

Curation Over Breadth: Design and Measurement of a Self-Hostable, Availability-Aware AI Gateway

Sai Vivek Peddi
MLPal

Claude
benchmark harness & analysis

Technical report v2 — August 12, 2026

Measurements executed 11–12 August 2026 unless dated otherwise.

Code, harness, and raw results: github.com/ML-Pal/mlpal-gateway (Apache-2.0)

Abstract

AI gateways compete on catalog breadth: OpenRouter advertises 400+ models, Portkey 1,600+, LiteLLM 100+ providers [1,7,5]. We argue breadth is the wrong metric. Production models now retire on a ~ 12 -month cycle — Gemini 1.5 snapshots lived exactly 12 months, Claude Opus 4.1 was retired one year to the day after release, GPT-4.5-preview survived 4.5 months, and AWS Bedrock has made a 12-month platform lifetime *contractual* [9,11,12,14] — so a large catalog is mostly retired-or-dominated inventory, while gateway utility concentrates in whether current models are served well and references to them survive churn. We present **MLPal Gateway**, an open-source (Apache-2.0), self-hostable gateway built on this hypothesis: a curated catalog (73 models served in production) behind an Anthropic-wire core, *availability-aware router tags* that resolve a stable alias to the best model a deployment can actually serve, pass-through compute-unit metering, and per-key policy, budgets, and cache/latency observability. We measure it four ways. (i) **Gateway overhead**, isolated against a zero-latency fake upstream ($N=100$ per system): **8.5 ms** median added TTFT with the full admission pipeline enabled — faster than a LiteLLM proxy in both its bare configuration (13.2 ms) and a matched production configuration with database-backed virtual keys, budgets, and spend tracking (15.3 ms), with the tightest tail (p95 33 vs. 58/41 ms). We document the three engineering changes that took our overhead from 20.3 ms to 8.5 ms. (ii) **Client-observed latency**: a CloudFront edge cutover executed during this study reduced production median TTFT from 871 ms to **642 ms**; a same-night four-way benchmark places OpenRouter at 898 ms on the identical model and vantage. (iii) **Provider-semantics preservation**: prompt caching passes through byte-faithfully (22k-token prefix, write $\rightarrow$ read on all systems) and metered cost reproduces the provider’s list price to five decimals, falling $12.3\times$ on cache hits. (iv) **System-level utility**: a coding agent routing sub-tasks via the gateway catalog cut sub-agent cost $\sim 10\times$ at near-matched correctness. We release the gateway, console, SDK, and the full reproduction harness.

1 Introduction

An AI gateway multiplexes model providers behind one API and one billing relationship. The category has consolidated around a breadth race: OpenRouter’s pricing page advertises 400+ models across 70+ providers [1]; Portkey’s gateway README claims routing to 1,600+ models [7]; LiteLLM supports 100+ providers [5]. Model count is legible and easy to market — and, we argue, largely disconnected from delivered utility.

Our hypothesis is that **provider agnosticism matters, but breadth does not**: models are retiring at a record and accelerating pace (§2), so most of a large catalog is dead or dominated weight, while real traffic concentrates on a small set of current models per provider. What a practitioner needs from a gateway is (a) that current models are served correctly — valid parameter ranges, caching, streaming fidelity, provider quirks handled per model; (b) that their own code survives retirements without edits; and (c) that the cost and behavior of every request is observable. Each is a per-model investment that scales to tens of models, not thousands.

