FROM EXPERIMENTATION TO ENTERPRISE LINE ITEM

AI Cost Optimization: Managing LLM Spend at Scale

Executive briefing — CFO, CTO & FinOps Leadership | ~20 min

Why LLM Spend Catches Finance Off Guard

- Traditional software is licensed per seat or per year — predictable and budgetable
- LLM spend is usage-based: every prompt, token, and API call has a marginal cost
- Cost scales with adoption success — the more employees use the tool, the higher the bill
- Usage often grows faster than procurement cycles can forecast or approve
- Finance teams accustomed to fixed SaaS costs are frequently surprised by month-over-month variance

The Core Cost Levers

- Model choice: frontier models cost meaningfully more per token than smaller or mid-tier models
- Prompt and context length: every token sent to the model is billed, including system prompts and history
- Caching: reusing previously computed context can reduce repeated-cost workloads
- Batching: grouping non-urgent requests can lower effective per-request cost
- Output token limits: uncapped generation length is an easy, invisible source of overspend

Model Routing: Match the Model to the Task

- Not every request needs the most capable (and most expensive) model
- Route simple, high-volume tasks (classification, extraction, short replies) to smaller models
- Reserve frontier models for complex reasoning, high-stakes, or low-volume tasks
- A routing layer (rules-based or model-based) can direct traffic automatically
- Industry-reported range: well-designed routing can meaningfully reduce blended per-request cost, though savings vary widely by workload

Build vs. API: When Self-Hosting Actually Pays Off

- API access has low fixed cost but variable per-token pricing that scales with usage
- Self-hosting requires upfront infrastructure, GPU capacity, and ongoing MLOps investment
- Self-hosting tends to make sense only at sustained, high, predictable volume
- At low or spiky volume, API pricing is usually cheaper than idle infrastructure
- The decision should be revisited periodically as usage patterns and vendor pricing evolve

Illustrative Scenario: A Cost-Optimization Program

- Illustrative scenario, not a verified case study — for discussion purposes only
- A mid-size enterprise rolls out model routing, caching, and output limits over one quarter
- Simple support and lookup tasks are shifted to a smaller model tier
- Frequently repeated context (policy documents, system instructions) is cached
- Illustrative outcome: a noticeable reduction in blended cost per request without a measurable drop in user satisfaction — actual results depend on workload mix

Monitoring and Attribution: Know Where Spend Comes From

- Aggregate spend dashboards are not enough — cost must be traceable to team, product, or use case
- Chargeback or showback models make individual teams accountable for their own usage
- Per-request tagging (team, feature, environment) enables granular cost analysis
- Attribution surfaces which use cases are cost-effective and which are not
- Visibility is a prerequisite for any optimization effort — you cannot manage what you cannot see

Caching and Prompt Engineering: Cutting Cost Without Cutting Quality

- Prompt caching avoids re-processing static context (instructions, reference documents) on every call
- Shorter, more precise prompts reduce token count without reducing task performance
- Structured output formats reduce wasted generation and re-prompting
- Removing redundant context or conversation history that no longer adds value lowers cost per call
- These techniques are typically the fastest wins because they do not change the underlying model

Negotiating Enterprise Pricing and Commitments

- Vendors often offer tiered or volume-based pricing above certain usage thresholds
- Committed-use or reserved-capacity agreements can lower per-token rates in exchange for spend commitments
- Multi-model or multi-vendor strategies can create negotiating leverage
- Contract terms should include rate protection, transparent usage reporting, and exit flexibility
- Procurement and FinOps should be involved before, not after, usage scales significantly

The Risk of Over-Optimizing: Quality Erosion

- Aggressive cost-cutting (smaller models, truncated context, shorter outputs) can degrade output quality
- Degraded quality often shows up as more user retries, follow-up prompts, or workarounds — offsetting savings
- Poor output quality erodes user trust and adoption, undermining the business case for the tool
- Cost and quality should be optimized jointly, not cost in isolation
- Establish quality guardrails and monitor them alongside cost metrics

Building a FinOps-for-AI Practice

- Apply the same discipline used in cloud FinOps — visibility, allocation, optimization, governance — to token spend
- Establish a cross-functional owner (Platform, FinOps, or a dedicated AI cost function)
- Define budgets and alerts at the team or product level, not just organization-wide
- Review usage and pricing options on a recurring cadence, not as a one-time exercise
- Treat AI spend as an operating expense requiring ongoing management, not a fixed line item

Common Pitfalls to Avoid

- Treating LLM cost as a one-time budgeting exercise rather than an ongoing discipline
- Optimizing for the cheapest model without measuring downstream quality impact
- Lack of attribution, making it impossible to identify which use cases drive cost
- No output or context limits, allowing runaway token consumption on edge cases
- Delaying vendor negotiation until spend has already scaled past leverage points

Roles and Responsibilities

- CTO/Platform: owns architecture decisions — model selection, routing, caching infrastructure
- FinOps: owns visibility, attribution, budgeting, and vendor commercial terms
- CFO: owns budget guardrails and ROI framing tied to business outcomes, not just spend totals
- Product owners: accountable for usage patterns and quality tradeoffs within their domain
- Shared governance forum ensures cost decisions do not happen in isolation from quality and product goals

What Good Looks Like: A Maturity View

- Stage 1: Spend is visible only in aggregate, no attribution, reactive to bill shock
- Stage 2: Per-team or per-product attribution in place, basic budgets and alerts
- Stage 3: Model routing and caching implemented, quality metrics tracked alongside cost
- Stage 4: Ongoing FinOps-for-AI practice with vendor negotiation and periodic architecture review
- Most organizations today sit between Stage 1 and Stage 2 — the opportunity is significant

Next Steps and the Ask

- Stand up baseline attribution: tag current LLM usage by team and product within the next quarter
- Pilot model routing and caching on one high-volume, low-complexity workload as a proof point
- Establish a joint FinOps/Platform review cadence (monthly) to track cost and quality together
- Engage procurement on enterprise pricing terms once usage baselines are established
- The ask: sponsorship to fund a small cross-functional working group to own this practice going forward