The AI-Native Software Aggregator

Architecture, Best Practices, and Model-Arbitrage Strategy for Enterprise Software Delivery

STRATINV — *Institutional Equity Research & Investment*

Author: Gairik Bhattacharya — Managing Director, Stratinv Capital
Date: 27 August 2026
Classification: Public research — free to circulate with attribution
Publisher: Stratinv FZE, 1901 Creative Towers, Hamad Bin Abdullah Street, Fujairah 4422, UAE

---

01 — Executive Summary

A structural opportunity has opened in enterprise software: the intelligence that powers modern applications is no longer built by the application company, it is procured. Frontier large language models (LLMs), open-source models, small language models (SLMs), and the inference engines that serve them have become a competitive, rapidly repricing commodity

market. The company that wins is not the one that owns a model — it is the one that owns the *aggregation layer*: the routing, orchestration, evaluation, and governance fabric that selects, arbitrarily and continuously, the best and cheapest combination of models and inference providers for every task, and packages the result as reliable enterprise software.

This paper sets out a reference architecture for such an aggregator, the model-selection and cost-arbitrage strategies it should run, the design doctrine for its agentic systems, platforms, APIs, external integrations, and billing systems, and a conservative operating framework that protects quality, security, and margin while the underlying model market keeps shifting underneath it. The core claim is simple: **treat intelligence as a supply chain, not a supplier.** Multi-source it, benchmark it continuously, route work to the cheapest node that clears a quality bar, and keep every layer swappable.

02 — The Thesis: Aggregation as Central Intelligence

Three facts define the current market. First, no single model is best at everything: frontier models lead on hard reasoning and code, while open-source and small models now match them on classification, extraction, summarisation, and routine drafting at a fraction of the

price. Second, prices for equivalent capability have fallen by orders of magnitude within a few years and continue to fall with every release cycle — meaning any architecture hard-wired to one vendor is guaranteed to overpay within quarters. Third, inference itself has become heterogeneous: the same open model can be served through hyperscaler APIs, specialist inference clouds, serverless GPU platforms, or self-hosted clusters, each with different latency, throughput, and unit economics.

An aggregated software company exploits all three facts at once. It sits above the model market as a **central intelligence and procurement brain**: it maintains a live scorecard of frontier models, open-source models, and SLMs; it maintains commercial and technical connectivity to multiple inference providers; and it exposes to its own product teams — and to enterprise customers — a single, stable, versioned interface behind which the underlying supplier can change at any time without the application noticing. Margin is created in the spread between the value of the delivered software and the continuously optimised cost of the intelligence consumed. Quality is protected by evaluation, not by brand loyalty.

> **Operating principle:** the application must never know which model answered. Every capability is addressed by task ("classify invoice", "draft contract clause", "plan migration"), and a routing layer resolves the task

> to a model, a provider, and a price — dynamically, and reversibly.

03 — Reference Architecture: The Five-Layer Stack

The aggregator is best designed as five cleanly separated layers, each independently replaceable. Separation is not academic tidiness; it is the mechanism that makes arbitrage possible.

Table 1 — The five-layer aggregator stack

Layer	Contents	Design rule
L1 — Model layer	Frontier LLMs (closed APIs), open-weight LLMs, fine-tuned SLMs, embedding and reranking models, speech and vision models.	Minimum of two qualified suppliers per capability class; every model wrapped behind a common internal schema.
L2 — Inference layer	Vendor APIs; inference clouds; serving engines (vLLM-class, TensorRT-class) on rented or	Same open model qualified on at least two serving paths; unit cost, latency, and

Layer	Contents	Design rule
	owned GPUs; quantised deployments; batch endpoints.	reliability measured per path, per week.
L3 — Orchestration & routing	Model gateway, task router, prompt/version registry, caching, fallbacks, rate and budget governors, evaluation harness.	This layer is the company's proprietary IP. It is built, not bought, and it is where all telemetry concentrates.
L4 — Agentic layer	Planners, tool-using agents, multi-agent workflows, retrieval systems, code-execution sandboxes, human-approval gates.	Agents are stateless workers over durable state; every consequential action passes a policy check and is logged.
L5 — Application & delivery	Enterprise products, platform services, public APIs, third-party integrations, billing and	Applications consume capabilities by task name only; no model identifiers, prompts, or

Layer	Contents	Design rule
	metering, tenant administration.	provider SDKs above this line.

