The Economics of AI Inference: An Efficiency Curve for Choosing Between Credit Plans, Pay-per-Token APIs, and Self-Hosted Models

Do Tuan Anh
Data & AI Engineering
Hanoi, Vietnam
tuananh071219@gmail.com

Abstract—Enterprises now buy the same resource—large language model inference—through three structurally different pricing models: credit-based subscription plans, pay-per-token APIs, and self-provisioned GPU capacity. FinOps practice and the literature offer little guidance on which mode to occupy at a given usage level, or which metrics reveal when to switch. We survey the literature and industry pricing landscape, then introduce the Efficiency Curve: the cost per token under the cheapest procurement mode at each volume, the lower envelope of three cost functions. Its crossover points—break-even thresholds U_1 and U_2—mark where pay-per-token overtakes credit plans and where self-hosting overtakes pay-per-token. Computing both from a July-2026 snapshot across three quality tiers, we find the first threshold is institutional, not economic: pay-per-token is cheapest only below the seat-fill point $u = F/\bar{p}$ (2–10 Mtok/month across tiers), beyond which the credit plan's idealized asymptotic rate ($0.52/Mtok) undercuts every API rate, so the credit-to-API crossover U_1 never forms and the boundary is instead set by seat saturation and terms of service that deny the subsidized seat rate to programmatic workloads. U_2, by contrast, is a genuine price crossover: priced at the smallest production-deployable self-hosting footprint—the smallest replica that can serve the model, plus an N+1 standby, at SLO-discounted serving throughput (eight GPUs at the frontier)—it ranges from roughly 720 to 1,800 Mtok/month across tiers, is flat in GPU utilization wherever one replica's discounted capacity covers the crossover, and falls as output-token share rises. We derive a monitoring framework in which every recommended metric is justified by the procurement decision it informs.

Index Terms—AI inference economics, LLM cost optimization, FinOps, break-even analysis, enterprise AI, GPU provisioning, token pricing

I. Introduction

Falling per-token prices are usually read as good news for anyone buying AI inference: the same workload costs less every quarter, so cost management can wait until the next price cut does the work. The author's own prior study of this effect finds the opposite dynamic in practice [1]: cheaper tokens induce more usage than they save, so aggregate AI spend restructures and grows rather than shrinking as efficiency improves. If that is right, cost control is not a problem that falling prices eventually solve on their own—it is a positioning problem that recurs every time volume crosses into a new regime, and it recurs faster as usage grows, not slower. An organization that waits for the next price cut to fix its inference bill is solving the wrong problem; the one worth solving is knowing, at any given volume, which of the ways to buy inference is currently cheapest, and when that answer is about to change.

A. The Three-Mode Procurement Choice

That question is harder than it sounds because the market does not offer one pricing structure with moving prices—it offers three structurally different ones, and a buyer's actual bill depends as much on which structure it occupies as on the rate within it. A concrete case makes the choice tangible. A ten-person engineering team standardizes on Claude Max 20x seats for day-to-day coding assistance: $200/month per seat, an idealized ceiling of 129.6 Mtok/month reconstructed from Anthropic's stated usage multiplier [2], consumed entirely through Anthropic's own client. That seat is cheap per token if the ceiling is real, but the terms that grant it do not extend its subsidized rate to automation at scale: the allowance is metered through rolling session windows tied to one individual's account, it cannot be pooled across seats, and the vendor has already moved to re-meter exactly this kind of usage at standard API rates (Section III)—so when the team wants the same model to review every pull request automatically in CI, the credit plan cannot serve that workload at seat economics: the seat serves a developer, not a pipeline. The team moves the automated workload to Claude Opus 4.8 over the API, at $5.00/$25.00 per Mtok input/output—a blended rate of about $9.00/Mtok at a 20%-output-token workload mix [3]—and pays per token with no cap and no seat to manage. As the pipeline's volume grows, a third option comes into view: self-hosting an open-weight model on reserved GPUs. A 70B open-weight model that can credibly stand in for the frontier tier must be sharded across four reserved H100-80GB GPUs—

a fixed $7,191.96/month regardless of how many tokens it serves—and past roughly 800 Mtok/month, the frontier-tier crossover this paper computes in Section IV, that fixed cost undercuts paying Anthropic per token for the share of the workload the open-weight model can serve. Three procurement modes, three different cost shapes, and the team has now touched all three without anyone deciding, in advance, which one the workload should occupy at which volume.

Existing tooling does not answer that question. FinOps dashboards reconcile spend within whichever mode an organization already occupies; serving-efficiency research and model routers optimize within pay-per-token pricing specifically; a nascent inference-economics literature models provider-side prices and cost frontiers [4], [5], [6], and industry calculators compare API against self-hosted cost pairwise; but no published analysis puts all three structures—a credit-plan seat, an API rate card, and a GPU rental rate—into the same model and asks where they cross (Sections II and III). Practitioners are left to guess, seat by seat and workload by workload, at a decision this paper turns into an equation.

B. Contributions

This paper makes four contributions.

- A survey of the academic literature on inference economics—serving efficiency, cost-aware model selection, cloud FinOps, and the efficiency-demand rebound effect—and of the industry pricing landscape those literatures do not connect, establishing that no existing work compares credit plans, pay-per-token APIs, and self-hosted capacity as a single procurement decision (Sections II and III).
- A formal three-mode cost model that expresses credit-plan, pay-per-token, and self-hosted monthly cost as functions of a single common variable, monthly inference volume, so the three otherwise incompatible pricing units—a flat seat fee, a per-token rate, and an hourly capacity charge—can be compared directly (Section IV).
- The Efficiency Curve—the lower envelope of the three cost functions—with its break-even volume thresholds U_1 and U_2 computed empirically from a July-2026 pricing snapshot across three quality tiers, and a sensitivity analysis over GPU utilization and output-token share, including the finding that the first threshold is institutional rather than economic: pay-per-token is the cheapest mode only below the seat-fill point $u = F/\bar{p}$, beyond which the idealized credit-plan rate undercuts every API rate, so the credit-to-API crossover U_1 never forms and the practical boundary is instead set by seat saturation and by terms that deny the subsidized seat rate to programmatic workloads (Section IV).
- A decision-linked monitoring framework and switching rules derived directly from that model, in which every recommended metric—and every threshold at which it should trigger a procurement decision—is justified by the specific choice it informs, rather than assembled from whatever a billing API happens to expose (Sections V and VI).

C. Roadmap

Section II surveys the academic literature and shows that none of its five tracks addresses the procurement-mode choice directly. Section III surveys the July-2026 industry pricing landscape across all three modes and states the resulting gap formally. Section IV builds the three-mode cost model, defines the Efficiency Curve, and computes U_1 and U_2 along with their sensitivity to utilization and token mix. Section V derives the monitoring framework those thresholds imply. Section VI separates within-mode optimization from mode switching, states switching rules as metric inequalities, and quantifies the real-world frictions that shift U_2 in practice. Section VII states the paper's limitations, and the Conclusion restates what the Efficiency Curve buys a practitioner that a price alone does not.

II. Related Work: The Economics of LLM Inference

The literature on large language model (LLM) inference has grown along several separate tracks: making a model cheaper to serve, choosing which model to call for a given query, managing cloud spend, modeling the economics of the inference market itself, and explaining why efficiency gains do not translate into lower total cost. Each track answers a real question. None of them addresses the question this paper takes up: given a fixed workload, which pricing structure—credit plan, pay-per-token API, or self-hosted GPU capacity—is cheapest, and at what usage level the answer changes.

A. Serving Efficiency Techniques

A substantial body of systems research reduces the cost of serving a given model on fixed hardware. PagedAttention [7] manages the key-value cache as paged virtual memory, eliminating fragmentation and raising achievable batch size; Orca [8] introduces iteration-level scheduling so requests of different lengths share a batch without head-of-line blocking; SGLang [9] adds RadixAttention to reuse KV-cache state across requests that share a prefix, and Prompt Cache [10] generalizes prefix reuse across sessions. Quantization methods such as GPTQ [11] and AWQ [12] cut memory footprint and, with it, the number of GPUs a given throughput requires; speculative decoding [13] trades a small draft model's forward passes for fewer large-model steps; knowledge distillation [14] produces smaller models that approximate a larger one's outputs at a fraction of the inference cost. Surveys of the full serving stack [15], [16] catalog these techniques together and quantify where each intervenes on the compute-memory-bandwidth roofline.

Every technique in this literature shifts a cost curve downward: it lowers the dollars or GPU-seconds needed to serve a fixed workload. None of it changes the structure of what is being bought. A quantized, cached, speculatively decoded model still has to be paid for through a credit-plan seat, a metered API call, or a self-hosted GPU-month, and this literature is silent on which of the three a buyer should choose or when to switch.

B. Cost-Aware Model Selection

A second strand treats model choice itself as the cost lever. FrugalGPT [17] cascades queries through progressively more expensive models, escalating only when a cheaper model's answer looks unreliable, and reports large API-spend reductions at near-constant quality. RouteLLM [18] learns a router from human preference data that sends each query to the cheapest model expected to satisfy it, again holding quality roughly fixed.

Both systems optimize within pay-per-token pricing: the decision variable is which vendor and model to call at that vendor's posted per-token rate, and both assume the buyer is already transacting through metered APIs. Neither compares a cascade's total cost against a credit-plan seat or a self-hosted deployment of the same models—the procurement mode is a fixed premise, not a variable the routing decision reasons about.

C. Cloud FinOps and Cost Management

Cloud cost governance is its own established field. Reviews of cloud and AI infrastructure cost optimization [19] and tooling built to automate it [20], [21] formalize practices such as cost allocation, rightsizing, and reserved-versus-on-demand purchasing; game-theoretic work on spot-instance markets [22] models the reserved-versus-spot tradeoff as a two-stage pricing game. The FinOps Foundation's practitioner framework [23] is the industry's closest thing to a standard vocabulary for cloud cost accountability.

This literature predates token-metered AI pricing and was built around infrastructure billed by the hour or the reservation term. None of it has a construct for a credit-based subscription seat with a usage cap, and none models a crossover between three qualitatively different pricing structures—subscription, metered API, and owned capacity—the way inference procurement requires; the reserved-versus-spot game is a single-mode problem restated with two prices, not three modes with three different cost shapes.

D. Inference-Market Economics

The track closest to this paper models the economics of the inference market itself. Zhuang et al. [4] construct an inference production frontier and analyze providers' marginal cost of serving tokens; Du [5] traces per-token price evolution and market competition across quality tiers of inference services; and Patil [6] argues that per-token prices alone mis-estimate infrastructure cost and derives a concurrency-aware costing methodology for self-hosted deployments. All three take the provider's or the market's perspective on a single pricing structure at a time: they explain how per-token prices form, evolve, or mislead, but none places the three procurement structures a buyer faces—credit seat, metered API, owned capacity—into one model, and none includes the credit-plan mode at all. They are the supply-side complement to the demand-side, procurement-facing question asked here.

E. Efficiency and Demand

A separate line of work asks whether efficiency gains reduce total resource consumption at all. Jevons's original observation that more efficient steam engines increased, rather than decreased, coal consumption [24] has been revisited and largely confirmed in modern energy economics [25], [26], and applied directly to AI's environmental footprint [27], whose energy and carbon costs a parallel measurement literature quantifies [28], [29], [30], [31]. The author's own prior work [1] extends this to AI infrastructure spend specifically: falling per-token cost does not shrink total AI expenditure, because falling cost induces new usage that more than replaces the savings, so spend restructures and grows rather than declining.

If efficiency gains do not shrink spend, the question this paper asks—which procurement mode is cheapest at a given usage level—is not a one-time calculation to make and forget. Usage volume keeps climbing, so organizations keep crossing the thresholds between modes. The rebound-effect literature explains why volume grows; it says nothing about how to buy inference at whatever volume it reaches.

