An AI Gateway for Research Labs

Keys, Allowances and Kill Switches

Charles E. Forsyth III
Research Computing
University of California, Riverside
Riverside, California, USA
charles.forsyth@ucr.edu

Abstract

Since September 2026 UC Riverside (UCR) Research Computing has run a pilot AI gateway: one web address and one personal key that reach Google's Gemini models, partner and open-weight models hosted on Google Vertex AI, and open models on the National Research Platform. It is a proof of concept. As of 23 October it served 34 model deployments to 13 keys, including three pilot labs. It runs the open-source LiteLLM proxy, v1.104.0, on Google Cloud Run, with a small admin server I call aigw in front that turns the proxy's teams and keys into labs, members and allowances.

Two lessons carry over to anyone building one. First, keep the admin layer out of the request path and let the proxy enforce limits, so a bug in your own code cannot spend money; but know that the proxy's budgets are checked before each call and can be overshot. Second, probe the real system before trusting a setting's name. A wildcard, a "disabled" console and a test fake each did something other than their label.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **Security and privacy** → *Access control*; • **Social and professional topics** → *Management of computing and information systems.*

Keywords

AI gateway, LiteLLM, research computing, budgets, rate limiting, API keys, kill switch, Model Context Protocol

Author note

This is a Field Note, a dated report on a system I built. The work was directed by me and carried out with AI agents under written briefs, independent checks and explicit approval steps. Figures come from the gateway's system specification and change log, the aigw specification and changelog, the 23 October load test results, and git history and tags. This note was drafted with AI assistance from those records and reviewed by me. All numbers are real unless marked. Names and addresses are representative, and per-lab allowances and key budgets are omitted rather than replaced.

Cite this note. C. E. Forsyth III. 2026. An AI Gateway for Research Labs. Field Notes. doi:10.5281/zenodo.23752275

1 Context

Without a gateway, a researcher who wants AI models finds their own way: cloud credits, a department recharge, or a credit card. None of those lets anyone see the total or stop a runaway script fast, and nobody can say for sure where the data went. For Research Computing it is the same job as compute and storage: share a resource fairly and keep the controls in one place.

The gateway started as an internal proof of concept run with UCR Information Technology Solutions (ITS), used by me and five ITS staff testers, and its specification still describes it that way. The pilot labs are small: one is my own group, one is a research lab, and one has been set up but not used. The longer-term plan is AI model access with an allowance per lab as part of a Tier 1 research computing service. That needs a people layer the proxy does not have: who belongs to which lab, who may change what, and a safe way to hand someone a key. This note covers both halves, and where they are still thin.

2 Related Work

Researchers buying their own cloud and software outside central IT is an old pattern. Silic and Back's study of "shadow IT" found it widespread [1]; per-researcher AI subscriptions and API accounts are a new instance of the same thing. Research computing's usual answer is a shared resource with fair allocation and central controls, as batch schedulers such as Slurm do for cluster nodes, with per-account limits and accounting [2]. The gateway applies that model to model calls: allowances in place of core-hours.

The background on governing research cloud spending, including managed services such as CloudBank, is in the companion note on governing cloud projects [3]. The FinOps Framework names the practices of allocating, monitoring and reconciling cloud cost [4]. The gateway's split, a computed value per call for researchers and the billing export for staff, follows that practice of showing use promptly and reconciling to the invoice afterward. Google's own controls are coarser: a budget sends alerts and does not by itself stop use, and Google documents that reports arrive with a delay [5]. Its sample for capping cost is a function that disables billing on a notification [6]. The gateway's widest brake uses the same trigger but turns off only Vertex AI, which leaves the project and its data in place.

The gateway's key rules draw on standard security principles. Short expiries, per-key model lists and narrow route permissions follow Saltzer and Schroeder's principles of least privilege and fail-safe defaults [7]. NIST's zero trust architecture argues for granting access per request against an authenticated identity rather than network location [8], which fits a public endpoint that rests on keys. The claim link adds a sign-in check at the moment a key is created. Data limits follow the University of California's classification of institutional information into protection levels P1 to P4 [9].