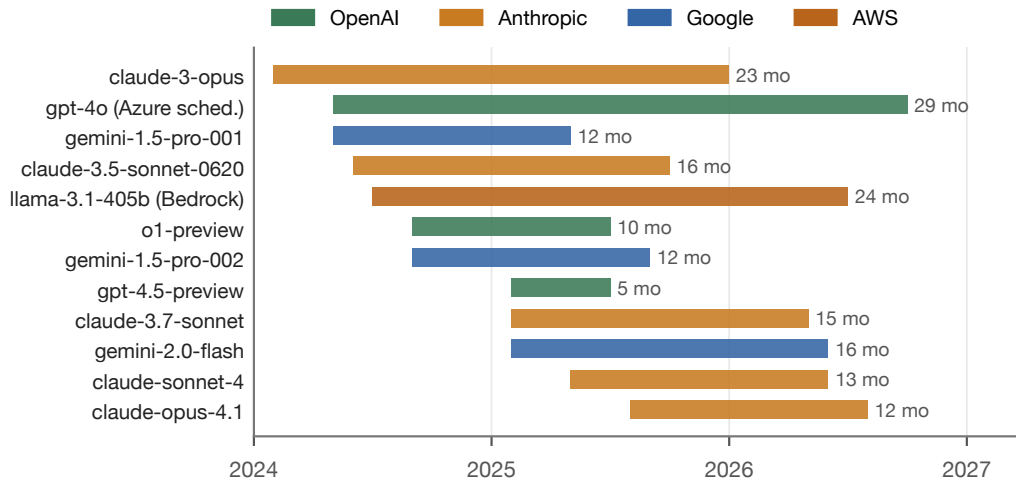


Figure 1: Launch→retirement spans for production API models, ordered by launch date, from official provider lifecycle pages [9,11,12,13,14] (gpt-4o shows Microsoft’s published Azure retirement schedule; Bedrock dates are platform-specific). Models launched in 2024 lived 22–29 months; models launched in 2025 live 12–16.

This report contributes:

1. A quantified case that model lifetimes have compressed toward ~12 months, from providers’ own depreciation ledgers (§2, Fig. 1).
2. The system design: an Anthropic-wire universal core with OpenAI-compatible endpoints, availability-aware router tags, pass-through compute-unit metering, and per-key admission — one codebase serving both a managed deployment and a self-hosted open-source distribution (§3).
3. Controlled measurements with a fair baseline: gateway overhead isolated against a zero-latency upstream, compared to LiteLLM in *both* bare and matched production configurations (§5.1); a client-observed latency decomposition and the measured effect of an edge cutover executed mid-study (§5.3); a same-night four-way comparison including OpenRouter (§5.4); cache and metering fidelity verified from the wire (§5.5); and system-level routing utility on a coding-agent workload (§5.6).
4. The performance-engineering record: the three changes that halved our hot-path overhead, each verified by trace inspection (§5.2).

2 The case for curation

Model lifetimes are compressing. Fig. 1 plots launch-to-retirement spans for twelve production API models from providers’ own lifecycle pages [9,11,12,13,14]. The 2024 cohort lived 22–29 months; the 2025 cohort converges on 12–16; GPT-4.5-preview — flagship-priced — survived **4.5 months** with a 91-day removal notice [9]. Google’s numbered Gemini snapshots retired exactly 12 months after release (1.5-pro-001: 2024-05-24→2025-05-24; -002: 2024-09-24→2025-09-24) [12]. Claude Opus 4.1 was retired one year to the day after release (2025-08-05→2026-08-05) with a ~61-day notice [11,16]. Notice windows compressed in parallel: Anthropic’s 2025 retirements carried ~181-day notices, its 2026 cohort ~61 days [11,16]; OpenAI’s policy floor is six months for GA models and *three* for specialized variants [9].

Churn is now contractual. AWS Bedrock’s lifecycle policy guarantees only 12 months of platform availability after launch, moves models to a Legacy state in which an account inactive on that model for 15 days loses access, and (for post-2026 EOLs) adds an extended-access phase

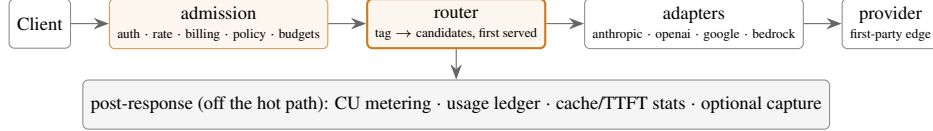


Figure 2: Request path. Admission runs before routing; adapters translate to each provider’s native API behind circuit breakers and relay SSE without re-chunking; metering, the usage ledger, and optional payload capture run after the response.

at *provider-set higher pricing* [14]. Deprecation is a scheduled property of the substrate, not an exceptional event.

Retirement breaks production. When OpenAI removed GPT-4o from ChatGPT at GPT-5’s launch with zero notice (2025-08-07), user revolt forced reinstatement within five days [10,15]; the February-2026 retirement wave left some business customers without model access for a week [10]. On Claude Opus 4.1’s retirement day, hard-coded model IDs failed simultaneously across Bedrock, Vertex, and the first-party API, and migration was complicated by a breaking parameter change in the successor [16].