Taken together, these five literatures leave the procurement-mode boundary unaddressed: serving research optimizes cost within a mode, cascades and routers optimize within pay-per-token specifically, FinOps has no vocabulary for credit plans, inference-market economics models the supply side of one structure at a time, and rebound-effect research explains why the question never goes away without saying how to answer it. Section III surveys the industry offerings that actually populate those three modes and states this gap formally as the basis for the framework introduced in Section IV.

III. The Industry Pricing Landscape

Section II showed that the academic literature treats serving cost, model selection, cloud spend, and rebound demand as separate problems and never asks which pricing structure a buyer should occupy. That question is not hypothetical: the market already offers three structurally different ways to buy the same underlying resource, and every enterprise procuring LLM inference has, whether it realizes it or not, already picked one. This section surveys the offerings that populate each structure, using a July 2026 pricing snapshot [2], [3], [32], [33], [34], [35], and shows that no existing tool compares across them.

A. Three Procurement Modes

a) Credit-based plans.: A credit-based plan charges a flat, usually monthly, fee for a seat and grants that seat access up to an opaque usage ceiling: a rolling rate-limit window, a message count, or a dollar-denominated credit pool converted to model usage at the vendor's internal rate. Anthropic's Claude Pro ($20/month) and Claude Max ($100 or $200/month, marketed as 5x and 20x Pro's allowance) [2], OpenAI's ChatGPT Plus and Pro ($20 and $200/month), and coding assistants such as GitHub Copilot ($10–$100/month across Pro, Pro+, Business, Max, and Enterprise seats) [34] and Cursor ($20–$200/month) all follow this structure. Per-seat pricing is common for team and enterprise tiers (Copilot Business, ChatGPT Business), and none of these vendors publishes a fixed token count for its cap; the caps used in this paper are idealized ceilings reconstructed from each vendor's stated usage multipliers and credit-to-dollar allowances under an assumption of continuous full-window use, converting message limits at an assumed 1,000 tokens per message (500 input, 500 output)—a constant every message-derived cap in Table I scales linearly in, and whose halving is equivalent to the 50%-cap row of the credit-cap sensitivity sweep in Section IV. The constraint that matters most for the rest of this paper is contractual rather than technical: none of the fifteen credit-plan offerings in the pricing snapshot extends its subsidized seat rate to programmatic workloads. Eleven of the fifteen—OpenAI's ChatGPT plans, GitHub Copilot's seats, and Cursor's plans—restrict usage to interactive consumption through the vendor's own client or IDE surface, and OpenAI's terms of use additionally prohibit automated or programmatic extraction of the service's output [36]; the API is a separate product with separate billing. The remaining four, Anthropic's plans, do permit programmatic surfaces (Claude Code, the Agent SDK, third-party apps), but that usage draws down the same session-windowed allowance tied to one individual's account, cannot be pooled across seats, and is the subject of an announced—and, as of this snapshot, paused—change that meters Agent SDK usage through a separate monthly credit pool billed at standard API rates once exhausted [37]. A credit plan is a seat, not an integration point, and this constraint binds Section IV's analysis rather than footnoting it: it is precisely why a workload that would be cheapest under a credit plan's idealized per-token rate cannot actually ride that rate at programmatic scale.

b) Pay-per-token APIs.: A pay-per-token API charges linearly per million tokens (Mtok) processed, with no monthly floor and no usage ceiling. Prices are asymmetric between input and output—across this snapshot's offerings, output tokens cost between 1x and roughly 8x input tokens, with 5–6x typical of the first-party vendors, reflecting the extra compute of autoregressive generation—and most vendors discount repeated input via prompt caching. Anthropic's Claude Developer Platform prices Opus 4.8 at $5.00/$25.00 per Mtok input/output with $0.50/Mtok cached input, Sonnet 4.6 at $3.00/$15.00, and Haiku 4.5 at $1.00/$5.00 [3]; OpenAI prices GPT-5.6 Sol (frontier) at $5.00/$30.00 and GPT-5.6 Terra (mid) at $2.50/$15.00 per Mtok [32]. Third-party hosts undercut both, chiefly by serving open-weight models: Together AI serves DeepSeek V4 Pro at $1.74/$3.48, GLM-5.2 at $1.40/$4.40 (the open-tier row from which Section IV computes its open-tier threshold), Llama 3.3 70B Instruct at a symmetric $1.04/$1.04, and OpenAI's own Apache-licensed gpt-oss-120b at $0.15/$0.60 per Mtok [33], roughly a third of the frontier-tier rate or less; the same hosts also resell some closed-weight models at comparable rates (e.g. Alibaba's API-only Qwen3.7-Max at $1.25/$3.75 [33], [38]). Unlike a credit plan, an API key is a programmatic interface by design—this is the mode every routing and cascading system surveyed in Section II assumes—and cost scales without bound as volume grows.

c) Self-provisioned capacity.: Self-provisioned capacity charges for the infrastructure rather than the tokens: a fixed hourly (or reserved-term) rate that the buyer pays regardless of how many tokens it actually serves, and a marginal cost per additional token that falls toward zero as utilization rises. Renting an H100-80GB GPU on demand costs $6.88/hour from a hyperscaler (AWS p5, normalized per GPU) [39] versus $2.99/hour from a specialized GPU cloud (RunPod) [35] or $4.29/hour from another (Lambda) [40]; committing to a one-year term cuts the hyperscaler rate to $2.463/hour, a 64% discount for accepting the commitment. A hybrid sits between self-hosting and the API: managed provisioned throughput, such as Azure OpenAI's Provisioned Throughput Units, sells a fixed tokens-per-minute capacity budget for a flat hourly rate ($40/hour for a GPT-4o PTU, a secondary-sourced indicative figure; see Table I) delivered through a hosted API rather than a rented GPU—priced and billed like owned capacity, consumed like a metered service. All three self-provisioned variants share the defining property that pay-per-token APIs and, functionally, credit plans lack: cost is fixed against capacity, not against consumption, so the effective cost per token depends entirely on how much of that capacity gets used.

B. Quality Tiers and the Limits of Self-Hosting

The rest of this paper groups API offerings into three quality tiers, and the grouping encodes more than a price band: it encodes whose weights a buyer can obtain, and therefore which procurement modes are available for a given model at all.

Frontier denotes the vendors' flagship proprietary models: in the July 2026 snapshot, Claude Opus 4.8 ($5.00/$25.00 per Mtok input/output) [3], GPT-5.6 Sol ($5.00/$30.00) [32], and Gemini 3.1 Pro ($2.00/$12.00 at up to 200k context) [41]. Mid denotes the same vendors' smaller, cheaper proprietary models: Claude Sonnet 4.6

($3.00/$15.00) and Claude Haiku 4.5 ($1.00/$5.00) [3], GPT-5.6 Terra ($2.50/$15.00) [32], and Gemini 2.5 Flash ($0.30/$2.50) [41]. Every model in both tiers is sold exclusively as a hosted service: none of these vendors distributes its weights, so a frontier or mid-tier model cannot be self-hosted at any price. Open-weight denotes models whose weights are published for download, and it is the only tier in which the third procurement mode exists. Its licensing is not uniform. Several current open-weight models ship under genuinely permissive terms—MIT for DeepSeek V4 Pro (a 1.6T-parameter MoE with 49B active parameters) [42] and for GLM-5.2 (753B MoE) [43], Apache-2.0 for OpenAI's gpt-oss-120b (117B MoE, 5.1B active) [44], for the Qwen 3.5/3.6 families [38], and for Mistral Small 4 (119B MoE) [45]—while Meta's Llama family uses a custom community license whose commercial grant is capped at 700 million monthly active users and which incorporates an acceptable-use policy, terms permissive enough for almost every enterprise but not open-source in the OSI sense [46]. Openness is a property of the model, not the vendor: OpenAI publishes gpt-oss weights while keeping GPT-5.6 closed, and Alibaba publishes Qwen 3.5/3.6 weights while keeping its frontier Qwen3.7-Max API-only [38].

Two consequences run through the rest of the paper. First, every "self-hosted" scenario in Section IV, including the one that stands in for frontier quality, is necessarily a comparison between a proprietary API and the best available open-weight substitute, because no frontier weights exist to self-host. Second, one snapshot row is mislabeled by this tier standard: Qwen3.7-Max is carried among the open tier's API rows, but it is a closed-weight model resold through the same host. We flag rather than reclassify it, because the open tier's median offering—GLM-5.2, the row from which Section IV's open-tier U_2 is computed—is the median with or without that row, so no computed threshold depends on the label.

C. A Representative Cross-Section

Table I lists twelve representative offerings spanning the three modes, drawn from the full pricing snapshot. "Effective unit" for credit plans is the vendor's headline fee divided by this paper's idealized monthly token ceiling for that plan (full-window saturation). The mark * flags these ceilings as upper bounds on value, not typical outcomes: no vendor publishes a fixed token count, and the ceiling assumes continuous full-window use all month. The mark † flags figures that rely on secondary rather than vendor-confirmed sources—OpenAI's consumer-plan message limits and the Azure PTU hourly rate.

D. Observability and FinOps Tooling

Cost-observability tooling has grown alongside cloud spend generally, and LLM-specific instrumentation has followed the same pattern: dashboards and cost-allocation platforms [19], [20], agent-based cost-optimization tooling [21], and reserved-versus-on-demand purchasing models [22] all instrument spend within whichever mode an organization already occupies. A token-metering dashboard reconciles API usage against a vendor's per-token bill; a cloud cost tool tracks GPU utilization against a reserved-instance commitment; the FinOps Foundation's own practitioner framework [23] formalizes allocation and rightsizing practices for exactly this kind of within-mode monitoring. None of this tooling ingests a credit-plan seat, an API rate card, and a GPU rental rate as three inputs to the same comparison. That is a structural gap, not an oversight: the three cost structures are priced in incompatible units—a flat fee, a per-token rate, an hourly capacity charge—and none of the observability tools surveyed here converts between them to tell a buyer which mode is currently cheapest for its workload, let alone when that answer will change.

The gap. Pairwise comparisons exist: industry calculators routinely weigh API against self-hosted cost, and the inference-economics literature of Section II models how per-token prices form and evolve [4], [5], [6]. But no published analysis we are aware of (i) formalizes all three cost structures surveyed above—credit plan, pay-per-token API, self-provisioned capacity—in a single model, (ii) computes where they cross for a given workload, and (iii) derives which metrics an enterprise must monitor to know its position relative to those crossovers. Section IV builds that model and computes the crossovers; Section V derives the metrics from it.

IV. The Cost Model and the Efficiency Curve

Section III established that the three procurement modes are priced in incompatible units—a flat seat fee, a per-token rate, and an hourly capacity charge—and that no existing tool converts between them. This section builds that conversion. We write each mode's monthly cost as a function of a single common variable, monthly inference volume, take the lower envelope of the three functions to define the Efficiency Curve, and locate the volumes at which the cheapest mode changes. We then compute those crossover volumes from the July 2026 pricing snapshot and analyze their sensitivity. The headline result is not a single number but a structure: one of the two crossovers we set out to find does not exist under idealized assumptions, and understanding why it does not is the section's central contribution.

A. The Three Cost Functions

Let u denote monthly inference volume in millions of tokens (Mtok/month), the common variable against which all three modes are priced. Each mode m has a monthly cost function $C_m(u)$ in USD/month.

TABLE I
Representative offerings across the three procurement modes, July 2026 pricing snapshot. API headline prices are input/output per Mtok. Credit-plan effective units are idealized full-utilization ceilings, not typical usage; see text.

