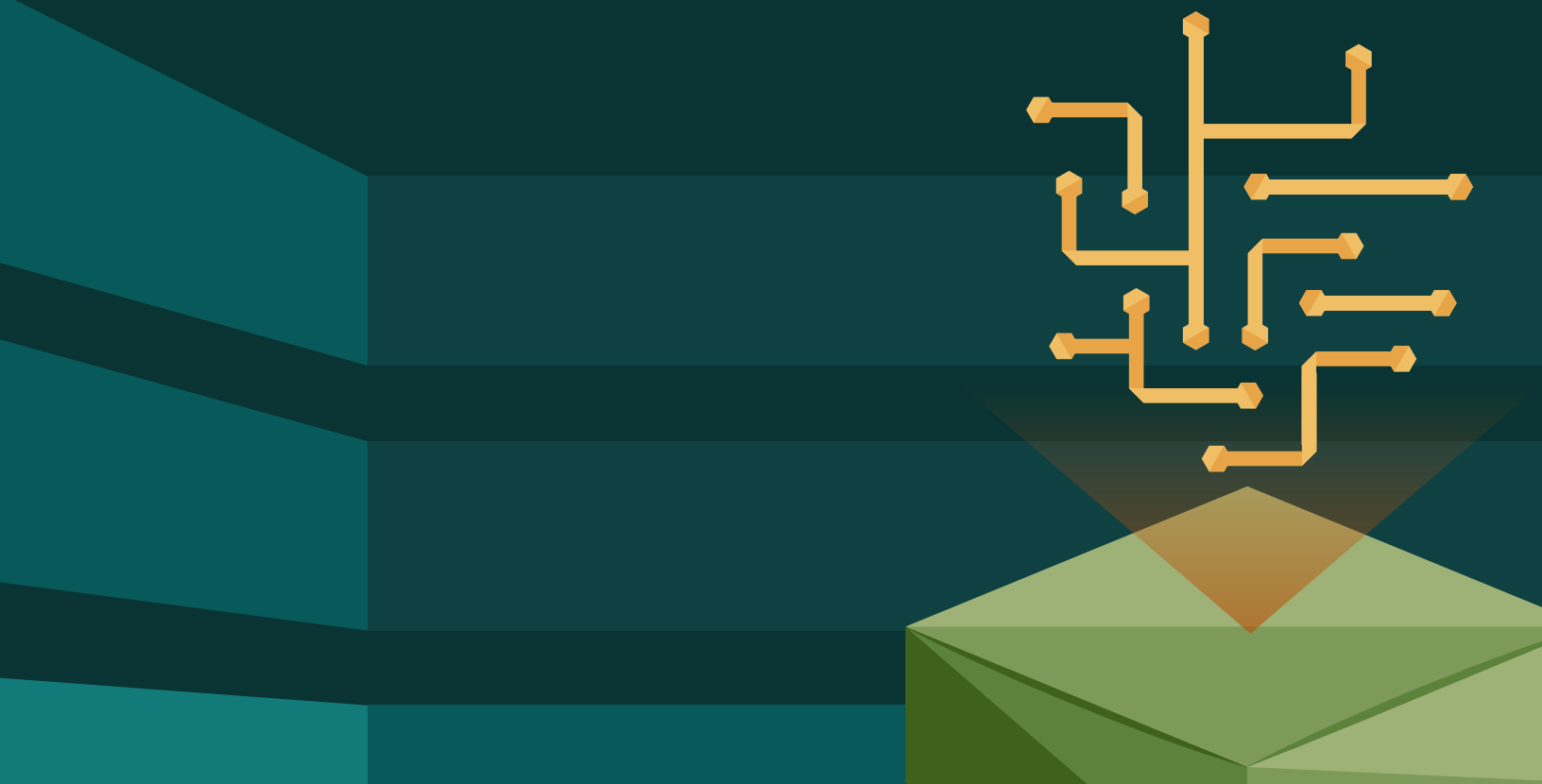
Spectro Cloud

Model routing is the new control plane

How AI clouds can escape the model commodity trap

Smart model routing gets positioned as a way to cut inference cost. Treated properly, it can do much more: it's the layer where an AI cloud enforces policy, survives outages, and stops competing on price. This paper makes the case for routing as a control plane as models commoditize.



Winner-takes-all is giving way to a market in models

For a couple of years the industry behaved as though one model would win and everyone would build on it. That's not how it's playing out. There are now well over fifty models worth taking seriously, large inference gateways already route across dozens, and the gap between the proprietary frontier and the best open weights keeps shrinking, with open releases trailing by six to eighteen months and undercutting closed pricing by a factor of ten to thirty. We've collectively realised that bigger isn't always better.