Two results explain the gateway's limits. Brewer's account of CAP describes the trade between consistency and availability when

state is spread across nodes [10]; spend counters held per instance and written in batches choose availability, which is why budgets can be overshot. Little's law ties requests in flight to throughput and wait [11]: once throughput is capped, the wait must grow in proportion to the queue, the shape seen in the load test once concurrency passed about 20. Serving engines such as vLLM, whose PagedAttention memory management raises how many requests one GPU can batch at once [12], work below the gateway; the gateway does none of that work and only routes and meters calls.

The software choices are documented in their projects. LiteLLM is an open-source proxy that offers the OpenAI API in front of many providers [13], with team, member and key budgets, budget durations and rate limits [14]. Envoy AI Gateway is an open alternative built on Envoy [15]. The Model Context Protocol defines how AI clients discover and call a server's tools [16]. The open models come from the National Research Platform [17], which grew out of the Pacific Research Platform [18]. I found no published account of a per-lab AI allowance service at a university, but my search was not exhaustive, and I make no novelty claim.

3 The Design

The core is LiteLLM [13], an open-source proxy that speaks the OpenAI application programming interface (API) on one side and many model providers on the other. A client sends a request with a key. LiteLLM checks the key, checks that the key may use that model, checks its budgets and rate limits, forwards the call, and writes one row to a Postgres log with the tokens, the computed value of the call, the latency and the outcome. It does not store prompt or response text in that log. I tested that directly on 4 October: the `messages` and `response` fields in the spend log came back empty.

Figure 1 shows the parts and how a request flows through them.

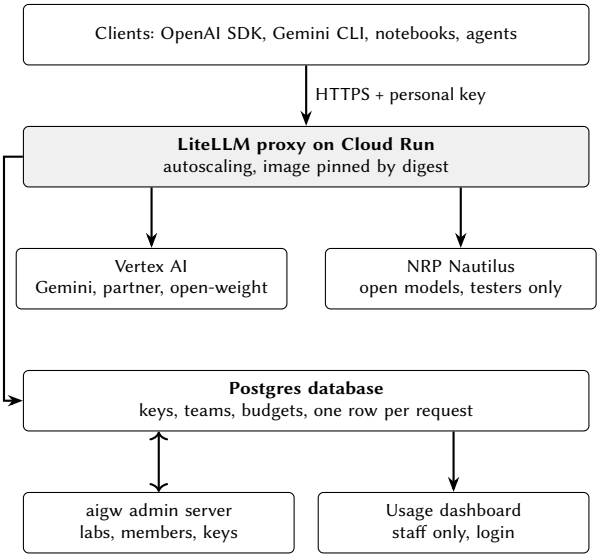


Figure 1: The gateway's parts. Clients talk only to the proxy, which logs each request to Postgres; aigw reads and writes the proxy's database and the dashboard reads it. Neither handles model traffic.

Google's own platform first. Vertex AI calls go out under the gateway's service account, so no Google provider key is stored. Gemini models, Google embeddings, Anthropic's Claude models offered as Vertex partner models, and the open-weight `gpt-oss` models are reached this way. Claude is on a subset of staff and tester keys only; keys issued to labs carry Google models.

Open models from a national platform. The gateway lists 13 models from the National Research Platform (NRP) Nautilus cluster [17], a national Kubernetes cluster for research and education, each listed with an `nrp-` prefix. They include Qwen3, Gemma, GLM-5 and DeepSeek. NRP does not bill for them, so they log as zero. For now only the testers can call them, because all calls share one NRP credential and its fair-use limits. I wrote about getting onto Nautilus in an earlier field note [19].

A plain open-source build. The gateway runs the community LiteLLM image with no enterprise license key. Some features (team admins, organizations, key regeneration) need a license. I probed each one on 4 October against throwaway objects and found the features I needed worked without one: team budgets, per-member budgets, several budget windows on one key, block and unblock, and daily activity reports.

One source of record. The model list and the dashboard live as files in a private repository and as stored secret versions that must stay byte-identical to them. A drift-check runbook compares the two; the last recorded check, on 23 October, found both identical.

The gateway logs a value for every call from a price table in its configuration, but the Google billing export is the authority for real dollars. The two drift apart when credits apply, when rounding differs, or when a provider changes a price before the table is updated. So the number a researcher sees is a measure of use against an allocation, and staff reconcile it with billing [4].