Mode	Offering	Headline price	Effective unit	Constraint
Credit	Claude Pro (Anthropic)	\$20/mo	\$3.09/Mtok*	Session rate limit; programmatic use draws the same per-account allowance
Credit	Claude Max 20x (Anthropic)	\$200/mo	\$1.54/Mtok*	Session rate limit (20x Pro, relative only); programmatic use draws the same per-account allowance
Credit	ChatGPT Plus (OpenAI)	\$20/mo	\$0.52/Mtok*†	160 msgs/3-hr window; no programmatic use
Credit	Copilot Pro+ (GitHub)	\$39/mo	\$3.34/Mtok*	7,000 AI credits/mo; IDE-bound, no raw API access
API	Claude Opus 4.8 (Anthropic)	\$5.00/\$25.00 per Mtok	Linear	None; cached input \$0.50/Mtok
API	Claude Haiku 4.5 (Anthropic)	\$1.00/\$5.00 per Mtok	Linear	None; cached input \$0.10/Mtok
API	GPT-5.6 Sol (OpenAI)	\$5.00/\$30.00 per Mtok	Linear	None; cached input \$0.50/Mtok
API	DeepSeek V4 Pro (Together AI)	\$1.74/\$3.48 per Mtok	Linear	None; open-weight (MIT), third-party hosted
API	GLM-5.2 (Together AI)	\$1.40/\$4.40 per Mtok	Linear	None; open-weight (MIT), third-party hosted; open-tier median in Section IV
Self-hosted	H100-80GB on-demand (RunPod)	\$2.99/hr	\$2,183/mo, fixed	None; hourly, no commitment
Self-hosted	H100-80GB 1-yr reserved (AWS)	\$2.463/hr	\$1,798/mo, fixed	1-year commitment term
Provisioned	GPT-4o PTU (Azure OpenAI, hybrid)	\$40.00/hr†	\$29,200/mo, fixed	Fixed tokens/min budget; capacity commitment

a) Credit plans.: A credit plan sells a seat for a flat fee F that grants an effective monthly cap of q Mtok. Serving a volume u requires enough seats to cover it, so

$$C_{\text{credit}}(u) = \lceil u/q \rceil \cdot F, \qquad \text{(at least one seat)}, \quad (1)$$

a step function that rises by F each time volume crosses a multiple of q. The ceiling encodes the discreteness of seats. Two caveats are built into this form and must be stated plainly. First, the seat-scaling in Eq. (1) treats $n = \lceil u/q \rceil$ seats as delivering $n \cdot q$ of usable capacity, which assumes a seat's full cap q is extractable and that capacity composes linearly across seats. Second, and more binding, no credit plan in the pricing snapshot extends its subsidized seat rate to programmatic workloads: most restrict usage to interactive consumption through the vendor's own client or IDE, and the remainder meter programmatic surfaces through the same session-windowed, per-account allowance (Section III). Eq. (1) therefore describes an idealized instrument: it is an upper bound on a credit plan's applicability, the cost a buyer would pay if the seat cap were fully and legally extractable at scale. We carry both readings—idealized and constrained—through the analysis rather than collapsing them.

b) Pay-per-token APIs.: An API charges linearly per token with no floor and no ceiling:

$$C_{\text{api}}(u) = u \cdot \bar{p}(\rho, h), \quad (2)$$

where $\bar{p}$ is the blended rate per Mtok. Because input and output tokens are priced asymmetrically and repeated input can be cache-discounted, the blended rate depends on the output-token share ρ and the cache-hit fraction h:

$$\bar{p}(\rho, h) = (1-\rho)\big[(1-h)\, p_{\text{in}} + h\, p_{\text{cache}}\big] + \rho\, p_{\text{out}}, \quad (3)$$

with p_{in} the uncached input rate, p_{cache} the cache-hit input rate, and p_{out} the output rate (all in USD/Mtok). The inner bracket blends cached and uncached input; the outer expression blends input against output.

c) Self-provisioned capacity.: Self-hosting rents whole replicas—the smallest GPU set that can serve the model—at a fixed monthly cost G per replica. A replica benchmarked at sustained throughput T (tok/s), run at utilization η over S seconds per month, serves a monthly capacity of

$$\kappa = \frac{T \cdot \gamma \cdot \eta \cdot S}{10^6} \ \text{Mtok/month}, \quad (4)$$

where the 10^6 converts tokens to Mtok, $S = 2{,}628{,}000$ s (730 hours), and $\gamma \in (0, 1]$ is an SLO goodput factor: published serving benchmarks are measured at throughput-optimal, batch-heavy operating points, and the goodput deliverable under interactive latency targets (time-to-first-token and inter-token bounds) is a fraction of that peak. Serving volume u requires $\lceil u/\kappa \rceil$ replicas; a production endpoint additionally holds r standby replicas, because a single-replica deployment turns any host or GPU failure into a total outage. Finally, enterprises rarely pay list price for infrastructure: a negotiated agreement (an AWS enterprise discount program on EC2, a committed-spend contract at a neocloud) takes a fraction d off the rental rate. So

$$C_{\text{self}}(u) = (\lceil u/\kappa \rceil + r)\cdot(1-d)\cdot G, \qquad \text{(at least } 1+r \text{ replicas)}, \quad (5)$$

another step function, rising by $(1-d)\,G$ each time volume crosses a multiple of the per-replica capacity κ. The base case sets $\gamma = 0.5$ (benchmark throughput discounted twofold for interactive SLOs, an assumption the sensitivity

analysis sweeps), $r = 1$ (N+1 redundancy, the minimum serviceable configuration), and $d = 0$ (list price, since a negotiated discount cannot be assumed; the sensitivity analysis sweeps d at typical enterprise levels). Setting $\gamma = 1, r = 0, d = 0$ recovers the idealized single-replica, benchmark-face-value model.

B. The Efficiency Curve and Its Crossovers

For any volume, a rational buyer occupies whichever mode costs least. The effective cost per token under that choice, as a function of volume, is the Efficiency Curve:

$$E(u) = \min_{m \in \{\text{credit, api, self}\}} \frac{C_m(u)}{u}. \tag{6}$$

$E(u)$ is the lower envelope of the three per-token cost curves. The mode that attains the minimum in Eq. (6) changes as volume grows, and the volumes at which it changes are the break-even volume thresholds. Ordering the modes by the volume regime in which each is conventionally expected to be cheapest—credit plans cheapest at low volume where a flat seat fee is assumed to beat per-token charges, pay-per-token in the middle, and self-hosting cheapest at high volume once fixed GPU cost amortizes—we name the two thresholds:

- U_1: the crossover where pay-per-token overtakes the credit plan (the smallest volume at which $C_{\text{api}} \leq C_{\text{credit}}$ and stays so);
- U_2: the crossover where self-hosting overtakes pay-per-token (the smallest volume at which $C_{\text{self}} \leq C_{\text{api}}$ and stays so).

Whether either threshold exists is an empirical question about the particular prices, not a given: a crossover exists only if the mode that is supposed to take over actually becomes cheaper somewhere in the volume range.

C. Computed Thresholds

We instantiate the model with the July 2026 snapshot. For each of three quality tiers—open-weight, mid, and frontier, as defined in Section III-B—the credit plan is the cheapest seat per Mtok of cap (ChatGPT Plus, \$20/month, idealized cap $q = 38.4$ Mtok); the API is the median-priced offering in that tier by uncached input price, taking the upper median when the tier holds an even number of offerings—a tie-breaking rule that matters and is therefore stated: in the four-offering mid tier it selects GPT-5.6 Terra ($\bar{p} = \$5.00$) over Claude Haiku 4.5 ($\bar{p} = \$1.80$), and taking the lower median instead would nearly triple the mid-tier U_2 to $\approx 1{,}998$ Mtok/month. A second, subtler tie-break sits in the frontier tier, where the median input price (\$5.00/Mtok) is shared by Claude Opus 4.8 ($\bar{p} = \$9.00$) and GPT-5.6 Sol ($\bar{p} = \$10.00$); the selection resolves to Opus 4.8 by dataset row order, and choosing Sol instead would lower the frontier U_2 by about 10%. And self-hosting pairs the cheapest non-provisioned GPU (AWS H100-80GB, one-year reserved, \$1,797.99/GPU-month) with a tier-matched open-weight serving benchmark [47], priced at the granularity of a whole deployable replica. A model sharded across k GPUs cannot serve on fewer than k, so its indivisible fixed cost G is the cost of all k GPUs, not one. Because no frontier-tier weights are distributed (Section III-B), frontier-quality self-hosting is necessarily an open-weight stand-in: it is proxied by Llama 3.1 70B served in 8-bit floating point at tensor-parallel degree four (FP8, TP4), which is sharded across four H100s and sustains 6,511.88 tok/s in aggregate [47]; its replica atom is therefore four GPUs at $G = \$7{,}191.96$/month. Open and mid tiers are proxied by Llama 3.1 8B (FP8), a single-GPU (TP1) deployment at 6,514.66 tok/s, whose replica atom is one GPU at $G = \$1{,}797.99$/month. The base case fixes utilization $\eta = 0.5$, output share $\rho = 0.2$, cache hit $h = 0$, SLO goodput $\gamma = 0.5$, redundancy $r = 1$, and infrastructure discount $d = 0$ (list price), so the self-hosting entry floor is two replicas of the tier's replica atom: \$3,595.98/month (open, mid) and \$14,383.92/month (frontier). Figure 1 plots the three per-token cost curves and their lower envelope for each tier; Table II reports the thresholds.

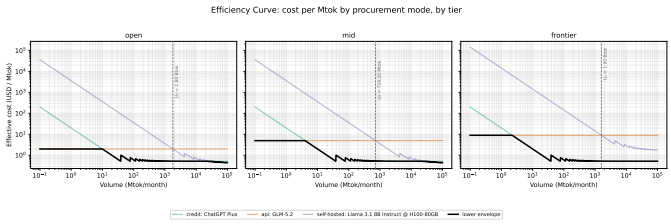


Fig. 1. The Efficiency Curve (black lower envelope) over the three per-token cost curves—credit, pay-per-token API, and self-hosted—for each quality tier, on log–log axes. The idealized envelope has three segments: pay-per-token rides the envelope at low volume (below $u = F/\bar{p}$, i.e. 10.0/4.0/2.2 Mtok for open/mid/frontier), the credit plan takes over from there to U_2 as its flat fee amortizes toward its asymptotic \$0.52/Mtok rate, and self-hosting is cheapest beyond U_2. No U_1 is marked because the credit-to-API crossover never forms (see text); the marked threshold is U_2, where self-hosting overtakes pay-per-token.

The headline finding is the empty U_1 column, and reading it correctly means tracing the idealized envelope segment by segment. The cheapest credit plan bills \$20/month for one seat—a flat fee F whose effective rate F/u is high at low volume (\$20/Mtok at $u = 1$) and falls only as the seat fills, reaching its floor $F/q = \$20/38.4 = \$0.52$/Mtok at seat saturation ($u = q = 38.4$ Mtok). The API, by contrast, charges a constant blended rate $\bar{p}$. At low volume the flat seat fee is the more expensive way to buy a handful of Mtok, so pay-per-token is the cheapest mode until its linear bill catches the flat fee, at

$$u = F/\bar{p} = \$20/\bar{p}, \tag{7}$$

which is 10.0 Mtok for the open tier ($\bar{p} = \$2.00$), 4.0 Mtok for mid (\$5.00), and 2.2 Mtok for frontier (\$9.00). The idealized envelope therefore has three segments in every tier: pay-per-token cheapest below $u = F/\bar{p}$; the credit

TABLE II

Break-even volume thresholds by tier, base case ($\eta = 0.5$, $\rho = 0.2$, $h = 0$, $\gamma = 0.5$, $r = 1$, $d = 0$), from break_even_results.json. Volumes in Mtok/month. $\bar{p}$ is the blended API rate from Eq. (3). Credit plan is ChatGPT Plus ($20/mo, cap 38.4 Mtok); GPU is AWS H100-80GB 1-yr reserved at $1,797.99/mo. The self-hosting floor is the smallest deployable replica plus one N+1 standby: two GPUs ($3,595.98/mo) for the 8B open/mid proxy, eight GPUs ($14,383.92/mo) for the 70B (TP4) frontier proxy. Thresholds are price-only; the operational frictions of Section VI-C (ops headcount, migration, compliance) move the volume at which switching is actually worth executing further right.