Serving a model well is per-model work. Extracting a model’s best behavior requires model-specific handling a lowest-common-denominator passthrough cannot provide: reasoning-budget ranges that are discontinuous and differ per model (we maintain per-model valid ranges for Gemini’s `thinking_budget`); prompt-cache minimums that differ per model (a $\sim 2.5k$ -token prefix silently fails to cache on claude-haiku-4.5, whose minimum is 4,096 tokens — the request succeeds and bills full price); parameter support that changes across successors [16]; and per-provider streaming and error quirks. This scales to a curated set. The right unit of comparison is *utility per supported model*, not the count.

3 System design

3.1 One codebase, two deployments

The managed service (`models.mlpal.ai`) and the open-source distribution (`mlpal-gateway`) are the same codebase. Deployment differences are confined to composition-root *seams* — auth backend, billing backend, payload-capture default (on for self-hosters operating their own box; off in the managed service, which retains billing metrics and never stores payloads) — never forks. The distribution boots with one `docker compose` up; a catalog reconcile seeds the model registry, pricing, and routing feed; 341 unit tests and a console build gate CI.

3.2 Wire surface

The core endpoint is Anthropic-wire `POST /v1/messages` — request/response envelopes, SSE event grammar, tool use, and `cache_control` are Anthropic’s, for every provider behind the gateway. OpenAI-compatible endpoints (`/v1/chat/completions`, `embeddings`, `images`, `audio`) serve existing OpenAI-SDK code unchanged. Every response carries its metered cost in an `X-MLPal-Compute-Units` header and a routing envelope naming the resolved model.

3.3 Availability-aware router tags

A router tag (`mlpal`, `mlpal-flash`, `mlpal-lite`) maps to a priority-ordered, provider-spanning candidate list maintained in the catalog feed. Resolution walks the list and picks the first candidate the *local deployment* can serve (model active, not paused, provider key configured); the resolution is cached per instance and warmed at startup. A client that ships `"model": "mlpal"` survives every retirement in Fig. 1 without a code change, and the mechanism degrades gracefully to single-provider deployments: a box with only an Anthropic key still resolves every tag. The same curated intelligence is exposed declaratively as a ranked catalog (`GET /v1/catalog`) for clients that choose their own model per task; §5.6 measures that consumption mode.

3.4 Metering: pass-through compute units

Cost is metered in compute units pegged at 1 CU = \$10 of provider list price: $CU = \sum_c \text{tokens}_c \cdot \text{rate}_c / 10$ over billing components c (input; output; cache-write at the provider’s $1.25\times$ input rate; cache-read at $0.1\times$). There is no markup term: self-hosters pay providers directly, and the managed tier bills tokens at cost with a flat platform fee, so the meter is an instrument, not a margin. The ledger stores CU at `numeric(24, 12)`; §5.5 verifies the meter against the wire. Per-key spend budgets (multiple sliding windows, checked at admission, never cutting a running stream) and model-policy globs complete the admission pipeline; per-key aggregates expose cache hit rate, latency p50/p95, time-to-first-token, and stream share.

4 Experimental setup

Overhead study (§5.1). To isolate gateway overhead from provider variance, all gateways are pointed at a *fake upstream*: a local HTTP server answering `/v1/messages` with a canned, valid Anthropic streaming response (14 SSE events, 9 content deltas) with zero think time. Four systems, $N=100$ streaming trials each, order-rotated: (a) the fake upstream called directly (client+server floor); (b) **MLPal Gateway** (Docker, full admission: key auth, Redis rate limiting, billing gate, model policy, budget windows, metering, usage persistence, payload capture on); (c) **LiteLLM bare** (`ghcr.io/berriai/litellm:main-latest`, pulled 2026-08-11; static master-key auth, no database); (d) **LiteLLM production config**: the same image with PostgreSQL-backed virtual keys — the benchmark authenticates with a generated virtual key carrying a model allowlist, budget, and rpm limit, with spend tracking enabled. Configuration (d) is the fair comparison: it is LiteLLM doing the same class of per-request work as (b).