4 Allowances and Caps

A lab's AI allowance is an allocation. ITS pays the bill; the researcher does not. LiteLLM already enforces budgets [14], so aigw maps the people layer onto LiteLLM objects and keeps no database of its own. A lab is a LiteLLM team, and the allowance is the team's budget. A member's cap is their budget inside the team. Each key then has its own daily and monthly windows. Figure 2 shows the nesting.

The reset needs care. LiteLLM budgets use a duration, here 30 days, not a calendar month [14]. When I created the first pilot lab, the proxy set its first reset to 00:00 Coordinated Universal Time (UTC) on the 1st of the next month. After that the window counts 30 days from the last reset, so over a year it drifts away from the 1st. I describe the allowance as "about monthly" until I have watched a few resets.

Every layer is enforced by LiteLLM at request time. When a key passed its daily window in testing, the next call came back HTTP 422 with "Key over 24h budget". aigw only sets the numbers and never sits in the request path, so a bug in aigw cannot raise a limit by itself.

That does not make the limits exact. LiteLLM checks a budget before each call against the spend it already knows, and it only learns a call's cost after the response. Several calls in flight at once, or one long streamed answer, can all pass the check and land past

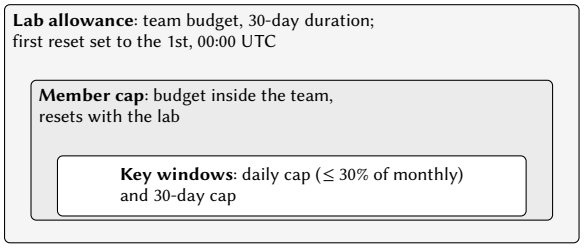


Figure 2: The three budget layers. The first limit reached refuses the call.

the limit. Spend counters are also held per instance and written to the database in batches; with no shared cache, two or more instances can each think there is room, a choice of availability over consistency [10]. The overshoot is bounded by the parallel-request limit on each key times the cost of its largest call, which is one reason every key carries a parallel limit. I have not measured the worst case under load.

The defaults are modest, and nothing is issued uncapped. A member added without a cap gets the lab's default, or 2% of the allowance if the lab has none. Member caps may add up to more than the allowance, because most members use little; the allowance still stops use when it runs out. A cap can never be set above the allowance, and aigw refuses a monthly allowance over a set ceiling as a guard against a typed extra zero.

A principal investigator (PI) sees each member's use by model and by day. Nobody, including me, sees prompts, because the gateway does not keep them. How we chose to show these numbers to researchers, as use against an allocation rather than a bill, belongs to the wider portal work and is covered in the field note on governing cloud projects [3].

5 How a Key Reaches a Person

Before aigw, I made keys with a Python command-line tool and the gateway's master key, then emailed each one. That does not scale to labs, and a key in an inbox can be found later. aigw replaced it with a claim link. Issuing a key does not create a key. It opens a claim that names the person, the lab, the caps and the expiry. The PI or I send the link however we like. The person opens it, signs in with their UCR Google account, accepts the terms of use, and presses a button. Only then is the key made on the gateway, and it is shown once in that browser.

- The link carries only a claim id. A forwarded or intercepted link is useless to anyone not signed in as the named person.
- Opening the link never makes a key. Minting needs a form post with a per-session token, so a mail scanner or link preview cannot trigger it.
- A claim works once and expires after 72 h. At collection aigw re-reads the lab, so a person removed in the meantime gets nothing.
- The terms come first. Data on the gateway is limited to UC protection levels P1 and P2 [9]; P3 and P4 need written approval from the Information Security Office. Nobody can accept for someone else. This is an attestation, not a technical control.
- Every key expires after 90 days, has a daily cap of at most 30% of its monthly cap, and carries limits on tokens per minute, requests per minute and parallel requests [14]. Every lab also has a shared ceiling.
- Keys issued through aigw carry only Google models, and are limited to model calls plus reading their own key information, so they cannot list the rest of the catalog.

aigw itself is a Model Context Protocol (MCP) server [16], an endpoint that AI clients can call as a set of tools. PIs and staff already work in those clients, and one server can hold every rule. A web front end can sit on top later as a client with a capped role instead of a second copy of the rules. aigw has 37 tools; a member sees 10 of them, a viewer 21 and an admin all 37. Any change that commits money or removes access is two-step: the first call returns a plan and a confirm token bound to that person, that tool and those exact arguments, valid once for 10 minutes.

aigw's own key to the gateway cannot spend anything. It has a $0 budget, no models, and an exact list of admin routes. On every call aigw re-reads that key and reports itself unhealthy if the key ever gains a model, a budget or a route outside the list.