When this happens, models stop competing on raw capability alone and start competing the way commodities do: on latency, cost, reliability, specialty, jurisdiction, and reasoning quality. A frontier reasoning model is overkill for classifying an expense receipt. A tiny model is useless for a contract review. A multilingual specialist beats a generalist on translation. Different requests want different models, and the same application will want different models for different requests.

A close analogy is the electricity grid. You don't pick one power station and route all demand to it forever; you draw from whichever source is cheapest, greenest, or closest at the moment, and the grid abstracts that away from the consumer.

Intelligence is heading the same way, toward something closer to a spot market than a vendor lock-in. Today the question many companies still ask is "which model should we standardize on?" The more useful question, and the one a routing system answers, is "which model should answer this particular request, right now, under these constraints?"

For an operator, that shift is either an opportunity or a threat depending on which layer you own. If you own the model, commoditization erodes you. If you own the routing layer, commoditization is the thing that makes you valuable.

Cost optimization is great, but it's just the start

The price spread across the model menu is wide and widening: public API rates in 2026 run from roughly $0.10 per million tokens for small, fast models to $60 and beyond for frontier reasoning models. GPT-4–class capability that cost about $30 per million tokens in early 2023 is now under a dollar.

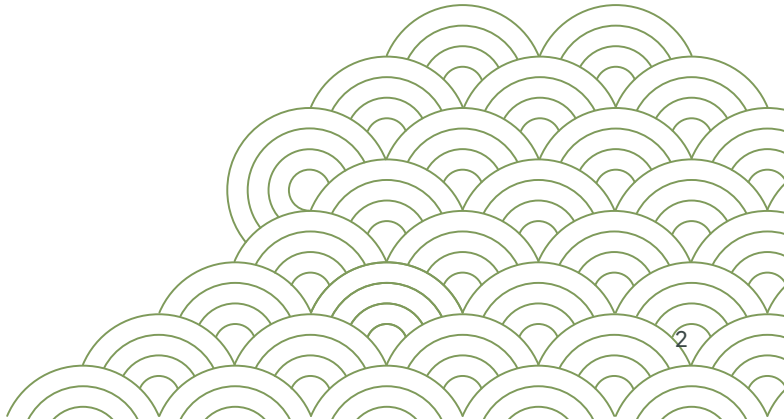
LMSYS's RouteLLM research showed a trained router cutting cost by around 85% on one benchmark while retaining 95% of GPT-4's quality. Send the easy majority of traffic to a small model, reserve the expensive one for the requests that need it, and the savings are substantial and immediate.

But for all the frenzy around token costs and 'tokenomics' — "we spent our year's token budget with Claude by April" — cost is still only one variable among several an enterprise cares about, and rarely the one that keeps a buyer awake.

Ask what an enterprise platform owner loses sleep over and the list is predictable: latency, SLA compliance, uptime, governance, risk, and auditability. Cost matters, but a board report should run on the best reasoning model available regardless of price, while an internal expense-categorization job should run on the smallest model that clears the bar. Business policy drives that decision, not the price sheet.

Cost optimization turns out to be a side effect of routing well, rather than its purpose.

For a neocloud or GPU cloud, the orchestration layer that decides where inference happens, under what policy, at what cost, and with what fallback is a defensible place to stand. Model routing is that layer: a control plane for AI, not a discount mechanism.



The router is the one place that sees every request

The router is the single component that inspects every request before it reaches a model. That makes it the natural place to enforce rules about what is allowed to happen, not only what is cheapest.

Consider the policies an enterprise wants enforced. A request carrying personal or regulated data routes only to a model hosted in an approved jurisdiction. A Platinum-tier customer gets the premium model with no cost ceiling; a free-tier user gets the economy option. A legal workload routes to approved vendors only. A request that exceeds its latency budget switches providers. A tenant that hits its spend cap spills to cheaper capacity instead of failing. When a GPU region goes dark, traffic fails over automatically.

Write those down and the routing layer stops looking like machine-learning infrastructure and starts looking like enterprise middleware, the policy engine that sits in front of a sensitive system and decides what each caller is permitted to do. That's a governance function that happens to produce cost savings.

Reliability is another key outcome. Everyone benchmarks quality; enterprises obsess over uptime. A provider outage should trigger a retry, then a fallback, then a switch to a different provider behind the same API, with the caller none the wiser.