Client-observed study (§5.3–5.4). Same protocol as our July study [17]: identical prompt (~150-word CAP-theorem task), `max_tokens` 256, streaming, one cold connection per request, one unmeasured warmup, $N=15$ per system, A/B rotation, 0.5 s spacing, model snapshot `claude-haiku-4-5-20251001` everywhere (OpenRouter: `anthropic/claude-haiku-4.5`, OpenAI wire). Metrics: TTFB (first SSE byte), TTFT (first content token), total, content-chunk count. Client: Apple-silicon macOS, Python `httpx`, residential network. Server-side ground truth comes from AWS X-Ray traces of the production service.

Cache experiment (§5.5). A ~22k-token system prefix marked `cache_control: ephemeral` (above haiku-4.5’s 4,096-token minimum), distinct per system so systems cannot share provider cache entries; four sequential non-streaming trials per system against the real provider; usage fields read from the wire; metering fidelity verified by reading the CU header on a cache-write and cache-read call of the same prefix.

5 Results

Table 1 orients the section: each comparison is like-for-like — our open-source gateway against the open-source proxy a self-hoster would otherwise deploy, and our managed service against the managed gateway a team would otherwise buy — plus the raw-provider floor that bounds what any gateway can achieve.

5.1 Self-hosted head-to-head: MLPal Gateway vs. LiteLLM (gateway overhead)

Table 2 and Fig. 3 give the central result: with authentication, rate limiting, billing gate, model policy, budget windows, metering, and payload capture all enabled, MLPal Gateway adds **8.5 ms** at the median — less than LiteLLM checks a single static string for (13.2 ms), and 6.8 ms less than LiteLLM doing comparable per-request work (15.3 ms). The tail is more decisive: p95 of 33 ms vs. 58 ms (bare) and 41 ms (production). Against a real ~3 s completion, all of these are noise; the result matters because it removes the standing excuse that governance costs latency — admission-time policy, budgets, and metering are computationally free at gateway scale.

Table 1: The comparisons in this report, like-for-like. Each row names the deployment class, the contender, and the headline result; details in the referenced sections.

Matchup	Systems	Headline result
Self-hosted vs. self-hosted (§5.1)	MLPal Gateway (OSS) vs. LiteLLM proxy (OSS), bare and production configs	MLPal +8.5 ms overhead with full admission vs. +13.2 (bare) / +15.3 (production); p95 33 vs. 58 / 41 ms ($N=100$)
Managed vs. managed (§5.4)	<code>models.mlpal.ai</code> vs. OpenRouter, same model/night/vantage	MLPal 642 ms median TTFT vs. 898; p95 957 vs. 1,424 ($N=15$); OpenRouter streams finer chunks (93 vs. 9)
Managed vs. raw provider (§5.3, §5.4)	<code>models.mlpal.ai</code> vs. direct <code>api.anthropic.com</code>	+99 ms at the median (642 vs. 543) buys routing, metering, policy, and observability
Semantics fidelity (§5.5)	MLPal, LiteLLM, direct	prompt caching byte-faithful on all three; MlPal’s meter reproduces list price to five decimals
Broader study [17]	MLPal vs. OpenRouter, 11 model pairs (July 2026)	MLPal won median TTFT 8/11 with tighter tails; OpenRouter won throughput and fast/cheap models; neither dominated

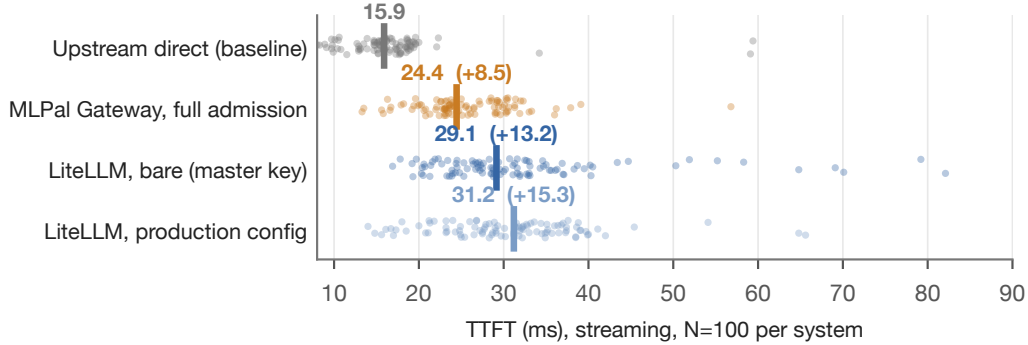


Figure 3: Added latency isolated: per-trial TTFT against a zero-latency fake upstream ($N=100$ per system, streaming; bar = median, label = median (delta vs. baseline)). MlPal Gateway with its full admission pipeline undercuts LiteLLM in both its bare (master-key) and production (database-backed virtual keys, budgets, spend tracking) configurations.