Tier	Median API (vendor)	$\bar{p}$ ($/Mtok)	U_1[a]	U_2 (Mtok/mo)
Open	GLM-5.2 (Together)	2.00	—	1,797.99
Mid	GPT-5.6 Terra (OpenAI)	5.00	—	719.20
Frontier	Claude Opus 4.8 (Anthropic)	9.00	—	1,598.21

[a] U_1 (as defined: pay-per-token overtaking credit as volume grows) does not exist against the snapshot's cheapest seat per Mtok of cap. The API is cheaper only below $u = F/\bar{p}$; beyond it the credit plan's asymptotic rate ($20/38.4 = $0.52/Mtok) undercuts every API rate ($\geq$ $2.00/Mtok) and, under idealized seat-scaling, stays lower, so no later volume restores pay-per-token to the envelope. With a costlier seat the conclusion reverses: at Claude Pro's $3.09/Mtok floor (Table I) the credit segment never enters the envelope at all. See text.

plan cheapest from there up to U_2 (where its amortizing fee, tending to $0.52/Mtok, stays below every API rate); and self-hosting cheapest beyond U_2—the ordering api $\rightarrow$ credit $\rightarrow$ self-hosted.

Given that structure, why is the U_1 column empty? Because U_1 is defined as the volume at which pay-per-token overtakes the credit plan and stays cheaper—credit at the bottom, the API taking over as volume grows. The real crossover at $u = F/\bar{p}$ runs the other way: the credit plan overtakes the API, and under the linear seat-scaling of Eq. (1) its asymptotic $0.52/Mtok rate keeps it below every API rate at all higher volumes, so there is no later volume at which pay-per-token wins the envelope back. The break-even scan finds no U_1 not because pay-per-token is never cheapest—it is, below $u = F/\bar{p}$—but because it is never cheapest again once the credit plan takes over. In all three tiers, and in all 27 sensitivity cells computed below, no such crossover forms.

This is a genuine structural finding rather than a modeling artifact, but it is contingent on the idealized cap (as the credit-cap sweep below makes precise), and both facts must be read carefully. The credit segment of the idealized envelope—from $u = F/\bar{p}$ up to U_2, hundreds of Mtok wide—exists only if a seat's full cap is legally and technically extractable at scale, which is exactly what Eq. (1)'s idealization assumes and what no vendor's terms actually offer. None of the fifteen credit plans in the snapshot extends its subsidized seat rate to programmatic use: eleven disallow API-style access outright, and the four that permit programmatic surfaces meter them through session-windowed, per-account allowances that saturate far below the idealized cap q and are slated for re-metering at standard API rates (Section III). An enterprise cannot contractually or technically arbitrage the $0.52/Mtok asymptotic rate across a programmatic workload of hundreds of Mtok/month. The idealized envelope and the constrained envelope therefore tell two different stories, and both belong in the analysis:

- Idealized envelope (Eq. (1) taken literally): three segments—pay-per-token below $u = F/\bar{p}$, credit from there to U_2, self-hosting beyond—so the crossovers are the credit-overtakes-API point at $u = F/\bar{p}$ and U_2; the credit-to-API threshold U_1 as defined never forms.
- Constrained envelope (terms of service binding): the credit segment is removed for any programmatic workload, not because it is more expensive but because it is unavailable. Deleting it merges the pay-per-token segment straight into self-hosting, so the practical envelope is api $\rightarrow$ self-hosted and the only crossover a buyer actually traverses is U_2.

The practical takeaway is that for a programmatic enterprise workload the credit segment is institutionally inaccessible, so the first procurement boundary a growing workload meets is not the economic crossover at $u = F/\bar{p}$ but the institutional wall of seat saturation and terms-of-service eligibility—which is what makes pay-per-token, not credit, the default programmatic mode below U_2.

a) The null result is contingent on the idealized cap.: That U_1 does not exist is not a fixed law of the rate cards but a consequence of the credit plan's idealized cap $q = 38.4$ Mtok, from which the $0.52/Mtok floor is computed. Scaling that cap down while holding the $20 fee fixed raises the floor F/q, and once the floor rises above an API's blended rate the API undercuts credit at every volume and U_1 appears. Sweeping the effective cap to $\{100\%, 50\%, 25\%\}$ of its idealized value—floors of $0.52, $1.04, and $2.08/Mtok respectively—locates exactly where that happens: no U_1 forms in any tier at the full and half caps, and at the quarter cap it begins to exist for the open tier alone, whose $2.00/Mtok API rate is the only one the $2.08/Mtok quarter-cap floor clears (the mid and frontier rates, $5.00 and $9.00, remain above even that floor, so no U_1 forms for them). The quarter-cap crossover is degenerate rather than interior: the computed value sits at the scan's lower bound because, with the credit floor above the API rate, credit never wins at any volume and pay-per-token is simply cheapest throughout. The null is therefore a knife-edge on the least-reliable input in the dataset—a single secondary-sourced, full-saturation cap estimate. Far from weakening the conclusion, this contingency reinforces it: the entire credit segment on which null-U_1 rests exists only at an extraction rate no terms of service permit, so the finding that a growing programmatic workload is moved off a credit plan by institutional limits rather than by price does not depend on the exact cap at all.

b) The null is seat-specific, not a property of credit plans as a class.: The $0.52/Mtok floor belongs to the

snapshot's single cheapest seat per Mtok of cap. Re-running the comparison against other plans in Table I reverses the conclusion outright: Claude Pro carries the same $20 fee against an estimated cap of only 6.48 Mtok, a floor of $3.09/Mtok—above the open-tier blended rate ($2.00) and above Claude Haiku 4.5's blended $1.80—so against those APIs the credit segment never enters the envelope and pay-per-token is the cheaper mode at every volume. Even where an idealized floor survives, a seat pays for itself only above its own seat-fill point. Consider a buyer holding a Claude Max 20x seat at $200/month (idealized cap 129.6 Mtok, floor $1.54/Mtok) whose workload could run on an open-tier model: against GLM-5.2 pay-per-token at $2.00/Mtok blended, the seat wins only if the buyer actually pushes more than $F/\bar{p} = \$200/\$2.00 = 100$ Mtok/month through it. A buyer whose realized volume sits below that mark—most individual seats, given that the caps in Table I are full-saturation ceilings no real user reaches—pays less by dropping the seat and metering the same workload per token. The API side also holds levers the seat lacks: batch-tier discounts of roughly half off both token rates and cache-read pricing near a tenth of the input rate both lower the effective $\bar{p}$, pushing the seat-fill point further out and widening the volume range in which pay-per-token wins. The null-U_1 finding therefore characterizes the envelope—the cheapest seat at idealized extraction—not credit subscriptions generally; re-computing F/q and $F/\bar{p}$ per vendor, per plan, and per realized volume routinely restores pay-per-token to the envelope.

For U_2 the model behaves as expected and the thresholds are concrete. Self-hosting overtakes pay-per-token at roughly 719 Mtok/month for the mid tier (719.20), 1,598 for frontier (1,598.21), and 1,798 for open (1,797.99). Each is a single-serving-replica calculation—at these volumes demand fits inside one deployable replica's SLO-discounted capacity, so C_{self} equals the flat entry floor $(1+r)\,G$ and the crossover is simply where the API bill matches it—but the replica is not one GPU in every tier. The general rule is that the self-hosting floor is the smallest production-deployable footprint: the smallest replica that can serve the model, plus the N+1 standby. The open and mid proxies (Llama 3.1 8B, TP1) deploy on one GPU per replica, so their floor is two GPUs at $2G = \$3{,}595.98$; the frontier proxy (Llama 3.1 70B, TP4) is sharded across four GPUs and cannot serve on fewer, so its floor is two four-GPU replicas—eight GPUs at $2G = \$14{,}383.92$. Hence for mid, $\$3{,}595.98 \div \$5.00/\text{Mtok} \approx 719$ Mtok/month; for open, $\div \$2.00 \approx 1{,}798$; and for frontier, $\$14{,}383.92 \div \$9.00 \approx 1{,}598$. A striking consequence is that the frontier threshold lands just below the open one despite frontier tokens costing more than four times as much: the frontier tier's higher rate would pull its crossover far lower, but its four-times-larger replica atom pushes back almost exactly as hard, and the two effects nearly cancel. Self-hosting pays off soonest not at the frontier but at the mid tier, which pairs a one-GPU replica with an already-expensive rate card. Relative to the idealized single-replica, benchmark-face-value model ($\gamma = 1$, $r = 0$), every threshold is exactly doubled by the standby replica; the SLO discount leaves the base-case thresholds untouched because even at $\gamma = 0.5$ one replica's capacity ($\approx 4{,}280$ Mtok/month at $\eta = 0.5$) still exceeds the crossover—a margin the sensitivity analysis below shows is not unconditional.

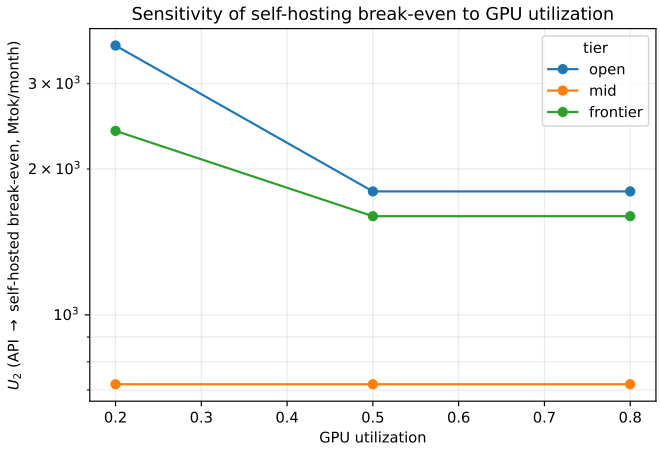


Fig. 2. U_2 is flat in GPU utilization only while one serving replica's SLO-discounted capacity covers the crossover. The mid tier is flat across $\eta \in \{0.2, 0.5, 0.8\}$ at 719.20 Mtok/month; the frontier and open tiers are flat at $\eta \in \{0.5, 0.8\}$ (1,598.21 and 1,797.99) but climb to 2,397.32 and 3,595.98 at $\eta = 0.2$, where per-replica capacity ($\approx$ 1,711 Mtok/month at $\gamma = 0.5$) falls below the crossover and the threshold moves into the multi-replica scaling regime.

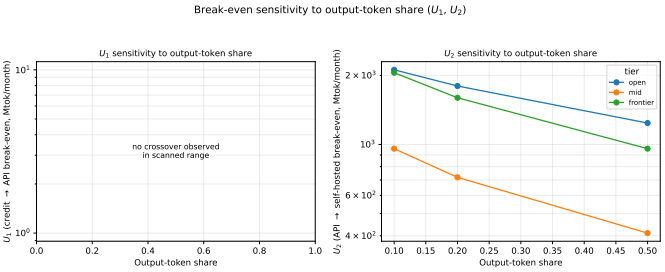


Fig. 3. Break-even sensitivity to output-token share. Left: U_1 is absent at every ρ (the credit-to-API crossover never forms), consistent with Table II. Right: U_2 falls monotonically as output share rises, because a higher share of expensive output tokens raises the blended API rate and makes self-hosting pay off sooner.