This is where regulation turns a preference into a requirement. The EU AI Act's high-risk obligations apply from 2 August 2026, with penalties reaching €35 million or 7% of global turnover at the top of the scale. The compliance question has moved from "where does the data reside" to "where is this inference processed, and who can see it as it happens," and regulators have begun penalizing missing controls rather than only breaches. That means routing a regulated request to a compliant model is no longer good practice, it's a logged, auditable control you can be fined for lacking. A router that optimizes cost and nothing else cannot satisfy that auditor. One built on tenancy and policy can, which is where sovereign deployments earn their keep.

What the router decides

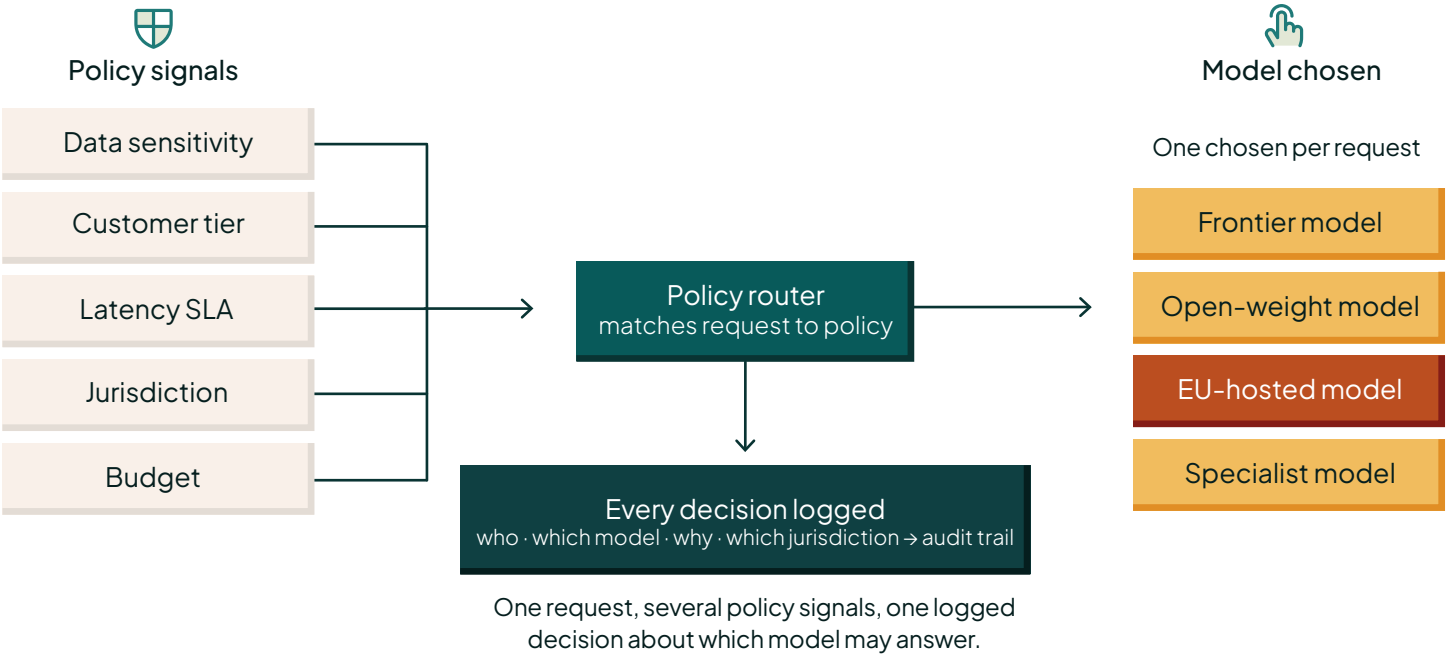


Figure 1: the policy decision a router makes — jurisdiction, tier, SLA, sensitivity, and budget resolving to a model choice, with every decision logged.

Three types of model routing

Model routing gets used for three different jobs, at three layers.

At the bottom, intra-cluster routing decides which GPU serves the request. The KV-cache-aware version scores how much of a request's context already sits in cache across the cluster and places the request to avoid recomputing it. Both chip vendors now ship this: NVIDIA's Dynamo and AMD's ATOMesh, the latter preserving the vLLM and SGLang interfaces teams already run, and both splitting the prefill and decode phases of inference onto separate GPUs. The open, Kubernetes-native version of the same idea is llm-d's endpoint picker, which scores GPU pods by cached-prefix affinity, queue depth, and memory pressure. All of it operates below the application.