5.2 How the overhead was halved

Our first measurement of this experiment gave **20.3 ms** — *slower* than bare LiteLLM (13.4 ms). Three changes, each identified from AWS X-Ray traces of the production service and re-verified after deployment, took it to 8.5 ms:

1. **Connection pooling to the provider.** The Anthropic edge constructed a new HTTP client — and therefore a fresh TCP+TLS handshake to the provider — on *every request*. Replaced with a per-event-loop pooled client. This is the largest single saving and also removes a per-request handshake from the production path to `api.anthropic.com`.
2. **Pure-ASGI observability.** Two Starlette `BaseHTTPMiddleware` layers each wrapped every response in an extra task and memory stream (a scheduling hop on every streamed chunk, including the first token), and one parsed the *entire* request body with `json.loads` solely to read the model name for a metrics label — a double full-body parse on multi-hundred-kilobyte agentic payloads. Replaced by a single pure-ASGI middleware that never buffers or re-parses the body (the model dimension comes from a bytes-regex on the first chunk) and emits metrics only after the response has started.

Table 2: Overhead study, $N=100$ streaming trials per system against the fake upstream (medians; Δ vs. the direct-call baseline). All three gateways preserve the 9-chunk stream exactly.

System	TTFT med (ms)	Δ med	p95	total med
Upstream direct (baseline)	15.9	—	20.0	16.2
MLPal Gateway, full admission	24.4	+8.5	33.3	30.4
LiteLLM, bare (master key)	29.1	+13.2	58.3	29.4
LiteLLM, production config	31.2	+15.3	41.1	31.5

Table 3: Network-layer cost to each host from the client vantage (median of 5 cold probes, 2026-08-11), before and after the CloudFront cutover executed during this study.

Host	TCP (ms)	TLS done (ms)	/health TTFB (ms)
models.mlpal.ai (direct ALB, before)	75	250	380
models.mlpal.ai (CloudFront, after)	11	32	94
api.anthropic.com (reference)	12	26	40

- 3. Timer-free relay.** The SSE relay awaited `wait_for(queue.get(), timeout)` per chunk — a timer plus exception-based control flow on every event — to inject keepalive pings during provider silence. Replaced by a single idle-ping task; chunks now cross one queue hop with no timer.

Post-deployment X-Ray traces confirm the structure: all admission I/O completes by +8.6 ms (the previously serialized Redis calls now visibly overlap; budget windows are read in one pipelined round trip), the provider call fires at +19.5 ms, and metering/accrual appear only after the response. An isolated probe of the admission pipeline (full stack, no provider call) measures **2.7 ms** median on the local box and ~ 12 ms warm on production (in-VPC Redis/database hops).

5.3 Client-observed latency and the edge cutover

From this vantage, the managed service’s latency gap over direct provider access was dominated by geography, not software: a single-region ALB cost ~ 75 ms TCP and ~ 250 ms TLS per cold connection, against a CDN edge $\sim 12/26$ ms away (Table 3); the service itself contributes ~ 20 ms (§5.2). Our July report [17] predicted that edge termination would recover this tax; during this study we executed the cutover — `models.mlpal.ai` now resolves to CloudFront in front of the same ALB, with streaming validated end-to-end before the DNS change (the gateway’s 15 s SSE heartbeat keeps long streams under CloudFront’s 60 s origin idle timeout). Measured effect on the full request path ($N=15$, same night): median TTFT **871 ms** $\rightarrow$ **642 ms**, p95 1,357–2,613 $\rightarrow$ 957 ms. Existing clients required no changes.