D. Sensitivity

We vary the two operational parameters a buyer can influence—GPU utilization η and output-token share ρ—across a 3×3 grid ($\eta \in \{0.2, 0.5, 0.8\}$, $\rho \in \{0.1, 0.2, 0.5\}$) for each tier, and separately sweep the SLO goodput factor $\gamma \in \{1.0, 0.5, 0.25\}$, the assumption the capacity term of Eq. (4) is least certain about. Figure 2 sweeps utilization; Figure 3 sweeps output share.

a) Utilization moves the crossover only past a capacity boundary.: U_2 is flat in utilization exactly as

long as one serving replica's discounted capacity durably covers the crossover— formally, while $\kappa(\eta) = T\gamma\eta S/10^6$ comfortably exceeds $(1+r)\,G/\bar{p}$. In that regime C_{self} is the constant entry floor $(1+r)\,G$, and where the API bill equals a constant is independent of η. The mid tier sits deep inside that regime and is flat across all three utilizations at 719.20 Mtok/month. The frontier and open tiers are flat between $\eta = 0.5$ and 0.8 (1,598.21 and 1,797.99) but not at $\eta = 0.2$: there, per-replica capacity drops to $\approx$ 1,711 Mtok/month, below or barely above the crossover volume, a second (and third) serving replica is needed before the API bill catches the self-hosted floor, and the threshold climbs to 2,397.32 (frontier) and 3,595.98 (open) Mtok/month (Figure 2). The earlier idealized model's unconditional flatness was an artifact of taking benchmark throughput at face value; once capacity is SLO-discounted, low utilization pushes the first crossover into the scaling regime, where η sets the spacing of the steps in Eq. (5) and therefore the threshold itself. Two boundary conditions follow and should be checked before reusing any number in Table II: the flatness condition above, and an existence condition—if goodput degrades far enough that $\kappa < G/\bar{p}$ (for the frontier tier, an effective aggregate throughput below $\approx$ 304 tok/s per replica, roughly a tenfold degradation from the benchmark at $\eta = 0.5$), the self-hosted per-token floor G/κ exceeds the API rate at every replica count and U_2 ceases to exist.

b) Output share moves it monotonically.: U_2 decreases as the output share rises, because output tokens are the expensive side of the rate card and a higher ρ lifts $\bar{p}$ in Eq. (3) (Figure 3, right). For frontier, U_2 falls from 2,054.85 Mtok/month at $\rho = 0.1$, to 1,598.21 at $\rho = 0.2$, to 958.93 at $\rho = 0.5$. The same monotonic fall appears in every tier: mid moves 958.93 $\rightarrow$ 719.20 $\rightarrow$ 410.97 and open moves 2,115.28 $\rightarrow$ 1,797.99 $\rightarrow$ 1,239.99 over the same three output shares. A generation-heavy workload—long completions, agentic chains, code synthesis—crosses into self-hosting territory at roughly half the volume of a retrieval- or classification-heavy one dominated by input tokens. U_1 is absent at every output share (Figure 3, left): the credit-to-API crossover never forms regardless of mix, since above the crossover of Eq. (7) the credit plan's asymptotic rate stays below every API rate.

c) Caching raises the crossover.: The base case fixes the cache-hit fraction at $h = 0$, which maximizes the blended rate $\bar{p}$ and therefore minimizes U_2; any real caching lowers $\bar{p}$ and pushes U_2 up, so self-hosting pays off later. Sweeping $h \in \{0, 0.25, 0.5, 0.75\}$ at the base utilization and output share, U_2 rises monotonically in every tier: frontier from 1,598.21 to 1,775.79, 1,997.77, and 2,283.16 Mtok/month; mid from 719.20 to 790.33, 877.07, and 985.20; and open from 1,797.99 to 2,029.33, 2,329.00, and 2,732.51. A coding or agentic workload that reuses prefixes at a 50–75% hit rate—the paper's own motivating case—therefore reaches self-hosting economics some 22–52% later than the $h = 0$ base case implies (the exact lift depends on the tier's cached-input discount: 25–43% at the frontier, 22–37% at mid, 30–52% in the open tier), so Table II should be read as a lower bound on the switching volume for cache-friendly workloads. The lift itself is an upper bound on the caching effect: the model prices cache reads but not the write premiums some vendors charge on cache misses (e.g. 1.25–2$\times$ the input rate per written prefix), which claw back part of the discount at high miss churn.

d) SLO goodput moves it past a boundary, like utilization.: Sweeping $\gamma \in \{1.0, 0.5, 0.25\}$ at the base utilization, output share, and redundancy: at $\gamma = 1.0$ and 0.5 every tier's threshold is unchanged—one serving replica covers the crossover with capacity to spare, so the entry floor, not throughput, sets U_2. At $\gamma = 0.25$ (a fourfold degradation from the benchmark's throughput-optimal operating point, plausible under tight time-to-first-token targets), per-replica capacity falls to $\approx$ 2,140 Mtok/month and the frontier and open thresholds climb to 2,397.32 and 2,696.99 Mtok/month respectively, while mid stays at 719.20. The mechanism is the same capacity boundary as in the utilization sweep: γ and η enter Eq. (4) multiplicatively, so a report of either one alone understates the joint risk—at $\gamma = 0.25$ and $\eta = 0.2$ the frontier tier's per-replica capacity ($\approx$ 856 Mtok/month) is barely half its crossover volume.

e) A negotiated infrastructure discount moves the crossover linearly—and leftward.: Every parameter so far either leaves U_2 unchanged or pushes it up; the enterprise discount d is the one buyer-side lever that pulls it down. In the single-serving-replica regime $U_2 = (1+r)(1-d)\,G/\bar{p}$, linear in d: sweeping $d \in \{0, 0.10, 0.20\}$ at the base case, the frontier threshold falls from 1,598.21 to 1,438.39 and 1,278.57 Mtok/month, mid from 719.20 to 647.28 and 575.36, and open from 1,797.99 to 1,618.19 and 1,438.39. A ten-point discount on infrastructure rental therefore buys a ten-percent-earlier crossover, no more—useful, but an order of magnitude weaker than the levers that act on the other side of the comparison. Two asymmetries are worth naming. First, the discount applies to the self-hosted mode only if the buyer's agreement covers the GPU SKU in question; committed-spend agreements priced against general compute do not always extend to scarce accelerator capacity at the same rate. Second, the same negotiation channel usually exists on the API side too—volume and committed-use discounts on token rates—and a discount on $\bar{p}$ pushes U_2 up by the same linear logic, so an enterprise negotiating both sides at similar percentages roughly cancels the effect on the crossover while lowering absolute spend in either mode.

E. Interpretation

The thresholds computed here are volatile in level and stable in structure. Every number is anchored to a single July 2026 snapshot, one GPU SKU, and specific open-weight proxies standing in for frontier-quality serving; a

new rate card, a cheaper reserved instance, or a different benchmark would move 1,598.21 or 1,797.99 by 10–50%. Quality parity across modes is assumed within a tier and is itself a strong idealization: a frontier API and a self-hosted 70B open-weight model are treated as substitutes here, which is a cost proxy, not a statement of equivalence. The same proxy gap applies, more sharply, to the open tier, where the self-host benchmark is only an 8B model: the open-tier API offerings (GLM-5.2, a 753B-parameter MoE [43]; DeepSeek V4 Pro, 1.6T total parameters [42]) are large, frontier-competitive MoE models an 8B deployment cannot serve, so the open-tier U_2 is an optimistic lower bound—a quality-matched open self-host would need many more GPUs and cross well later. The mid tier carries the same caveat, previously left implicit: its self-host side is also the 8B proxy, standing in for GPT-5.6 Terra-class quality, so the mid-tier $U_2 = 719.20$—the tier where self-hosting appears to pay off soonest—is likewise an optimistic lower bound, not a quality-matched comparison. A cost granularity caveat compounds this: the reserved GPU price is normalized per GPU from an eight-GPU AWS p5.48xlarge node sold only as a whole node, so the true minimum commitment on that SKU is the node, not one GPU; genuine single-GPU granularity exists at neoclouds (e.g. the RunPod on-demand row in the snapshot) but at higher hourly rates. The exact volumes should be read as illustrative, not as procurement advice.

a) What the model still leaves out.: Two omissions bound the thresholds from opposite sides. First, operational labor: Eq. (5) prices hardware only. Running a serving stack in production requires engineering time—deployment, upgrades, incident response—whose monthly cost does not appear in G and, for a first deployment, is often the same order as the hardware floor itself (even a fractional MLOps allocation of a few thousand dollars per month is comparable to the mid tier's $3,595.98 entry floor). We deliberately keep labor out of the price model, because it varies with organization far more than any rate card; Section VI-C treats it as a switching friction and quantifies how far it moves the effective threshold. Every U_2 in Table II is therefore a hardware-only lower bound. Second, provisioned throughput: for closed frontier models—which cannot be self-hosted at any price—the realistic fixed-cost alternative to pay-per-token is the vendor's own provisioned-throughput tier (e.g. Azure OpenAI PTUs, AWS Bedrock provisioned throughput), not a rented GPU running an open-weight proxy. The snapshot carries one such row ($29,200/month per PTU), excluded from the model because it prices a capacity unit with a vendor-defined tokens-per-minute budget rather than a GPU, and its usable throughput is not published at GPU-comparable granularity. Taken at face value against the frontier blended rate, that price would put a provisioned-throughput crossover near $29,200/$9.00 $\approx$ 3,244 Mtok/month—roughly twice the proxy-based frontier U_2—but the number is indicative only. The frontier row of Table II should accordingly be read as the cost of frontier-adjacent open-weight self-hosting, not as the volume at which a closed frontier model becomes cheaper to run in-house; for the closed model itself, no such volume exists below the provisioned tier's price.

What is stable is the shape of the answer. There are two candidate crossovers; whether each exists is decided by the relationship between a flat rate, a per-token rate, and a capacity floor, not by their momentary values; the credit-plan subsidy is real but fenced by terms of service, so the practical envelope runs from pay-per-token to self-hosting across a single crossover U_2; and U_2 is governed by the production entry floor $(1 + r)\,G$ and the token mix—and by utilization and SLO goodput only once per-replica capacity fails to cover the crossover. These structural facts survive any repricing. The durable contribution of this section is therefore the model and the re-runnable analysis that produced Table II, not the entries in it: an enterprise re-runs Eqs. (1)–(5) against its own contracted rates and its own workload mix to locate its own U_2. Section V turns that observation into a monitoring framework, deriving from this model exactly which quantities an organization must measure to know where it sits relative to its crossovers.

V. A Monitoring Framework for Enterprise AI Cost

Section IV located the crossovers a buyer sits between; knowing they exist does not tell an organization which numbers to watch day to day. FinOps dashboards for AI spend tend to accumulate metrics by convenience—whatever the billing API exposes—rather than by relevance. We propose the opposite starting point: a metric earns its place in the monitoring framework only if it changes a procurement decision. A number that moves without ever crossing a threshold that would make a rational buyer switch modes, add seats, provision a GPU, or renegotiate a rate is not a monitoring metric; it is decoration. Applying that filter to the cost model of Section IV yields three metric families—position, drift, and unit economics—each answering a different question a procurement decision depends on: where the organization sits on the Efficiency Curve, whether it is approaching a threshold, and whether the spend is worth it once it arrives.