The strategic asset is L3. Models (L1) and inference (L2) are rented commodities; applications (L5) are the revenue surface; agents (L4) are increasingly reproducible patterns. The routing and evaluation fabric — with its accumulated telemetry on which model does which task at what cost and quality — is the compounding, defensible layer, because every unit of production traffic makes its routing decisions better and its cost curve lower.

04 — Model Selection and Arbitrage Strategy

4.1 The task-tiering doctrine

Arbitrage begins with honest task classification. The overwhelming majority of enterprise AI workload — classification, entity extraction, formatting, routing, templated drafting, summarisation of short documents — does not need frontier capability, and paying frontier prices for it is the single largest avoidable cost in the industry. A disciplined aggregator tiers every task and assigns a default model class to each tier, reserving frontier models for the work that genuinely earns them.

Table 2 — Task-to-model routing matrix (illustrative defaults)

Task tier	Examples	Default model class	Escalation trigger
T1 — Deterministic-adjacent	Classification, tagging, field extraction, intent routing, PII detection	Fine-tuned SLM, self-hosted or batch API	Confidence below threshold; novel document class
T2 — Routine generation	Summaries, emails, report sections, FAQ answers, boilerplate code	Mid-tier open-weight model via cheapest qualified provider	Quality evaluator flags output; user rejection
T3 — Complex reasoning	Multi-step analysis, non-trivial code, contract reasoning, planning	Frontier model, standard tier	Long-horizon or high-stakes cases
T4 — Critical / agentic	Autonomous workflows, architecture	Best available frontier	Never auto-downgraded; cost is

Task tier	Examples	Default model class	Escalation trigger
	design, financial or legal drafting for review	model with extended reasoning; human gate on output	secondary to correctness

Target steady state: 70–85% of token volume on T1–T2 economics; frontier spend concentrated where it changes outcomes.

4.2 Cascade and fallback routing

The highest-leverage routing pattern is the **cascade**: attempt the task on the cheapest qualified model, score the output with a lightweight evaluator (a rubric-following small model, schema validation, or business-rule checks), and escalate to a stronger model only on failure. Cascades routinely cut blended cost by 50–90% on mixed workloads while holding end-quality, because the expensive model is consulted only for the minority of cases that need it. The inverse pattern — **fallback** — protects availability: if a provider degrades or rate-limits, the gateway reroutes to the next qualified provider automatically, which is why every capability must have at least two qualified suppliers at all times.

4.3 Continuous benchmarking and repricing

Model choice is never a one-time decision. The aggregator maintains a private evaluation suite built from its own anonymised production tasks — not public benchmarks, which are saturated and gamed — and re-runs it on every notable model release and every provider price change. A model earns its way into a routing tier by clearing the quality bar on the private suite; among models that clear the bar, the router prefers the cheapest cost per successful task (not per token — a cheap model that fails 20% of the time and triggers retries can be dearer than a mid-priced one that succeeds). Switching decisions are made monthly at minimum, and immediately upon major releases or reprices.

4.4 SLMs and fine-tuning as a cost weapon

Wherever a task is high-volume, narrow, and stable — the profile of most enterprise back-office work — the endgame is a small model fine-tuned on the aggregator's own traffic, distilled from the frontier model that originally served the task. The frontier model becomes the teacher and the exception handler; the SLM becomes the workhorse at one-tenth to one-hundredth the unit cost, often with lower latency and the option of fully private deployment inside a customer's perimeter — itself a saleable enterprise feature.

05 — Inference-Layer Economics

Even with the model fixed, the serving path is an independent arbitrage dimension. The same open-weight model can differ several-fold in cost across providers, and further still against self-hosting at high, steady utilisation. The conservative doctrine is to **rent first, own late**: use per-token APIs while volume is uncertain, move steady predictable workloads to committed capacity or self-hosted clusters only when utilisation forecasts justify it, and keep burst traffic on APIs permanently.