5.4 Managed head-to-head: MLPal vs. OpenRouter, same night

With an OpenRouter key supplied for this study we re-ran the head-to-head on the same model, vantage, and evening (Table 4). Post-cutover, the managed service leads OpenRouter by ~ 250 ms at the median and ~ 470 ms at p95 on this model, and sits ~ 100 ms behind direct provider access — the residual being client $\rightarrow$ edge $\rightarrow$ us-east-2 $\rightarrow$ provider geography plus ~ 20 ms of service time. OpenRouter retains the finer-grained stream (93 vs. 9 chunks; upstream Anthropic chunks coarsely on our route, a perceived-smoothness advantage for OpenRouter in interactive UIs). Our July study [17], with its eleven model pairs, remains the broader comparison: OpenRouter won median TTFT on fast/cheap models near its edge and sustained higher steady-stream throughput, while MLPal won every Anthropic pair with substantially tighter tails; neither system dominated. Those results predate both this study’s engineering and OpenRouter’s own evolution since July.

5.5 Provider semantics survive the hop: caching and metering fidelity

Prompt caching is the highest-leverage provider feature a gateway can break silently: reordering or rewriting the prefix destroys cache identity and the client pays full price with no error. All three

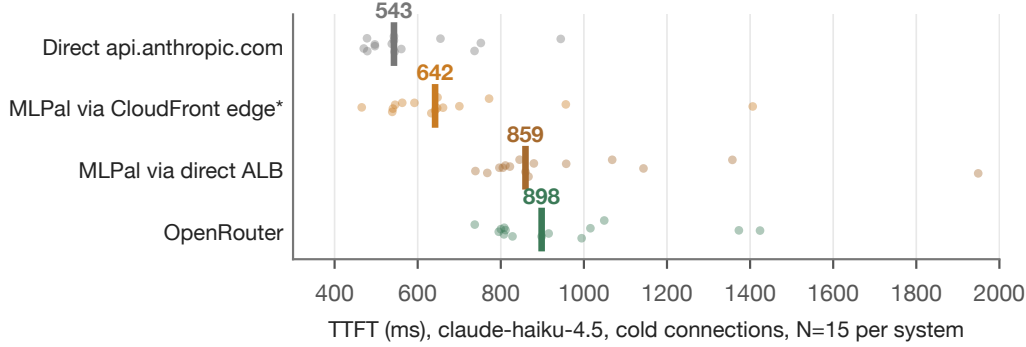


Figure 4: Client-observed TTFT on `claude-haiku-4.5`, cold connections, $N=15$ per system, all runs within the same evening from the same client. *The CloudFront row was measured ~ 40 minutes after the other three (post-cutover); Anthropic-side variance was elevated all evening, so cross-row deltas carry $\sim \pm 50$ ms of session noise.

Table 4: Same-night comparison ($N=15$ each, streaming, cold connections, identical prompt and model snapshot). OpenRouter serves ~ 93 fine-grained chunks vs. Anthropic’s ~ 9 coarse chunks through both MLPal paths — the chunking difference of [17] persists.

Path	TTFT med (ms)	p95	total med	chunks med
<code>Direct api.anthropic.com</code>	543	944	3,051	9
MLPal via CloudFront edge	642	957	3,134	9
MLPal via direct ALB (pre-cutover)	859	1,357	3,459	10
OpenRouter	898	1,424	3,155	93

systems preserved it exactly (write of 21,213–22,013 tokens on trial 1; reads of the identical count after; Fig. 5). On MLPal, the meter tracked it end-to-end: the cache-write call metered **0.002756 CU** and the cache-read call **0.000224 CU** — a **12.3 $\times$** reduction, each figure reproducing Anthropic’s list price exactly ($22,013 \times 1.25 \times \$1/\text{M} \div \$10 = 0.00275$; $\times 0.1$ for reads = 0.00022) [19]. The meter is verified from the wire, not from our own configuration. Cache read/write tokens land in each usage row; per-key cache hit rate is a first-class dashboard metric (an agentic coding workload on our development deployment sustains 44%).

5.6 Utility over breadth at the system level

The curation thesis predicts that a ranked, current catalog beats a large flat one for real workloads. A measurement from our coding agent (yodex, 2026-07-28 [18]): on a four-module build with delegation, the agent consulted the gateway catalog to route each sub-task by rated complexity — trivial modules to `gemini-2.5-flash-lite`, harder ones upward — cutting sub-agent cost $\sim 10\times$ ($0.212 \rightarrow 0.022$ CU) versus inheriting the pinned frontier model, at near-matched correctness (11/12 vs. 12/12), and $\sim 2\times$ cheaper end-to-end than a same-model Claude Code baseline. The enabling primitive is a catalog small enough to rank honestly and current enough to trust.