A. Position Metrics: Where Are You on the Curve

Position metrics locate the organization relative to U_1 and U_2 at the present moment. The most basic is total Mtok/month, per workload and in aggregate, computed the same way u is computed in Eq. (6): without it, no other quantity in this section can be related back to the curve at all. Alongside it, the blended effective $/Mtok actually paid—the realized counterpart of Eq. (3), computed from the organization's own invoices rather than the rate card—tells the buyer whether it is paying the price the model

assumes or something worse, because negotiated discounts, minimum commitments, and contract tiers routinely move the realized rate away from the posted one.

The remaining two position metrics are mode-specific, and the credit-plan one requires the correction Section IV established: because the credit-to-API crossover U_1 never forms under the July 2026 snapshot, the metric that matters for a credit-plan buyer is not distance from an idealized cap whose extraction no vendor's terms support. It is seat utilization: consumption against the realistic per-seat throughput a user can extract under the vendor's session and rate-limit constraints, not the idealized cap q used in Eq. (1). This metric carries a collection caveat: none of the vendor consoles behind the snapshot's fifteen credit plans exposes per-seat consumption in token units, so seat utilization cannot be read directly and must be proxied—from workspace-admin usage exports, gateway logs, or message counts multiplied by a tokens-per-message estimate—inheriting the same estimation uncertainty Section VII flags for the caps themselves. Seat utilization approaching saturation is the signal to add seats or migrate the workload to pay-per-token; it is, in effect, the organization's only legitimate observation of where the practical U_1 sits, since that boundary is set by seat saturation and eligibility rather than by price. For an organization that has already crossed U_2, GPU utilization η is the analogous position metric—but only in that regime. Section IV's sensitivity analysis showed U_2 itself is flat in η while one serving replica's SLO-discounted capacity covers the crossover, so in that regime tracking utilization before crossing tells a buyer nothing about when to cross; it becomes the metric that governs fleet sizing only after self-hosting is adopted and volume enters the multi-replica scaling regime of Eq. (5)—or earlier, if low utilization or degraded SLO goodput has already pushed the crossover itself into that regime.

B. Drift Metrics: Are You Approaching a Threshold

Position metrics are a snapshot; drift metrics convert the curve into a forecast. The central one is time-to-threshold: given the current volume u, a threshold U_i—in practice the economic threshold U_2, or the practical seat-capacity ceiling that stands in for the absent U_1—and the trailing rate of volume growth $\dot{u}$ (growth in Mtok/month per month), the number of months before the organization crosses that threshold at the current growth rate is

$$\tau_i = \frac{U_i - u}{\dot{u}}. \tag{8}$$

τ_i is the metric that turns a static crossover into an action item: a short τ_2 is the trigger to begin GPU procurement—with its lead time—before the API bill exceeds the self-hosting cost, and, for organizations still gated by seat saturation, the same formula applied to the practical seat-capacity ceiling gives the lead time to negotiate additional credit-plan seats or begin an API migration. $\dot{u}$ itself—the month-over-month token growth rate—is worth tracking on its own, since a sudden acceleration can shorten τ_i far faster than a one-time position check would reveal.

Two further drift metrics track the parameters that move the thresholds rather than the volume that approaches them. Cache hit rate trend (h in Eq. (3)) drifting downward raises the realized blended rate $\bar{p}$ even at constant volume, silently eroding the position computed in Section V-A without any change in u. Output-share trend (ρ) matters for the same reason and in a direction Section IV already established: U_2 falls monotonically as output share rises, so a workload drifting toward longer, more generative completions is drifting toward self-hosting becoming favorable at a lower volume than previously computed. Neither metric requires a volume change to matter; both change where the threshold itself sits.

C. Unit-Economics Metrics: Is the Spend Worth It

Position and drift metrics answer where the organization sits and how fast it is moving; neither says whether the spend is buying anything. The first step toward that answer is cost per request: total spend divided by request count, tracked per workload and always read alongside the token weights Section V-A provides, since a request is not a unit of volume and the ratio is meaningful only with that caveat attached. Its decision value is as a mix detector: a drifting cost per request under a stable rate card means the workload's token profile is shifting, which invalidates the blended-rate assumption $\bar{p}$ behind the organization's computed U_2 and triggers a per-workload re-pricing of Eq. (3). Above it sits cost per business outcome—per ticket resolved, per pull request merged, per document processed—the metric that closes the loop between spend and value, because it is denominated in the unit the business actually cares about rather than in tokens or requests. A workload can be perfectly positioned on the Efficiency Curve and still be a bad idea if the cost per outcome exceeds the cost of the labor it replaced.

That comparison is also where this framework connects to the cost-shift thesis of the author's prior work [1]: falling per-token prices do not shrink AI spend, because cheaper tokens induce more usage than they save, so the labor-to-compute substitution shows up as restructured, not reduced, expenditure. The monitoring consequence is that spend growth rate compared against volume growth rate is itself a decision-relevant metric—if aggregate spend is growing faster than the token-weighted volume that Section V-A already tracks, the excess is not explained by the cost model of Section IV at all, and the right response is not to renegotiate rates but to govern usage growth directly. Read together with cost per outcome, this is what makes the labor→compute shift visible in the accounting rather than absorbed silently into a rising bill.

TABLE III
The monitoring framework: metrics that earn their place because each informs a specific procurement decision. u, U_1, U_2, ρ, and h are as defined in Section IV.

Metric	Definition	Decision informed	Collection source
Total Mtok/month	u per workload and aggregate	Locates the organization on the Efficiency Curve; determines whether any mode switch is worth evaluating at all	Vendor usage API / gateway logs
Blended \$/Mtok paid	Realized $\bar{p}$ from invoices (cf. Eq. (3))	Whether contracted rate still beats the alternative mode	Billing / invoice reconciliation
Seat utilization	Consumed vs. realistic per-seat throughput (not idealized cap q)	Add seats vs. migrate off credit plan to API	Proxied: admin usage export / gateway logs / msg count × tokens-per-msg (no vendor per-seat token metric)
GPU utilization η	Serving load vs. capacity, Eq. (4)	Add/remove GPUs (post-U_2 only)	GPU fleet telemetry
Token growth rate $\dot{u}$	Month-over-month change in u	Sets the urgency of the next procurement review; decides whether provisioning lead time must start now	Usage API, trailing window
Time-to-threshold τ_i	$(U_i - u)/\dot{u}$, Eq. (8)	When to start GPU procurement or seat/API renegotiation	Derived from above
Cache hit rate trend	Trend in h	Triggers re-pricing of $\bar{p}$ and prompt/cache remediation when the trend falls	API response metadata
Output-share trend	Trend in ρ	Triggers re-computation of U_2; brings forward the self-hosting evaluation when share rises	Token accounting by role (input/output)
Cost per request	Spend ÷ requests, token-weighted caveat (Section V-C)	Flags workload-mix shifts that invalidate $\bar{p}$ behind U_2; triggers per-workload re-pricing	Gateway logs × billing
Cost per business outcome	Spend ÷ outcomes (tickets, PRs, documents)	Continue, expand, or discontinue the workflow	Spend data joined to product/ops logs
Spend growth vs. volume growth	Spend growth rate − token growth rate	Renegotiate rate vs. govern usage growth [1]	Billing × usage API

D. Anti-Patterns

Not every number on a cost dashboard clears the bar this section sets, and the most common failures are worth naming because they are the metrics most FinOps tooling reports by default. Raw request counts without token weights are the clearest offender: a request can carry ten tokens or ten thousand, so counting requests conflates workloads of wildly different volume and cannot be compared against U_1, U_2, or Eq. (8) at all—it does not locate a position on the Efficiency Curve, and it does not predict a crossing. Spend dashboards without a volume denominator fail for the symmetric reason: a rising monthly bill is uninformative on its own, because it does not distinguish a worsening effective rate from simple, efficient growth in adopted usage—exactly the ambiguity Section V-C resolves by comparing spend growth against volume growth rather than reading either in isolation. Both anti-patterns share a diagnosis: they report activity rather than position relative to a threshold, so no procurement decision follows from watching them, however good they look on a dashboard.

VI. Optimization Levers and Switching Rules

Section IV located the crossovers, and Section V specified the metrics that reveal where an organization sits relative to them. This section closes the loop: what an organization does with that reading. We separate two actions that FinOps practice routinely conflates. Within-mode optimization—the serving, caching, and routing techniques surveyed in Section II—moves an organization's position on the Efficiency Curve without changing which mode is cheapest at its volume; a better-tuned API integration is still an API integration, priced by Eq. (2). Switching changes the mode itself, and is warranted only when a metric from Table III crosses a threshold, not merely when a lever happens to be available. We take each in turn, then state the frictions that shift those thresholds in practice.

A. Levers Within Each Mode

a) Credit plans.: The two levers available to a credit-plan buyer are allocative rather than technical: seat-count rightsizing—matching the number of seats to the realistic per-seat throughput of Section V-A, not the idealized cap q in Eq. (1)—and plan-tier matching, moving

individual users to the cheapest tier that still clears their session-based usage pattern. Neither changes $C_{\text{credit}}(u)$'s functional form; both reduce the seat count $\lceil u/q \rceil$ actually purchased for a given realized workload.

b) Pay-per-token APIs.: Four levers act directly on the blended rate $\bar{p}$ in Eq. (3). Caching raises the cache-hit fraction h by reusing repeated context—prefix caching at the serving layer [7], [9] and cross-session prompt reuse [10] both lower the effective input rate paid, since $p_{\text{cache}} < p_{\text{in}}$. Prompt compression shrinks u itself at fixed task quality by cutting redundant context before it is billed. Routing to cheaper tiers—cascading through progressively more expensive models only when a cheap one looks unreliable [17], or a learned router that dispatches each query to the cheapest model expected to satisfy it [18]—lowers the rate paid for the share of the workload that does not need the frontier tier. Batch APIs, offered at a standing discount for asynchronous, latency-tolerant workloads [3], [32], lower p_{in} and p_{out} directly for the fraction of u that can queue. All four levers reduce $\bar{p}$ or u; none touches G or q, so none relocates the API curve relative to the credit or self-hosted curves—they slide the organization along $C_{\text{api}}(u)$, not off it.

c) Self-hosted capacity.: Three levers apply once GPUs are on the books, and Section IV's sensitivity result orders them by leverage. Utilization improvement—raising η in Eq. (4) through better batching and scheduling [7], [8]—is the dominant lever within self-hosted mode precisely because Section IV showed η does not move U_2 itself while one serving replica's SLO-discounted capacity covers the crossover: in that regime the break-even is set by the flat entry floor $(1+r)\,G$, not by how full the replica runs, so every idle GPU-hour is capacity already paid for and not resold. Raising η is pure margin recovery on a bill that does not change, which is exactly why it becomes the dominant economic lever once an organization is past U_2, even though it was irrelevant to reaching U_2 in the first place. Quantization [11], [12] and related footprint-reduction techniques—speculative decoding [13], distillation [14]—raise the sustained throughput T a fixed fleet delivers, which raises κ in Eq. (4) and lowers the GPU count $\lceil u/\kappa \rceil$ a given volume requires. Engine tuning—scheduler- and kernel-level work in the same serving stack [15], [16]—is the mechanism behind both of the preceding levers rather than a separate one: it is how η and T are actually raised in production.

Across all three modes the pattern is the same: these techniques act on the free parameters of Eqs. (1)–(5)—q, $\bar{p}$, η, T—not on F, G, or the mode boundary itself. They determine where an organization sits on $C_m(u)$; they do not determine which m minimizes Eq. (6) at a given u. That is decided only by switching. The one exception is commercial rather than technical: negotiated pricing. An enterprise infrastructure discount scales G by $(1-d)$ and pulls U_2 left in direct proportion (Section IV's sweep: ten points of discount, ten percent earlier crossover), and a negotiated token rate acts on $\bar{p}$ and pushes it right; procurement negotiation is therefore the only optimization lever that relocates the mode boundary itself, and a buyer holding discounts on both sides should re-derive U_2 from the contracted rates, not the list prices modeled here.