6 Brakes and the Kill Switch

Use can be stopped at seven levels, from one key to the whole cloud project. Table 1 lists them from narrowest to widest. Each works without the others, so if one is misconfigured the next still holds.

Engaging the kill switch takes a reason and a confirm step. It records the switch first, so issuing and collecting stop even if a later block call fails. Then it blocks every key it is allowed to and writes down exactly which ones. Release unblocks exactly that list. Keys that were blocked for some other reason stay blocked, and keys in a lab that was frozen in the meantime stay frozen until the lab is unfrozen. If aigw cannot read the switch's state, it treats the switch as engaged, a fail-safe default [7].

The lab freeze does the same thing for one lab. I expect PIs to use it when a key leaks and they cannot tell which one, when a script will not stop, or when a student leaves.

The brakes are less independent than the table makes them look. The first six all act through the proxy and its database, so a failure there takes them down together. Only the cloud billing brake sits outside [6], and it is slow: billing data reaches the budget alert with a delay of hours [5], and when it fires it turns off Vertex AI for everyone, not one lab. It catches a runaway month, hours late.

None of these brakes has been pulled for real. The kill switch and the lab freeze were built on 5 October, before the first pilot key went out. Tests cover them, and no incident is recorded in the change logs.

7 Load and Advisories

On 23 October I ran a stepped load test against the live gateway with a test key: one-word replies from `gemini-2.5-flash-lite`, rising from 2 to 80 requests in flight, for 20 to 32 s per step. Table 2 has the measured results. All 4,694 requests succeeded.

Up to about 20 requests in flight, throughput rose with load and the median wait stayed near half a second, which is mostly the model's own response time. Past that, throughput flattened near 42 per second and the wait grew in step with the queue, as Little's

Table 1: Brakes on use, from narrowest to widest.

Brake	Who pulls it	What it does
Key windows and rate limits	Automatic	Refuses a key's request past its daily or monthly cap, or its token, request or parallel limits
Report a leak	The key's owner	Blocks the owner's own key at once, no confirm step
Block or revoke a key	PI, delegate or admin	A blocked key gets HTTP 401 on its next call; revoking deletes it
Member cap and lab allowance	Automatic	Refuses requests once the member or the lab runs out
Lab freeze	PI or admin	Blocks every key in one lab and stops new keys there; undo restores exactly those keys
Kill switch	Admin only	Blocks every key on the gateway except service keys and the admin's own, and stops all issuing
Cloud billing budget and throttle	Automatic	A Google Cloud budget on the project triggers a function that turns off Vertex AI for the whole gateway if the cap is crossed

Table 2: Load test on 23 October 2026, one Cloud Run instance. Latencies in seconds.

In flight	Requests	Req/s	Median	p95	Errors
2	80	4.0	0.49	0.62	0
5	311	10.3	0.48	0.60	0
10	607	19.9	0.49	0.61	0
20	1,086	35.5	0.53	0.71	0
40	1,272	41.3	0.95	1.16	0
80	1,338	42.2	1.87	2.10	0

law predicts [11]: 80 in flight at 42.2 per second gives about 1.9 s, against a measured median of 1.87. None failed. At the top step the instance was at about 42% of its two central processing units (CPUs) and the database at about 7%, so neither was the limit.

I read the plateau as a concurrency limit. The instance runs one proxy worker process, and Cloud Run sends it up to 80 requests at once [20]. With CPU idle and requests waiting, the likely limit is how many calls that one process, or the upstream model quota [21], will handle at a time; I have not isolated which. Cloud Run had not yet added a second instance. For scale, the busiest day in the month before the test was 4,627 requests, the busiest minute averaged about 1 request per second, and no more than three instances ran at once in that window.

The test has limits. It used tiny replies, one key and a few minutes. Real agent traffic streams long answers that hold a slot for 12 s or more, and the test said nothing about that. I have not tested more than one instance; any figure for the autoscaling maximum would be multiplication.

Advisories are the other half of running someone else's code. The gateway image is pinned by digest, so a redeploy cannot pull a new LiteLLM by accident. Upgrades go through a runbook: pick an explicit version, read the image's contents without running it, mirror it into our own registry, deploy as a new revision, keep the old one for rollback. A shorter security runbook answers one question, "are we affected?": read the advisory's affected versions and conditions, read the version inside the pinned image, check any other service that bundles LiteLLM, and write down the answer.