Table 3 — Principal inference cost levers

Lever	Mechanism	Typical impact
Prompt caching	Reuse of long, repeated context (system prompts, schemas, documents) priced at a deep discount by most providers	Large savings on context-heavy workloads
Batch processing	Non-urgent jobs (overnight enrichment, reporting, embeddings) sent to discounted	Commonly ~50% off interactive pricing

Lever	Mechanism	Typical impact
	asynchronous endpoints	
Semantic response caching	Serving repeated or near-duplicate queries from cache instead of the model	Eliminates a meaningful share of calls outright
Context discipline	Retrieval instead of stuffing; summarised memories; trimmed histories; structured outputs to cap generation length	Directly proportional token savings
Quantisation & serving engines	Serving open models quantised on optimised engines with continuous batching	Multiplies throughput per GPU on owned capacity
Committed-use pricing	Provider commitments negotiated against forecast volume, with multi-vendor leverage	Double-digit percentage discounts

The finance discipline matters as much as the engineering. Every request carries metadata — tenant, product, feature, task tier, model, provider — so that cost is attributable to the unit of business value it served. The company should know, weekly, its cost per document processed, per ticket resolved, per report generated, by customer; without that, arbitrage is blind and pricing is guesswork.

06 — Designing the Agentic Layer

Agentic systems — models that plan, call tools, and act across multiple steps — are where enterprise value concentrates, and also where cost and risk compound fastest, because errors and tokens both multiply across steps. The design doctrine is deliberately conservative.

Workflows before autonomy. Most "agent" use cases in the enterprise are better built as explicit, orchestrated workflows — fixed sequences of model calls, tool calls, and checks — than as open-ended autonomous loops. Workflows are testable, debuggable, and predictable in cost. Full autonomy is reserved for tasks where the path genuinely cannot be predetermined, and even then it runs under a step budget, a token budget, and a wall-clock budget.

Tools as contracts. Every tool an agent can call is a typed, versioned, least-privilege API with validated inputs and outputs. Agents receive the minimum toolset for the

task; destructive or financially consequential actions (payments, deletions, external communications, production changes) sit behind human-approval gates or hard policy engines that the model cannot talk its way past.

Durable state, stateless workers. Agent progress lives in a database, not in a context window. Any step can be retried, resumed, or handed to a different model mid-workflow — which also means the routing layer can arbitrage *within* an agentic run, using a frontier model for the planning step and cheap models for execution steps.

Full observability. Every step, prompt, tool call, and output is traced and replayable. This is simultaneously the debugging system, the audit trail enterprises demand, and the training data for the distillation programme described in §4.4.

07 — Enterprise Delivery: Platform, APIs, Integrations, Billing

7.1 Platform and product design

Enterprise buyers do not purchase models; they purchase outcomes with guarantees. The platform therefore leads with reliability engineering: multi-tenant isolation, per-tenant encryption and data residency options, role-based access control, SSO, audit logs, and contractual clarity that customer data is not used to train

third-party models. AI capabilities are packaged as named product features with service levels — not as a raw "chat with AI" surface — and every AI feature ships with a defined failure mode: what the user sees when the model is wrong, slow, or unavailable.

7.2 API design

The company's public API should mirror its internal philosophy: task-oriented, model-agnostic endpoints (`/v1/extract`, `/v1/draft`, `/v1/review`) with strict JSON schemas, idempotency keys, webhooks for long-running jobs, and explicit versioning with long deprecation windows. Structured outputs are the default; free-text is the exception. Confidence scores and citation/grounding metadata are first-class response fields, because enterprises build their own review workflows on top of them. Internally, the same gateway pattern applies to consumed third-party APIs: every external dependency — model providers, data vendors, SaaS integrations — is wrapped behind an internal adapter with circuit breakers, retries with backoff, and a qualified substitute where one exists.

7.3 Third-party and external integration systems

Integration breadth is a moat. The aggregator maintains a catalogue of connectors into the systems where enterprise work actually lives — ERP, CRM, ITSM, HRIS, document stores, data warehouses — built on standard protocols and event-driven synchronisation rather than

brittle point scripts. Emerging open protocols for tool and context interchange (of which the Model Context Protocol is the current standard-bearer) should be adopted early: they convert integration work from bespoke engineering into configuration, and they keep the agentic layer portable across model vendors.

