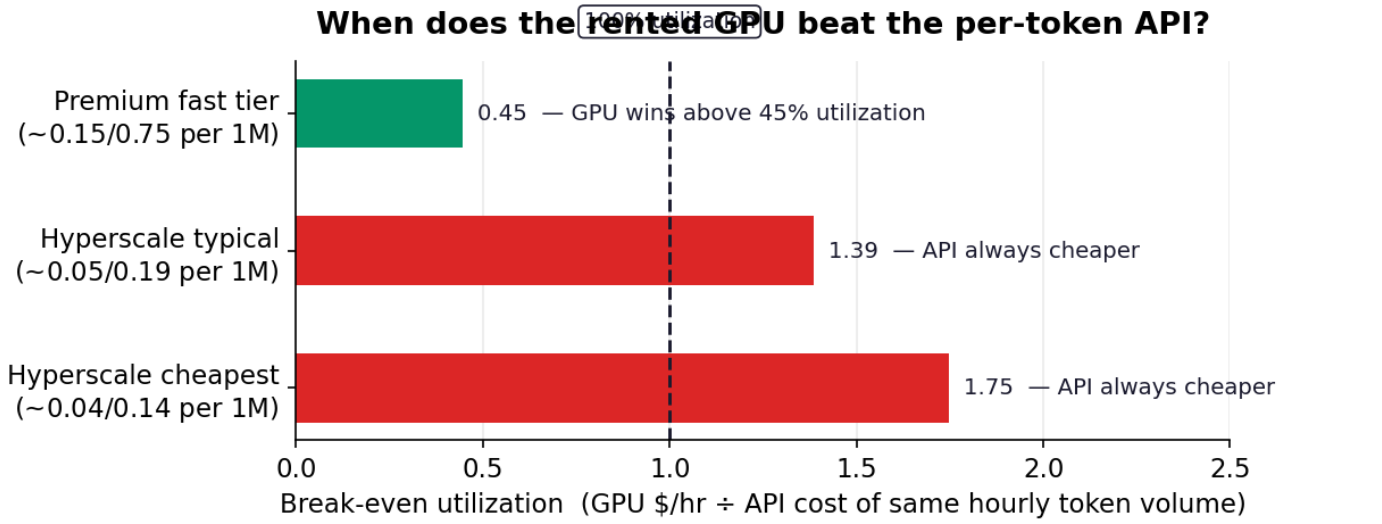
Where we lost, and where the equation flips

Now the result that a vendor-sponsored benchmark would bury. At our best measured throughput, the effective cost of the self-hosted 120B was $0.076 per million tokens. The cheapest hyperscale API listings for the very same model sit near $0.04 in and $0.14–0.18 out — meaning that for unique-document traffic, those providers beat our single GPU even if we kept it saturated 100% of the time. Break-even never arrives. The reason is structural: a provider batching requests from thousands of customers achieves a utilization no single tenant can, and one GPU cannot out-pool a

fleet.

But "the API" is not one price. Listings for this same model spread across roughly an 8x range between hyperscale batchers and premium low-latency tiers. Against the fast tier ($0.15 in / $0.75 out per million), our B200 broke even at 31–45% utilization and won decisively beyond it. And on repetitive traffic — the caching regime — the self-hosted option beat most of the market outright.

So the honest map has three regions. Unique documents at moderate volume: buy tokens from a hyperscale provider. High-volume repetitive traffic, or workloads currently priced at premium API tiers: the hourly GPU wins. Everything else: it depends on one number you can compute in advance — **break-even utilization**: your GPU's hourly price divided by what your hourly token volume would cost at the API. Ours ranged from 0.31 (self-hosting wins easily) to 1.86 (it cannot win).



GPT-OSS-120B on 1×B200 at $9/hr, measured saturation throughput (~119M tokens/hr, 98% input). July 2026 list prices.