6 Feature comparison

None of these systems dominates (Table 5). OpenRouter’s multi-provider failover, auto-router, and compliance controls have no MLPal equivalent; LiteLLM’s provider matrix is far larger and its OSS proxy offers key management and budgets of the same class as ours (§5.1 measures both); Helicone’s observability product is deeper than our console. MLPal’s distinct positions are the Anthropic-wire universal core, curated availability-aware tags, admission-time enforcement measured to cost < 9 ms with the tightest tail among tested gateways, and a meter verifiable against provider list price on the wire.

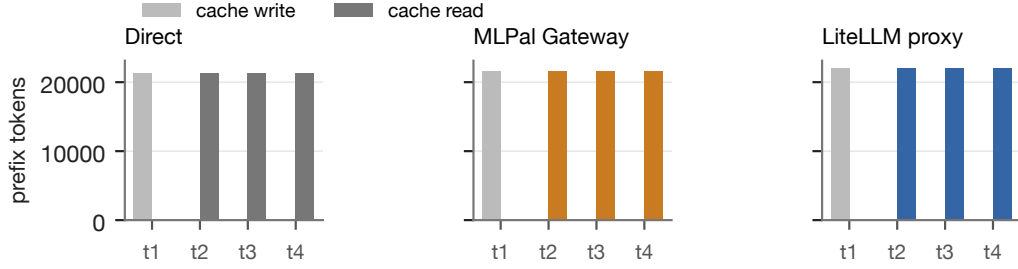


Figure 5: Prompt-cache passthrough: $\sim 22k$ -token prefix with `cache_control`, four sequential trials per system, distinct prefixes per system. Every system shows the canonical pattern — full write on trial 1, full read on trials 2–4.

Table 5: Feature surfaces from current public documentation (retrieved 2026-08-11) [1,2,4,5,6,7,8] and MLPal source [3]. “—” = not offered/documented; *italics* = the vendor’s hosted/enterprise tier rather than the OSS artifact. LiteLLM’s OSS proxy *does* provide key management, budgets, and callback-based logging (§4, config d); the differences below are of integration and defaults, not existence.

Dimension	OpenRouter	LiteLLM (OSS)	Portkey (OSS)	Helicone	MLPal Gateway
Catalog	400+ models, 70+ providers	100+ providers	1,600+ claimed models	100+ models	55 curated entries; 73 served (managed)
Self-host	—	MIT	MIT	Apache-2.0 (cloud-primary)	Apache-2.0 core
Anthropic-wire end-point	— (OpenAI wire + Responses beta)	yes	not documented	not documented	core surface, all providers
Fee model	5.5% on credits	free / quote enterprise	free / \$49+ hosted logs	0% + Stripe	free self-host; managed: tokens at cost + flat fee
Stable model alias	auto-router	operator-defined config	configs/routing rules	—	curated, availability-aware, feed-updated
Multi-provider fallback	yes, price-weighted	yes (strategies)	yes	yes	— (one pinned first-party edge per model)
Per-request cost on wire	via /generation lookup	cost tracking (DB)	<i>hosted</i>	registry-computed	CU header, every response, meter-verified (§5.5)
Per-key cache/TTFT/tail metrics	activity dashboard	via callbacks	<i>hosted</i>	core product	built-in per key
Spend controls	credit limits	budgets per key/team	<i>hosted</i>	tier limits	multi-window budgets + policy globs, admission-time

7 Limitations and conflict of interest

All measurements come from one client machine across one evening; $N=15$ per cell for client-observed runs makes medians robust but p95 coarse; one prompt shape and one model family for the latency studies. Provider-side variance was elevated on the measurement evening (direct-call p95 ranged 634–944 ms across sessions) and dominates client-observed deltas; the overhead study (§5.1) exists precisely to factor it out, at the cost of using a synthetic upstream that understates payload sizes. The fake-upstream floor includes client and server scheduling, so absolute overheads are upper bounds. LiteLLM’s production configuration enables virtual keys, budgets, rpm limits, and spend tracking but not logging callbacks; a callback-laden deployment would be slower, an untuned comparison in our favor we did not make. The Table 4 edge row was measured ~ 40 minutes after the other rows. Catalog counts across vendors are not directly comparable. Output quality was not evaluated — only delivery, semantics preservation, and cost. This report is written by the system’s authors; the harness, raw JSON, figure code, and the fake upstream ship in the repository so every number can be re-derived.