That runbook has had real use. On 23 October, after a compromised LiteLLM release (v1.107.0) was reported, the check found the gateway still on the v1.104.0 image it had run since 23 September, no sign that the bad release had been pulled, and no indicator files in the image. An idle test deployment that tracked a floating tag was removed the same day. On 28 October we checked LiteLLM advisory `GHSA-w4c7-34wj-hqm8` the same way; my check record found the open-source build with no enterprise license key unaffected. Both answers were written down the day the check started.

8 Why a Proxy

A self-hosted proxy is one option among several. Commercial and open gateways such as Portkey [22], Kong AI Gateway [23], Cloudflare AI Gateway [24] and Envoy AI Gateway [15] (which NRP itself runs in front of its models) offer similar routing, keys and rate limits, with different trade-offs in hosting, licensing and how budgets are modelled. Hosted routers such as OpenRouter add per-key spend limits but send traffic through a third party, which matters for the data terms researchers accept.

The strongest alternative is to skip the proxy: give each lab its own Google Cloud project and use Vertex AI's native quotas [21] and billing budgets [5]. That gives real dollars from billing instead of a price table, and isolation for free. It also gives billing that lags by hours and budgets that alert rather than stop, per-project setup for every lab, no single key that reaches NRP models, and no per-person view inside a lab. I chose the proxy for per-person keys and caps within a lab and one endpoint across providers, and accepted that its budgets are computed rather than billed. For labs that already have their own cloud project, the native route may be simpler.

9 What Went Wrong

A wildcard that granted more than it said. Until 5 October every key used LiteLLM's `all-proxy-models` setting. When I added the NRP models in an access group meant for testers only, I found that on v1.104 the wildcard also grants every access group. Every key would have reached them. All keys were moved to explicit model lists that day. The cost is real: a new model now reaches nobody until it is added to each key by hand, and lists go stale.

aigw copied the whole catalog into new keys. The same change exposed a bug in aigw: it built each new key's model list from every served model, so it would have handed the tester-only models to anyone. v0.5.0 skipped them. v0.8.0 went further and made every issued key Google-only.

A first live call showed zero models. aigw v0.2.0 read the model catalog through a route that filters by the calling key's models.

aigw's key has none, so it saw none. The fake gateway in the tests did not behave like the real one. The fix in v0.2.1 changed the route and taught the fake to filter the same way.

Prompt caching was off at the start. Before 23 September the gateway logged no cache tokens across 1,256 successful calls to the Claude models, so every agent turn was charged for its full context again. Turning on automatic caching cut the logged cost of a measured 145,000-token turn to roughly a tenth.

Wrong names in the log. From 4 to 17 September, 1,078 requests asked for one model and were served by another because of catalog naming drift. Totals by served model were right; the requested-name view was not. The rows were annotated rather than rewritten, so the audit trail stays intact.

A switch did less than its name. Turning off LiteLLM's built-in admin web console did less than the setting's name suggested, so I learned to check each such switch by probing rather than by reading the configuration.

Open risks. Rate limits and budget counters get fuzzy past a few instances, as described above. The endpoint is public by design, so protection rests on keys [8]. Some keys issued before aigw existed do not yet follow aigw's rules. The configured price table has to track provider price changes by hand. These items are tracked in the gateway's open-items list, and the hardening work has to land before the gateway grows past a pilot.

10 Lessons

(1) **Let the proxy enforce limits and keep the admin layer out of the request path.**
(2) **Treat proxy budgets as soft limits and bound the overshoot with parallel limits.**
(3) **Never let a person, a tool or an inbox hold a plaintext key.**
(4) **Make each undo exact, and know which brakes share a failure.**
(5) **Probe the real system before trusting a setting's name.**
(6) **Pin the image and keep an "are we affected?" runbook.**

11 What's Next

- Requests and approvals in aigw, so a member can ask for a higher cap and the PI approves it in one step.
- A burn-rate forecast for each lab: projected month end and who drives it.
- Rotating the older keys through aigw so every key follows the same rules.
- A test with streaming agent traffic and several instances, a measured budget overshoot, and a decision on a shared cache and database failover before the gateway grows past a pilot.