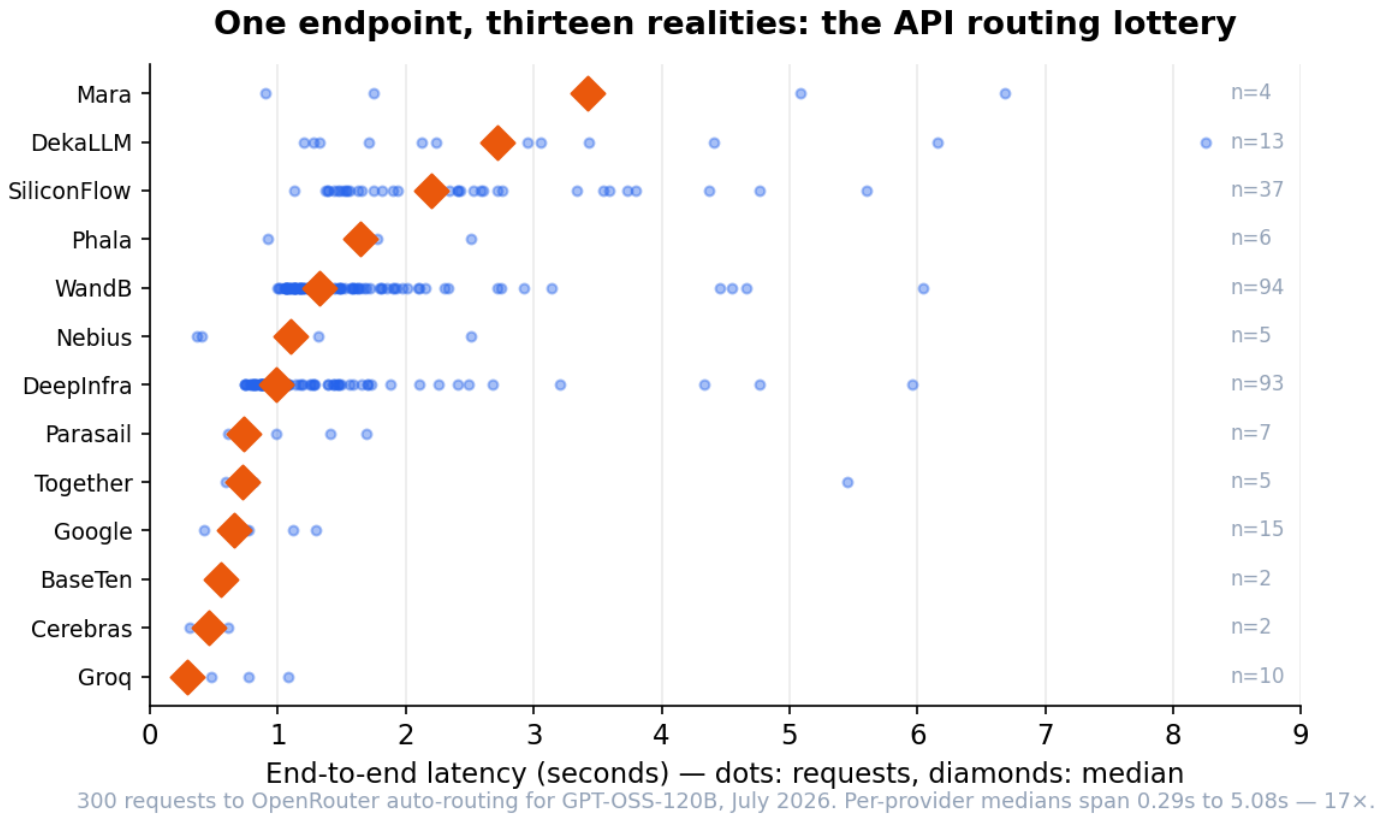
One scope note: our workload was 98% input tokens — classification over documents. Output-heavy workloads like chat or drafting shift the math on both sides, which is exactly why you should run the calculation on your own traffic mix. The prices in this article will age; that division will not.

How fast can one GPU go

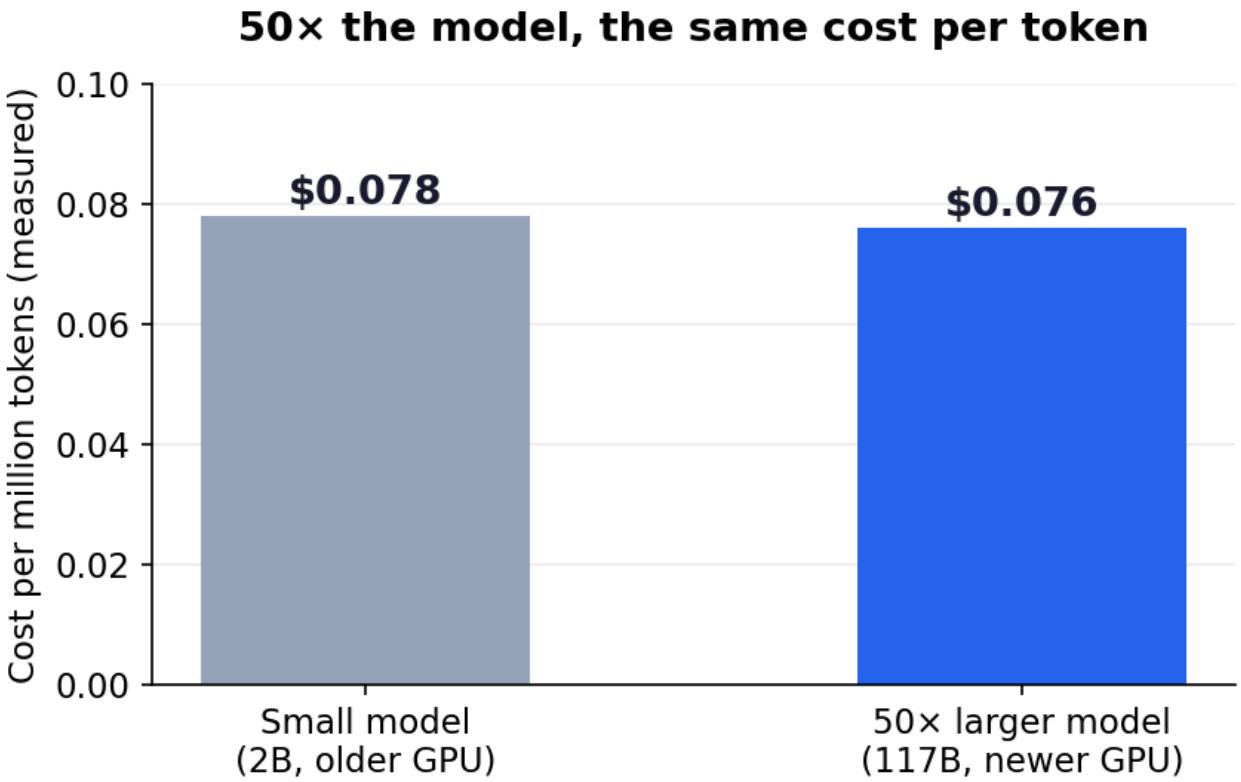
Cost is only half the fork; the other half is speed. So we ran a final side-by-side: 300 identical requests at concurrency 1, from the same machine, against our self-hosted B200 and against the same model on the API market via OpenRouter — once letting the router choose the provider, once pinned to the cheapest tier (DeepInfra), once pinned to the fastest (Groq).