B. Switching Rules

Every rule below is a condition over a metric defined in Table III, not a subjective judgment call.

Rule 1 (credit → API). Section IV established that U_1 does not exist as a price crossover, so this switch is never triggered by a rate comparison. It is triggered by the constraint that makes the idealized rate unavailable in the first place: sustained seat consumption against the realistic per-seat cap of Section V-A, combined with a change in what the marginal seat is for. Formally, switch when

$$\text{seat consumption}_t > \text{cap}_{\text{realistic}} \text{ (sustained)}$$
$$\text{and} \quad \text{marginal seat demand} \in \text{programmatic use.}$$

The first clause alone justifies buying another seat; it is the second clause—demand the plan's subsidized seat allowance cannot serve—that justifies leaving the mode rather than adding to it.

Rule 2 (begin self-hosting evaluation). Using the time-to-threshold metric τ_2 of Eq. (8) and a lead time L for GPU procurement (reserved-instance turnaround or hardware lead time, whichever applies), begin a self-hosting evaluation when

$$\tau_2 = \frac{U_2 - u}{\dot{u}} < L.$$

This is a trigger to start evaluating, not to switch immediately: U_2 itself should be re-derived from the organization's own contracted $\bar{p}$ and workload mix (Section IV's Interpretation), and the frictions below should be applied to it, before a provisioning decision is made. Because U_2 is not fixed, two drift metrics from Section V-B can pull the trigger forward even at constant u: a falling cache-hit rate trend raises $\bar{p}$ and lowers the effective U_2 relative to the organization's realized cost, and a rising output-share trend ρ lowers U_2 directly, per Section IV's finding that generation-heavy workloads cross into self-hosting territory at roughly half the volume of input-heavy ones. Both should re-trigger the τ_2 check against the recomputed threshold rather than the stale one.

Rule 3 (fleet sizing, post-U_2). Once self-hosted, further mode switching is not indicated by GPU utilization η—Section IV showed U_2 is flat in η while one serving replica covers the crossover—but η is exactly the metric that governs whether to add or shed GPU capacity within self-hosted mode, since it determines how close the fleet is to the next step of Eq. (5). Sustained η near saturation is the trigger to provision the next GPU before a request queue forms; sustained low η across a multi-GPU fleet is the trigger to consolidate, since excess capacity at that point is pure cost with no offsetting option value.

C. Switching Frictions

None of the rules above is frictionless in practice. Each has a companion non-price cost that Section IV's idealized crossovers omit, and every one of them we identify pushes U_2 right—it takes more volume than the price-only calculation suggests before self-hosting actually pays off.

Quality non-equivalence. Section IV's U_2 for the frontier tier is computed against an open-weight proxy (Llama 3.1 70B) standing in for a closed frontier model, a cost proxy rather than a claim of equivalence. Where the capability gap is real, the practical response is not to abandon self-hosting once past U_2 but to route the workload's capability-sensitive share back to the frontier API even after crossing—a partial, workload-scoped switch rather than the single all-or-nothing crossover Table II models, which raises the volume of fully self-hostable demand above the computed U_2.

Ops headcount. Self-hosting adds a fixed cost that Eq. (5) omits: the engineering time to operate the serving stack. Because this cost does not scale with GPU count over the range a small fleet occupies, it enters as an addition to the entry floor $(1 + r)\,G$ in the single-serving-replica regime, and the break-even volume moves proportionally:

$$U_2' \approx \frac{(1 + r)\,G + \text{ops}}{\bar{p}}. \tag{9}$$

Illustratively, at the frontier tier's $\bar{p}$ = \$9.00/Mtok, adding even a fractional on-call allocation of \$3,000/month to the eight-GPU entry floor (\$14,383.92/month) moves the threshold from 1,598.21 to ≈ 1,932 Mtok/month—a roughly 21% increase, because the ops cost is amortized over the same fixed floor that $(1 + r)\,G$ is. The effect is proportionally larger for a single-GPU tier, where ops is a bigger multiple of the fixed cost it rides on: the same \$3,000 added to the mid tier's two-GPU floor of \$3,595.98 (at $\bar{p}$ = \$5.00) moves its threshold from 719.20 to ≈ 1,319 Mtok/month, nearly doubling it. And \$3,000/month is itself a conservative illustration—roughly a fifth of a loaded MLOps engineer; a first production deployment staffed at half an FTE would roughly double the adjustment again. The friction shrinks as fleet size grows and ops cost is spread across more replica-months, but for an organization evaluating its first replica it is often the largest single adjustment to Eq. (5).

Deeper redundancy. The base model already carries the minimum serviceable configuration—Section IV sets $r = 1$ (N+1) in Eq. (5), which is what doubles the entry floors of Table II relative to a bare single replica. Stricter availability targets go further: multi-zone or multi-region serving, or N+2 against correlated failures, each add another standby replica, and every additional standby moves the threshold by $G/\bar{p}$— +799 Mtok/month at the frontier, +360 at mid, +899 at open, per replica. For a redundancy-bound workload this is not optional tuning but a precondition of serving at all, so the floor should be priced at the availability target the workload actually carries, not at the model's minimum.

Latency and compliance. Data-residency, regulatory, or latency-sensitivity requirements can force self-hosting below the computed U_2—the one case where a friction moves the threshold left, because the constraint sits outside the cost model entirely and overrides it rather than adjusting it. More commonly, the reverse applies: compliance review, security audit, and change-management overhead for a newly self-hosted service add lead time and fixed cost on top of Eq. (9), pushing U_2 further right.

Egress and lock-in. Migrating a workload off pay-per-token incurs a one-time cost—data egress fees, re-tooling, contract termination exposure—that Eq. (5) does not amortize because it is not a monthly charge. That cost is only worth paying if the volume gap persists long enough to recoup it: switching is favorable only when the expected holding time T_{hold} above U_2' satisfies

$$T_{\text{hold}} \cdot \left[C_{\text{api}}(u) - C_{\text{self}}(u)\right] \;>\; M, \tag{10}$$

where M is the one-time migration cost. A workload with volatile or uncertain growth—one where $\dot{u}$ from Table III is itself unstable—can sit above U_2' today and fail Eq. (10) anyway, because the switch cannot be recouped before volume falls back below threshold.

Taken together, these frictions do not change the shape of Section IV's result—there is still a single price crossover U_2 past which self-hosting is cheaper on tokens alone—but they routinely move the volume at which switching is actually worth executing well above the number Table II reports, sometimes by a factor of two or more. An organization that switches the moment it crosses the idealized U_2 and ignores Eqs. (9)–(10) is following the letter of the model while missing the cost it was built to avoid.

D. A Worked Example

Return to the ten-person team of Section I, now running automated pull-request review on Claude Opus 4.8 over the API at $\bar{p}$ ≈ \$9.00/Mtok. Rule 1 has already fired—the workload is programmatic, so no credit seat can serve it at any price, which is why they are on pay-per-token rather than their Max 20x seats. Suppose the CI workload runs at u = 250 Mtok/month and grows at $\dot{u}$ = 40 Mtok/month per month. The base frontier threshold—which already prices the production floor of two four-GPU replicas (N+1) at SLO-discounted throughput—is U_2 = 1,598.21 Mtok/month, so time-to-threshold is $\tau_2 = (1{,}598.21 - 250)/40 \approx 34$ months—far longer than a reserved-GPU lead time—so Rule 2 does not fire and the team should stay on the API. Two corrections push the real trigger further out still: a PR-review workload that re-reads the same repository context at a high cache-hit rate lifts the base threshold

from 1,598.21 to ≈ 2,283 Mtok/month at $h = 0.75$ (Section IV); and a fractional ops allocation of $3,000/month on the eight-GPU floor (Eq. (9)), applied at the cache-adjusted blended rate, puts the friction-adjusted threshold at ≈ 2,759 Mtok/month. What they should actually watch, per Table III, is the cache-hit-rate trend and the output-share trend—both move U_2 with no change in volume—and $\dot{u}$, whose acceleration would shorten τ_2 faster than a periodic position check would reveal. The correct action today is therefore to remain on pay-per-token, instrument cache-hit and output-share, and begin a self-hosting evaluation only when τ_2 against the friction-adjusted U_2 (≈ 2,760 Mtok/month here) drops below their GPU lead time.

VII. Limitations

Pricing volatility. Every number in Table II is anchored to a single July-2026 snapshot of seat fees, API rates, and GPU rental prices. Vendors reprice credit plans, cut API rates, and adjust GPU rental terms on a timescale of months, not years, and any of the levels reported here—1,598.21, 719.20, or 1,797.99 Mtok/month—will be stale well before this paper is. What survives repricing is not the number but the method: the cost functions of Eqs. (1)–(5), the envelope construction of Eq. (6), and the analysis/code that produced Table II from data/pricing_2026.json are the paper's actual contribution. An organization—or a future version of this paper—re-runs that analysis against a current rate card to get a current threshold; the levels reported here are illustrations of the method, not a claim about where the market will sit next quarter.

Quality non-equivalence across modes. The Efficiency Curve compares cost at an assumed capability parity within each tier, and that assumption is strongest where it is least examined: frontier-tier self-hosting in Table II is proxied by an open-weight 70B model (Llama 3.1 70B) standing in for a closed frontier model no enterprise can actually self-host. That proxy is a cost surface, not an equivalence claim—the frontier API and the self-hosted 70B are not the same model, and a workload with a genuine capability requirement may find U_2 irrelevant regardless of volume, because the cheaper mode is not an acceptable substitute at any price. Section VI-C's partial-switching response—routing only the capability-insensitive share of a workload past U_2—mitigates this but does not remove it from the model. The same proxy gap runs the other way in the open tier and is arguably sharper there: its API offerings (GLM-5.2, DeepSeek V4 Pro) are large, frontier-competitive MoE models, but the self-hosting benchmark paired with them is an 8B model that cannot serve, or even load, those weights on the single GPU per replica priced here. The open-tier U_2 = 1,797.99 Mtok/month is therefore an optimistic lower bound on self-hosting at matched quality: a genuinely quality-matched open self-host would require multi-GPU replicas, like the frontier proxy already does, enlarging the entry floor and pushing the open-tier threshold well past the number reported here.

Hyperscaler node granularity. The self-hosting floor G is built from an AWS reserved H100 price normalized per GPU, but the underlying SKU—the p5.48xlarge—is an eight-GPU node sold only as a whole node. The true minimum reserved commitment on that SKU is therefore the full eight-GPU node, not the one- or four-GPU replica the thresholds assume; the per-GPU figure is a normalization, not a purchasable unit. Genuine per-GPU (or per-replica) granularity does exist, but at neoclouds—the snapshot's RunPod on-demand row is an example—whose hourly rates are higher than the reserved price used here. A buyer committed to the reserved SKU faces a larger indivisible fixed cost than G implies unless the node is shared across concurrent workloads, which the single-workload framing above does not model.

Single-workload framing. The cost model of Section IV treats an organization as if it had one workload with one volume, one output share, and one cache-hit rate. Real organizations run many workloads simultaneously, at different points on their own volume curves, and the cheapest procurement mix is very likely a portfolio allocation across credit plans, pay-per-token APIs, and self-hosted capacity rather than a single crossing from one mode to the next. Formalizing that portfolio problem—which workloads to route to which mode simultaneously, and how the crossovers interact when GPU capacity is shared across several workloads at once—is future work; this paper's single-workload framing is a deliberate simplification needed to make the crossovers computable in closed form, not a claim that real procurement is single-workload in practice.