In the middle, cross-cluster routing decides which cluster takes the request, by load or budget: distribute on token consumption, and when one cluster hits its cap, overflow to the next without dropping anything.

At the top, policy routing decides which model answers at all, in which jurisdiction, for which tier of customer, against which SLA.

That both chip vendors now ship a router at the bottom layer tells you routing is core plumbing for the AI era. Neither hands you the top layer. You can optimize cache-hit rates to perfection and still send a request to a model your customer's contract forbids, which is solving the wrong problem with great efficiency.

The lower layers make inference fast and cheap; the top layer makes it correct and compliant. An AI cloud needs all three.

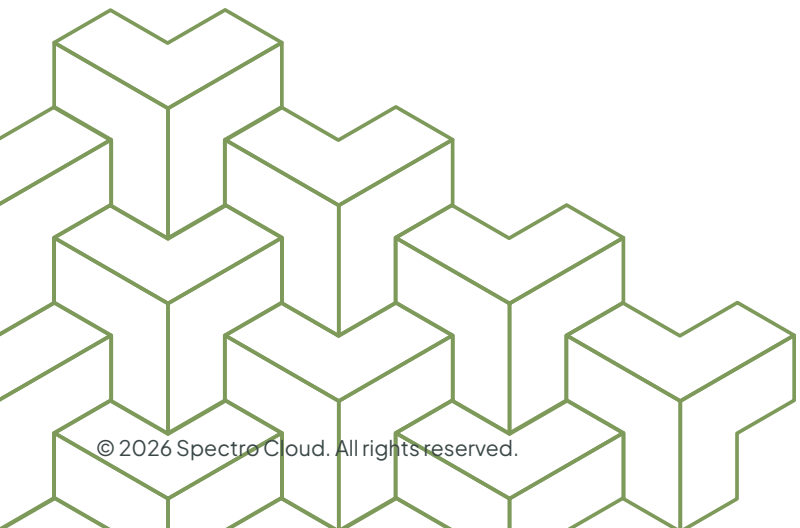
From models to objectives

Today an application (or user) starts by naming a model: call GPT-5, call this open-weight endpoint. The more durable abstraction is that applications specify a reasoning budget and let the routing layer resolve it. "Answer within 200 milliseconds." "Spend no more than $0.02 on this request." "Keep all processing within the UK or EU." "Reach 95% confidence before responding." "Maximize accuracy, cost no object."

The routing layer translates those business objectives into concrete model choices, retries, and escalation paths. Developers program against policies; model names become an implementation detail. An early version of this already runs in Launchpad: an application keeps naming a model in a standard API call, and the routing layer substitutes the right on-prem model behind it, so the name the app sends no longer decides which model answers.

This matters because enterprises rarely want a model. They want an outcome: invoices processed, tickets resolved, contracts reviewed. Nobody ever asked how many SQL queries powered their CRM, and in time nobody will ask which model answered their support ticket, as long as it was answered well, cheaply, and within policy. If the router swaps between five models invisibly to hit the objective, the enterprise is indifferent to which one ran. Value shifts upward, away from the model vendor and toward whoever owns the layer that turns objectives into execution.

The harder technical problem is confidence estimation. A mature router doesn't decide on keywords; it has a small model attempt an answer, estimates how confident that answer is, and escalates to a larger model or a multi-model vote only when confidence is low. That makes routing adaptive rather than rule-bound.



The router sees what no single model provider can

A model provider sees prompts and outputs for its own model. A routing layer sees every model tried, every latency measurement, fallback frequency, cost per successful task, user corrections, and ultimately business outcomes, across every model on the platform.

Over time that becomes a rich, proprietary picture of how different models perform on real workloads, which is information no individual model vendor possesses about its competitors.

Potentially the most valuable asset for you may be the best empirical understanding of how all the models perform across real traffic, used to route better than anyone who only sees their own slice. That's a moat built from telemetry rather than weights, if you can start to gather it.