Sources

The gateway and aigw repositories are private because they name internal services, accounts and people. Records used: the gateway specification (v1.17), change log, README and runbooks; the 23 October load test results and architecture summary; the aigw specification (v2.11), changelog and tags v0.1.0 to v0.8.2; and the 28 October advisory check record.

References

[1] Mario Silic and Andrea Back. 2014. Shadow IT – A View from behind the Curtain. *Computers & Security* 45 (2014), 274–283. doi:10.1016/j.cose.2014.06.007
[2] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 44–60. doi:10.1007/10968987_3
[3] Charles E. Forsyth III. 2026. Governing About 170 Research Cloud Projects. Field Notes. https://charles-forsyth.github.io/papers/2026-10-29-governing-research-cloud-projects.html
[4] FinOps Foundation. 2026. FinOps Framework. https://www.finops.org/framework/ Accessed 28 October 2026.
[5] Google Cloud. 2026. Create, Edit, or Delete Budgets and Budget Alerts. Cloud Billing documentation. https://cloud.google.com/billing/docs/how-to/budgets Accessed 28 October 2026.
[6] Google Cloud. 2026. Disable Billing Usage with Notifications. Cloud Billing documentation. https://cloud.google.com/billing/docs/how-to/disable-billing-with-notifications Accessed 28 October 2026.
[7] J. H. Saltzer and M. D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. doi:10.1109/PROC.1975.9939
[8] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. 2020. *Zero Trust Architecture.* NIST Special Publication 800-207. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-207
[9] University of California Office of the President. 2026. Institutional Information and IT Resource Classification. UC Information Security policy pages. https://security.ucop.edu/policies/institutional-information-and-it-resource-classification.html Accessed 28 October 2026.
[10] Eric Brewer. 2012. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer* 45, 2 (2012), 23–29. doi:10.1109/MC.2012.37
[11] John D. C. Little. 1961. A Proof for the Queuing Formula: $L = \lambda W$. *Operations Research* 9, 3 (1961), 383–387. doi:10.1287/opre.9.3.383
[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. ACM, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165
[13] BerriAI. 2026. LiteLLM: Python SDK and Proxy Server (AI Gateway) to Call 100+ LLM APIs in OpenAI Format. GitHub repository. https://github.com/BerriAI/litellm Accessed 28 October 2026.
[14] BerriAI. 2026. Budgets, Rate Limits. LiteLLM proxy documentation. https://docs.litellm.ai/docs/proxy/users Accessed 28 October 2026.
[15] Envoy Project. 2026. Envoy AI Gateway. https://aigateway.envoyproxy.io/ Accessed 28 October 2026.
[16] Model Context Protocol. 2025. Model Context Protocol Specification, Version 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18 Accessed 28 October 2026.
[17] National Research Platform. 2026. National Research Platform. https://nationalresearchplatform.org/ Accessed 28 October 2026.
[18] Larry Smarr, Camille Crittenden, Thomas DeFanti, John Graham, Dmitry Mishin, Richard Moore, Philip Papadopoulos, and Frank Würthwein. 2018. The Pacific Research Platform. In *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18)*. ACM, New York, NY, USA, 1–8. doi:10.1145/3219104.3219108
[19] Charles E. Forsyth III. 2026. A Road to Nautilus in One Day. Field Notes. https://charles-forsyth.github.io/papers/2026-10-06-a-road-to-nautilus-in-one-day.html
[20] Google Cloud. 2026. Maximum Concurrent Requests per Instance. Cloud Run documentation. https://cloud.google.com/run/docs/about-concurrency Accessed 28 October 2026.
[21] Google Cloud. 2026. Generative AI on Vertex AI Quotas and System Limits. Vertex AI documentation. https://cloud.google.com/vertex-ai/generative-ai/docs/quotas Accessed 28 October 2026.
[22] Portkey. 2026. What is Portkey? Portkey documentation. https://portkey.ai/docs Accessed 28 October 2026.
[23] Kong. 2026. Kong AI Gateway. Kong documentation. https://developer.konghq.com/ai-gateway/ Accessed 28 October 2026.
[24] Cloudflare. 2026. Cloudflare AI Gateway Overview. Cloudflare documentation. https://developers.cloudflare.com/ai-gateway/ Accessed 28 October 2026.