7.4 Billing and metering systems

Billing is a strategic system, not back-office plumbing, because AI costs are variable while enterprise buyers demand predictability. The recommended design is a real-time metering pipeline (every request emits a usage event with full attribution) feeding a rating engine that supports several commercial models simultaneously: seat-based subscriptions for predictability, usage-based tiers with included allowances and soft caps, outcome-based pricing for well-measured tasks, and enterprise commitments with true-up. Two governors are non-negotiable: per-tenant budget caps with alerting, so a runaway workload cannot surprise either party; and an internal margin monitor that compares the metered revenue of every feature against its attributed model cost, weekly, so that repricing by upstream providers is passed through deliberately rather than absorbed unknowingly.

08 — Best Practices: Quality, Safety, and Governance

Table 4 — Operating best practices for the aggregator

Domain	Practice
Evaluation	A private, production-derived evaluation suite is the company's quality constitution. No model, prompt, or routing change ships without passing it; regressions block release exactly as failing tests do in software.
Guardrails	Input screening (prompt-injection and data-exfiltration defence), output validation (schema, policy, PII), and grounding checks on retrieval answers run as an independent layer that no single model controls.
Security	Least-privilege credentials per agent and per tool; sandboxed code execution; secrets never enter prompts; third-party model traffic minimised to the data strictly necessary for the task.
Data governance	Classification of what may leave the perimeter to which provider under which contract; regional routing for residency; retention schedules on traces; customer opt-outs honoured mechanically, not procedurally.
Compliance	Map deployments against emerging AI regulation (EU AI Act risk classes and

Domain	Practice
	equivalents); maintain model cards, decision logs, and human-oversight documentation as standing artefacts, not audit-time scrambles.
Reliability	Multi-provider fallbacks, graceful degradation paths, load-shedding by task tier (T1 before T4), and chaos-testing of provider outages as a scheduled exercise.
Organisation	A small central platform team owns L2–L3; product teams own L4–L5 against published internal SLOs; a weekly model-market review ratifies routing and pricing changes with engineering and finance in the same room.

09 — The Conservative Operating Doctrine

The strategy described here is aggressive in procurement and conservative in everything else. Five rules keep it safe.

1. No single point of intelligence. Every capability has at least two qualified models and two serving paths.

Vendor concentration above roughly 60% of spend is treated as a risk finding, not an efficiency.

2. Quality bars are fixed; costs float. Arbitrage only ever operates among options that clear the private evaluation bar for the task tier. The router minimises cost *subject to* quality — never the reverse — and T4 critical work is exempt from downgrading entirely.

3. Rent before owning; commit only against measured demand. Capital commitments to GPUs or long provider contracts follow at least two quarters of observed, attributable utilisation, and burst capacity always remains rented.

4. Humans gate consequences. Autonomous systems propose; authorised humans (or hard policy engines) dispose, wherever an action is irreversible, financial, legal, or customer-facing. This is both risk management and, presently, what enterprise buyers require to sign.

5. Everything is replaceable, therefore nothing is a crisis. Model deprecations, provider reprices, and capability leapfrogs are routine supply-chain events handled by the routing layer — because the architecture assumed from day one that they would happen.

> **Investment framing:** in this structure, gross margin is an engineered quantity. It expands mechanically as upstream model prices fall, as cascades push volume down-tier, and as distilled SLMs replace rented frontier capacity — while the evaluation and telemetry

> moat deepens with every request served. The aggregator captures the deflation of intelligence instead of being disrupted by it.

10 — Conclusion

The decisive competitive position in enterprise AI software is not model ownership but **orchestrated neutrality**: a disciplined aggregation layer that treats frontier models, open-source models, SLMs, and inference providers as an interchangeable, continuously benchmarked supply base, and converts that flexibility into simultaneously higher quality (best tool per task), lower cost (cheapest qualified path per task), and lower risk (no dependency, full auditability). Companies that hard-wire themselves to a single model vendor inherit that vendor's prices, outages, and roadmap. Companies that build the five-layer aggregator inherit the entire market's progress — and sell it, packaged, governed, and guaranteed, to the enterprise.

---

illustrative of market structure and subject to rapid change. Readers should perform their own diligence before making commercial or investment decisions.