End-to-end, c=1	p50	p95	p99	Failures
Self-hosted B200	0.35s	0.47s	0.57s	0
Groq (premium tier)	0.31s	0.85s	1.29s	2 (rate limits)

End-to-end, c=1	p50	p95	p99	Failures
DeepInfra (cheap tier)	0.98s	2.36s	4.01s	0
Router's choice	1.24s	4.37s	6.16s	7



Three things in that table are worth a contract review. First, the premium tier beat our median by 44 milliseconds — and lost everywhere else: our p95 was almost twice as good, our p99 was 2.3x better, our jitter four times tighter, and we were never rate-limited. Dedicated tenancy does not buy you the fastest median; it buys you a tail that barely exists, and tail latency is what users actually feel. Second, "the API" as most users consume it — the default route — is a lottery: our requests were served by thirteen different providers whose individual medians spread from 0.29 to 5.08 seconds, a 17x range behind a single endpoint.



Measured at maximum sustained load, unique-document traffic, July 2026. Hardware rented per hour.

Third, an accidental finding we consider the most underreported risk in this market: every one of the seven failures on the default route was a malformed structured-output response, and all seven came from one provider — a 32% schema failure rate on the requests it served, against zero malformed responses in our 345,000 self-hosted requests. Structured-output reliability is a provider property, and nobody's pricing page mentions it.

Note the symmetry with the economics: the only API tier that matches a dedicated GPU's speed is the premium one — precisely the tier our break-even said self-hosting beats from 45% utilization. The more your product depends on latency, the less you are competing with the API market's floor, and the better the hourly GPU looks.

What the pricing page doesn't tell you

Three operational facts from our logs, because a field report owes you the friction. Cold starts ranged from four to nine minutes across four boots — a 2x variance — most of it engine initialization and CUDA-graph capture, not weight loading — which makes scale-to-zero unsuitable for interactive workloads without warm-up strategies. Under sustained overload with unique documents, the small model's serving recipe crashed and self-healed, costing eight minutes of availability mid-run; the 120B configuration processed 164,641 requests with a single error. And across all 344,000 requests, strict JSON-schema output never produced one malformed response —

structured output at the engine level is, in our experience, production-grade.

None of this appears in per-token pricing, and all of it appeared in our first three GPU-hours. That asymmetry is the deeper argument for running the test: the hourly model shows you your system's actual behavior — its caches, its ceilings, its failure modes — because you own the serving stack that produces it.

Four questions before your next inference decision

1. What fraction of your prompt tokens repeat across requests — and has anyone measured it?
2. What would one hour of your real token volume cost at your current provider's rates?
3. What is your break-even utilization against the cheapest and the fastest API tier serving your model?
4. Is your latency budget defined at the median or at the 99th percentile — and which tier of the API market actually meets it?

Our entire protocol — polling, phased load, token accounting, the break-even math — is a pair of Python scripts we have open-sourced [repo link]. Running it against your own endpoint costs about the price of lunch.

The API sells you tokens. The GPU sells you an hour. Which one is cheaper is written in your traffic, not on their pricing page.

Benchmarks: Modal Auto Endpoints, July 2026 — Gemma 4 E2B-it (1×A100-40GB, $2.50/hr) and GPT-OSS-120B (1×B200, $9/hr), SGLang recipes, us-west, one container, identical 3-phase protocol. API reference prices from public provider listings, July 2026. Latency comparison: 300 requests per target at concurrency 1 via OpenRouter (auto, DeepInfra-pinned, Groq-pinned), same client and payload; completion-token counts varied by provider (19-35 tokens) as reasoning controls are honored differently. Full per-request data and scripts: [repo].

Revision #8
Created 14 July 2026 16:48:50 by EMB
Updated 10 August 2026 09:57:29 by EMB