For neoclouds: operate the fabric or rent the generators

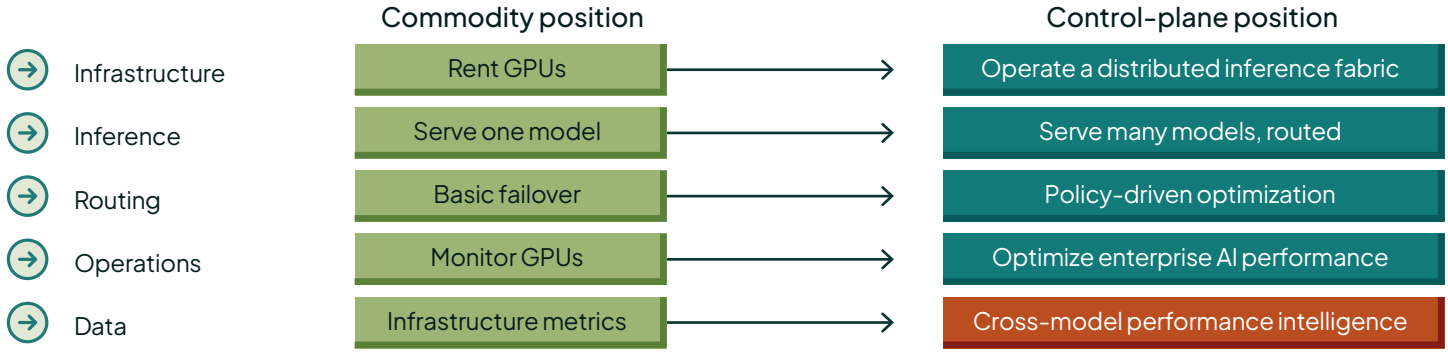
Put the economics together and the strategic choice for a neocloud is stark. Selling GPU capacity is renting the generators. Operating the layer that decides where inference runs, under what policy, with what resilience, is running the grid. The progression looks like this:

- Infrastructure: from renting GPUs to operating a distributed inference fabric across sites and providers.
- Inference: from serving one model to serving many, with the routing to match requests to the right one.
- Routing: from basic failover to intelligent, policy-driven optimization.
- Operations: from monitoring GPUs to optimizing enterprise AI performance end to end.
- Data: from infrastructure metrics to cross-model performance intelligence.

The risk is staying a commodity infrastructure provider while value accrues to the orchestration platforms above you.

The opportunity is to become the control plane enterprises trust for performance, governance, and resilience, the place they route their inference because it's where their policy is enforced and their outages are absorbed. That's a position raw capacity can't be commoditized into.

The neocloud value stack



Renting GPUs is the commodity floor. The value and defensibility live in the layers to the right.

Figure 2: the neocloud value stack, from renting GPUs to operating the inference control plane.

For enterprises: standardize on a routing platform, not a model

Enterprises should spend less energy deciding which model to standardize on and more on which routing platform to standardize on, because the model choice is now a moving target and the platform choice is the durable one.

The platform becomes responsible for policy enforcement, ensuring sensitive workloads only touch approved models and jurisdictions; for cost governance, allocating inference budgets by business process or customer tier; for operational resilience, absorbing outages and degraded performance automatically; for continuous optimization, improving routing from observed outcomes rather than static assumptions; and for avoiding lock-in, by making applications depend on capabilities and policies instead of one vendor's API.

A model you commit to today may be second-best in six months. A routing platform that lets you swap the model underneath without touching the application is how you stop that from being your problem.

Importantly, the model routing gateway can also sit truly internally, managing allocation to local models that have zero token cost, or bursting out to hyperscalers, neoclouds and frontier models as needed, truly breaking down lock-in.

How we're handling routing in PaletteAI

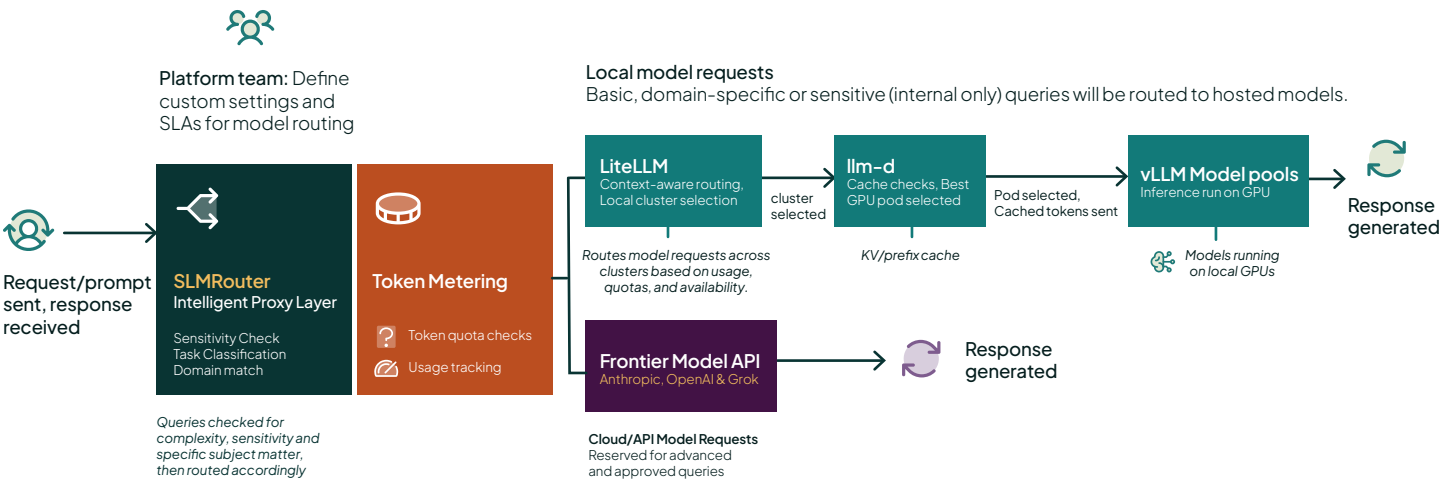
Today, routing in PaletteAI works at more than one layer.