Cap-estimation uncertainty. No credit-plan vendor in the pricing snapshot publishes its usage cap in token units. The idealized caps used throughout this paper—38.4 Mtok/month for ChatGPT Plus, 129.6 Mtok/month for Claude Max 20x, and the rest of data/pricing_2026.json's credit_plans entries—are reconstructed from stated relative multipliers, session rate limits, and credit-to-dollar conversions under an assumption of continuous full-window saturation that no real user achieves. Where the underlying multiplier itself comes from a secondary rather than vendor-confirmed source, Table I marks the row accordingly. This uncertainty affects U_2 only indirectly, through the credit segment's role in the constrained envelope of Section IV, but it is the single largest source of imprecision in any number this paper reports about credit-plan economics specifically.

VIII. Conclusion

The question this paper opened with—which of three structurally different pricing models is cheapest for a given inference workload, and when does the answer change—does not have a fixed answer, because the prices behind it move every quarter and the volume in front of it keeps

growing [1]. What the Efficiency Curve supplies is not a fixed answer but a well-posed question: instead of asking whether AI inference is expensive, an organization asks where its current volume sits relative to its thresholds—the economic crossover U_2 and the institutional boundary that stands in for U_1—and how fast it is drifting toward either.

The two thresholds turn out to be different in kind, and that difference is the paper's central result. U_2, the crossover from pay-per-token to self-hosting, is a genuine economic threshold: it is where a linear per-token bill overtakes a fixed capacity cost, it is computable from a rate card and a GPU price, and Section IV computes it concretely at roughly 719, 1,598, and 1,798 Mtok/month across the mid, frontier, and open tiers—priced at the production entry floor of a deployable replica plus an N+1 standby, at SLO-discounted serving throughput. U_1, the crossover from credit plans to pay-per-token, is not economic at all under the assumptions a rate card alone would imply: pay-per-token is the cheapest mode only below the seat-fill point $u = F/\bar{p}$, and beyond that point the idealized credit rate of the snapshot's cheapest seat undercuts every API price in the snapshot and stays lower, so the credit-to-API crossover never forms in the direction U_1 names. The boundary that actually governs a credit-plan buyer is institutional, not economic: it is set by seat saturation and by terms that deny the subsidized seat rate to the programmatic use a growing workload eventually needs. Treating that boundary as a price problem, as a naive reading of the rate card would invite, misses the mechanism that actually moves a buyer off a credit plan.

This is why the monitoring framework of Section V tracks position and drift rather than price alone: seat utilization and eligibility reveal the practical U_1 that no rate card shows, while blended rate, GPU utilization, and time-to-threshold reveal distance from the economic U_2 that the cost model computes directly. Both are moving targets—U_2 shifts with output-token share and cache-hit rate even at constant volume (Section V-B), and the frictions of Section VI-C push it further right than the price-only number suggests. Monitoring the right metrics does not fix that volatility; it is what makes the volatility legible in time to act on it, which is the only form of cost control available to an organization whose spend, on the evidence of this paper's own prior finding, is going to keep growing regardless of how efficient the underlying tokens become.

References

[1] T. A. Do, "The Jevons paradox in AI infrastructure: Efficiency, rebound, and the limits of cost reduction," SSRN Working Paper, Abstract ID 6775299; Zenodo, DOI 10.5281/zenodo.20361387, 2026, arXiv submission pending endorsement (cs.AI / cs.CY) as of July 2026; cited here via its SSRN/Zenodo preprint record. [Online]. Available: https://doi.org/10.5281/zenodo.20361387

[2] Anthropic, "Claude pricing (Pro, Max, Team plans)," https://claude.com/pricing, 2026, accessed 2026-07-12.

[3] ——, "Pricing — Claude Developer Platform (Opus 4.8, Sonnet 4.6, Haiku 4.5 API rates)," https://platform.claude.com/docs/en/about-claude/pricing, 2026, accessed 2026-07-12.

[4] B. Zhuang, J. Qiao, M. Liu, M. Yu, P. Hong, R. Li, X. Song, X. Xu, X. Chen, Y. Ma, and Y. Gao, "Beyond benchmarks: The economics of AI inference," arXiv preprint arXiv:2510.26136, 2025.

[5] M. Du, "Tiered Super-Moore's Law: Price evolution, production frontiers, and market competition in large language model inference services," arXiv preprint arXiv:2603.28576, 2026.

[6] C. Patil, "Beyond per-token pricing: A concurrency-aware methodology for LLM infrastructure cost estimation," arXiv preprint arXiv:2606.11690, 2026.

[7] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in Proceedings of the 29th Symposium on Operating Systems Principles (SOSP), 2023, pp. 611–626.

[8] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for transformer-based generative models," in 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022, pp. 521–538.

[9] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "SGLang: Efficient execution of structured language model programs," arXiv preprint arXiv:2312.07104, 2023.

[10] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong, "Prompt Cache: Modular attention reuse for low-latency inference," arXiv preprint arXiv:2311.04934, 2023.

[11] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," arXiv preprint arXiv:2210.17323, 2022.

[12] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: Activation-aware weight quantization for LLM compression and acceleration," arXiv preprint arXiv:2306.00978, 2023.

[13] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," arXiv preprint arXiv:2211.17192, 2022.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv:1503.02531, 2015.

[15] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, Y. S. Shao, and A. Gholami, "Full stack optimization of transformer inference: a survey," arXiv preprint arXiv:2302.14017, 2023.

[16] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, Z. Zhou, C. Xue, B. Wu, Z. Li, Q. Gu, Y. J. Lee, Y. Yan, B. Chen, G. Sun, and K. Keutzer, "LLM inference unveiled: Survey and Roofline Model insights," arXiv preprint arXiv:2402.16363, 2024.

[17] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," arXiv preprint arXiv:2305.05176, 2023.

[18] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, "RouteLLM: Learning to route LLMs with preference data," arXiv preprint arXiv:2406.18665, 2024.

[19] S. Deochake, "Cloud and AI infrastructure cost optimization: A comprehensive review of strategies and case studies," arXiv preprint arXiv:2307.12479, 2023.

[20] ——, "ABACUS: A FinOps service for cloud cost optimization," arXiv preprint arXiv:2501.14753, 2025.

[21] N. P. A. Vo, M. Kesarwani, R. Mahindru, and C. Narayanaswami, "FinOps agent—a use-case for IT infrastructure and cost optimization," arXiv preprint arXiv:2510.25914, 2025.

[22] H. Lee, L. Vu, and M. Jang, "Economics of spot instance service: A two-stage dynamic game approach," arXiv preprint arXiv:2305.19578, 2023.

[23] FinOps Foundation, "FinOps framework overview," https://www.finops.org/framework/, 2026, accessed 2026-07-12.

[24] W. S. Jevons, The Coal Question: An Inquiry Concerning the Progress of the Nation, and the Probable Exhaustion of Our Coal-Mines. Macmillan, 1865.

[25] S. Sorrell, "Jevons' Paradox revisited: The evidence for backfire from improved energy efficiency," Energy Policy, vol. 37, no. 4, pp. 1456–1469, 2009.
[26] B. Alcott, "Jevons' paradox," Ecological Economics, vol. 54, no. 1, pp. 9–21, 2005.
[27] A. S. Luccioni, E. Strubell, and K. Crawford, "From efficiency gains to rebound effects: The problem of Jevons' paradox in AI's polarized environmental debate," in Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency, 2025.
[28] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 3645–3650.
[29] D. Patterson, J. Gonzalez, Q. Le et al., "Carbon emissions and large neural network training," arXiv preprint arXiv:2104.10350, 2021.
[30] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun et al., "Sustainable AI: Environmental implications, challenges and opportunities," arXiv preprint arXiv:2111.00364, 2021.
[31] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, and J. Kepner, "From words to watts: Benchmarking the energy costs of large language model inference," arXiv preprint arXiv:2310.03003, 2023.
[32] OpenAI, "API pricing," https://developers.openai.com/api/docs/pricing, 2026, accessed 2026-07-12.
[33] Together AI, "Pricing — Together AI," https://www.together.ai/pricing, 2026, accessed 2026-07-12.
[34] GitHub, "GitHub Copilot plans," https://docs.github.com/en/copilot/get-started/plans, 2026, accessed 2026-07-12.
[35] RunPod, "GPU Cloud pricing," https://www.runpod.io/pricing, 2026, accessed 2026-07-12.
[36] OpenAI, "Terms of use," https://openai.com/policies/row-terms-of-use/, 2026, consumer terms governing ChatGPT plans; prohibits automatically or programmatically extracting data or output. The API is a separate product under separate terms and billing. Accessed 2026-07-18.
[37] Anthropic, "Use the Claude Agent SDK with your Claude plan," https://support.claude.com/en/articles/15036540-use-the-claude-agent-sdk-with-your-claude-plan, 2026, documents that Agent SDK, claude -p, and third-party app usage draw from the subscription's usage limits, and an announced change (paused 2026-06-15) metering Agent SDK usage through a separate monthly credit pool billed at standard API rates once exhausted. Accessed 2026-07-18.
[38] Codersera, "Qwen 3.5 + 3.6 + 3.7 Max: Alibaba open-weights guide (2026)," https://codersera.com/blog/qwen-3-5-complete-guide-2026/, 2026, secondary source documenting the Qwen family's open/closed split: Qwen 3.5/3.6 weights published under Apache-2.0; Qwen3.7-Max closed-weight, API-only (launched 2026-05-19). Accessed 2026-07-18.
[39] Vantage, "AWS EC2 p5.48xlarge pricing," https://instances.vantage.sh/aws/ec2/p5.48xlarge, 2026, secondary aggregator of AWS on-demand and reserved-instance rates; per-GPU figures in the text are normalized from this 8-GPU node price. Accessed 2026-07-12.
[40] Lambda, "GPU Cloud — on-demand H100 instances," https://lambda.ai/service/gpu-cloud, 2026, accessed 2026-07-12.
[41] Google, "Gemini API pricing," https://ai.google.dev/gemini-api/docs/pricing, 2026, accessed 2026-07-18.
[42] DeepSeek-AI, "DeepSeek-V4-Pro model card," https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro, 2026, MIT-licensed weights; 1.6T-parameter MoE, 49B active parameters. Accessed 2026-07-18.
[43] Z.ai, "GLM-5.2 model card," https://huggingface.co/zai-org/GLM-5.2, 2026, MIT-licensed weights; 753B-parameter MoE. Accessed 2026-07-18.
[44] OpenAI, "gpt-oss-120b model card," https://huggingface.co/openai/gpt-oss-120b, 2025, Apache-2.0-licensed weights; 117B-parameter MoE, 5.1B active parameters. Accessed 2026-07-18.
[45] Digital Applied, "Gemma 4 vs Llama 4 vs Mistral Small 4: Full comparison," https://www.digitalapplied.com/blog/gemma-4-vs-llama-4-vs-mistral-small-4-comparison, 2026, secondary source for Mistral Small 4 (119B-parameter MoE, Apache-2.0, released 2026-03) and Llama 4 Community License terms. Accessed 2026-07-18.
[46] Meta Platforms, "Llama 3.3 community license agreement," https://www.llama.com/llama3_3/license/, 2024, custom source-available license: commercial grant capped at 700 million monthly active users; incorporates an Acceptable Use Policy. Accessed 2026-07-18.
[47] NVIDIA Corporation, "NIM for LLMs benchmarking: Performance," https://docs.nvidia.com/nim/benchmarking/llm/latest/performance.html, 2026, tensorRT-LLM serving benchmarks on H100-80GB: Llama 3.1 8B Instruct (FP8, TP1) at 6,514.66 tok/s and Llama 3.1 70B (FP8, TP4) at 6,511.88 tok/s aggregate, measured at throughput-optimal batch-heavy operating points. Accessed 2026-07-11.