8 Conclusion

Breadth is the legible metric, but the substrate has changed under it: when models live ~ 12 months and churn is contractual, a catalog’s value decays at the rate its entries retire, while the properties that compound — stable aliases that survive retirement, per-model tuning, faithful caching, verifiable metering, admission-time governance — are per-model investments that favor curation. This report adds a second claim, measured rather than argued: *governance is not a latency trade*. With every admission feature enabled, the gateway adds 8.5 ms at the median — less than a bare translation proxy — and after an edge cutover executed mid-study, the managed deployment serves the same model ~ 250 ms faster than OpenRouter from the same vantage. The gateway, console, SDK, and this paper’s full harness are open source at github.com/ML-Pal/mlpal-gateway.

References

1. OpenRouter, “Pricing.” openrouter.ai/pricing (retrieved 2026-08-11).
2. OpenRouter, “Provider selection & routing.” openrouter.ai/docs/guides/routing/provider-selection.
3. MLPal, `mlpal-gateway` source. github.com/ML-Pal/mlpal-gateway.
4. OpenRouter, “Zero completion insurance.” openrouter.ai/docs/guides/features/zero-completion-insurance.
5. LiteLLM, proxy documentation: load balancing, routing, virtual keys. docs.litellm.ai.
6. LiteLLM, “v1/messages (Anthropic unified endpoint).” docs.litellm.ai/docs/anthropic_unified.
7. Portkey, “AI Gateway” README. github.com/portkey-ai/gateway (retrieved 2026-08-11).
8. Helicone, “AI Gateway”; credits (0% markup). docs.helicone.ai/gateway/overview; helicone.ai/credits.
9. OpenAI, “Deprecations.” developers.openai.com/api/docs/deprecations (retrieved 2026-08-11).
10. OpenAI, “Retiring GPT-4o and older models” (2026-02-13). openai.com/index/retiring-gpt-4o-and-older-models. TechRadar (2026-02).
11. Anthropic, “Model deprecations.” platform.claude.com/docs/en/about-claude/model-deprecations (retrieved 2026-08-11).
12. Google Cloud, “Model versions and lifecycle.” docs.cloud.google.com/gemini-enterprise-agent-platform/models/model-versions.
13. Microsoft Azure, “Model retirement schedule.” learn.microsoft.com/azure/foundry/openai/concepts/model-retirement-schedule.
14. AWS, “Amazon Bedrock model lifecycle.” docs.aws.amazon.com/bedrock/latest/userguide/model-lifecycle.html.
15. Ars Technica, “OpenAI brings back GPT-4o after user revolt” (2025-08).
16. TheRouter, “Anthropic model deprecation cadence” (third-party; key dates cross-checked against [11]). therouter.ai.
17. Peddi & Claude, “When the Edge Isn’t Enough: A Head-to-Head Latency and Capability Study of Two LLM Gateways.” MLPal technical report, 2026-07-18.
18. MLPal, “Yodex vs Claude Code on claude-opus-5” (2026-07-28). github.com/ML-Pal/yodex.
19. Anthropic, “Prompt caching.” platform.claude.com/docs/en/build-with-claude/prompt-caching.

A Reproduction

`paper/bench/` in the repository contains the harness (`harness.py`, `cache_exp.py`), the fake upstream (`fake_anthropic.py`), both LiteLLM configurations, raw results (`results-*.json`), and figure code. Overhead study: start the fake upstream, boot the gateway with `docker compose up` and `MLPAL_ANTHROPIC_BASE_URL` pointed at it, start LiteLLM bare and, for the production configuration, with `DATABASE_URL` set and a virtual key generated via `/key/generate` with a model allowlist, budget, and rpm limit; then `python harness.py 100 fake,mlpal,litellm,litellm_full out.json`. Client-observed rows require only API keys. Model snapshot `claude-haiku-4-5-20251001`; client macOS/Apple silicon, Python 3.13, 2026-08-11/12.