Across clusters, we use LiteLLM to distribute requests by token consumption and overflow one cluster's traffic to the next when it hits its budget, with no client errors and no config change.

Inside a cluster, llm-d's endpoint picker — the Kubernetes-native inference scheduler — sends each request to the GPU that already holds its prompt prefix, scoring pods by cache affinity, queue depth, and memory pressure; in one AI Grid demo we conducted, that combination reached a 94.5% prefix-cache hit rate and cut time-to-first-token by a third.

Above both, our intelligent routing uses a small model to route on task type, content sensitivity, and topic, and lets an application keep making the same OpenAI- or Anthropic-style API calls while the platform maps each one to the right on-prem model, with no code change, and across locations: a local model where one can serve the request, cloud capacity where it can't. It all runs the same way on NVIDIA or AMD silicon.

Automatic model routing



A maturity model for model routing

- **Stage 0: hard-wired.** One model, named in the application. No choice, no failover. An outage anywhere is an outage everywhere.
- **Stage 1: load-balanced.** Multiple deployments behind a logical model, with latency- or cost-based strategies and basic failover. This is where a LiteLLM-based setup gets you, and it's a solid floor.
- **Stage 2: policy-routed.** Routing decisions follow business policy: jurisdiction, tier, SLA, data sensitivity, with every decision logged. This is the rung that survives an EU AI Act audit.
- **Stage 3: objective-routed.** Applications specify budgets and objectives; the router resolves them to models, with confidence-based escalation.
- **Stage 4: self-optimizing.** The router adapts from observed outcomes, and cross-model performance intelligence becomes a standing asset.

We believe most of the market sits at stage 0 or 1. Where do you sit?

Model routing moves AI clouds beyond commodity GPU rental, enforcing policy while cutting inference costs up to 85% without sacrificing quality

(Based on LMSYS's RouteLLM research)

The model-routing maturity ladder

Figure 3: the model-routing maturity ladder, from hard-wired to self-optimizing.

Routing is to models what Kubernetes was to containers

Anyone who runs Kubernetes for a living already knows this pattern. Kubernetes abstracted the machine. You stopped saying "run this process on that server" and started declaring a desired state, leaving the platform to place workloads, balance load, and heal failures. Routing abstracts the model the same way. You stop saying "call this model" and start declaring objectives, leaving the platform to select, place, retry, and fail over across whatever models and providers are available.

For a company whose business is managing Kubernetes fleets across edge, data center, and cloud, this isn't a borrowed metaphor. It's the same control-plane pattern applied one layer up the stack. Models become interchangeable execution engines, managed according to objectives around cost, latency, accuracy, governance, and resilience, in the same way containers became interchangeable units of compute managed by a scheduler.

The bottom line

Routing is not primarily about choosing a model. It's about allocating scarce intelligence under business constraints. Seen that way, it stops being an inference optimization feature and starts being the operating layer for AI, the place an enterprise expresses what it wants and an AI cloud proves it delivered what was asked, under policy and on the record. The operators who win the next few years will be the ones who own that layer. Everyone else will be renting generators and arguing about the price of electricity.

Talk to us

If you're an AI cloud operator or enterprise AI factory owner and you're keen to explore smart model routing, let's have a conversation. Reach out to book a meeting with an AI expert at **spectrocloud.com/get-started.**