THE AI COST OPTIMIZATION CHEAT SHEET

Cut Your LLM Bill 50-90% Without Losing Quality

7-LAYER PLAYBOOK | REAL 2026 NUMBERS | TOKEN FINOPS | PRODUCTION-TESTED

Tokens got ~280x cheaper in two years, yet enterprise AI bills rose 320%. The problem isn't pricing - it's usage. This one-page field guide gives AI engineers the exact levers to diagnose and slash LLM spend at scale, with real 2026 benchmark numbers. Print it, pin it, ship profitable AI.

A free resource from **Tech With Koushik** • **www.koushikmaji.com**

1 THE PARADOX & TOKEN ECONOMICS

Why cheaper tokens = bigger bills

Per-token prices fell ~280x while total spend rose 320% and the median bill grew 7.2x YoY. Cheaper tokens didn't cut costs - they made token-hungry patterns (agents, long context, reasoning modes) economical, so consumption exploded. **You don't have a pricing problem; you have a usage problem.**

Cost driver	Why it burns tokens
Output token premium	Output costs 3-10x more than input (generated one token at a time).
Invisible reasoning tax	Reasoning tokens billed as output but never shown to you.
Agent multiplier	1 chatbot call = 10-20 agent calls; inference = 85%+ of AI budgets.
Quadratic context	128K context can cost ~64x an 8K prompt (attention scaling).
Re-send tax	Full history re-sent every turn; 10 steps = history billed 9x.
System-prompt repetition	2K-8K token prompt re-billed on every single call.

Real horror story: two agents looped for 11 days and ran up a **$47,000** bill. Unbounded agents are a financial risk - cap them (Layer 7).

The cost anatomy of a typical bill

Bucket	Share	Fix it with
Main task work	40-60%	Routing, output control, compression
Retries & tool loops	10-20%	Guardrails, budgets, better prompts
System-prompt overhead	10-15%	Prompt caching (Layer 1)
Pathological inputs (1%)	10-15%	Token budgets + input limits
Background tasks	5-10%	Batching + small/OSS models

2 THE 7-LAYER SAVINGS PLAYBOOK

#	Layer	What to do	Typical savings
1	Prompt caching	Cache the static prefix; put static first, dynamic last.	Up to 90% off repeated context
2	Semantic caching	Reuse answers for semantically similar queries (cos > 0.95).	15-40% hit rate on Q&A;
3	Model routing	Cheap classifier routes each task to the smallest capable model.	50-80% on mixed traffic
4	Batching	Async Batch API for non-urgent jobs; vLLM for self-host.	50% off; up to 23x throughput
5	Output control	'Be concise' + structured output; cap max tokens.	40-60% fewer output tokens
6	Context compression	Summarize history; prefer JSON over verbose free text.	Cuts input + output bloat
7	Token budgets	Hard caps, step limits, loop breakers per task.	Prevents runaway $-loops

Sequencing: do caching FIRST (near-free, fastest ROI, zero accuracy risk), then routing, then batching. Save quantization / custom infra for LAST. Stacked, these layers compound to **60-90%** total savings.

Prompt-caching structure (concept)

Order your prompt so the cacheable part comes first:

```
[ STATIC: system rules + few-shot examples ] <- cached (cheap)
[ DYNAMIC: user query + fresh tool results ] <- not cached
```

Anthropic cache read ~$0.30/M vs $3.00/M uncached = ~90% off. Break-even < 3 reuses/hour.

Bonus lever: open-source & small models

For narrow tasks (classification, extraction, summarization) use a small/OSS model - now ~38% of enterprise token volume. A task on a frontier reasoning model can cost up to **190x** the same task on a fast small model. Reserve frontier models for genuine reasoning.

3 THE MINDSET SHIFT: AI FINOPS

Principle	What it means
Cost-per-successful-output	Stop optimizing cost-per-token. A cheap wrong answer you retry isn't cheap.
TokenOps / tag everything	Tag every call (team, feature, env) - turn a black box into unit economics.
The FinOps flywheel	Visibility -> Allocation -> Optimization -> Governance -> repeat.
Diagnose before optimize	Find your biggest line item first; apply the right lever to the real leak.
Blended cost per token	Track it alongside total volume for accurate forecasting.

4 COST-CUTTING CHECKLIST + QUICK REFERENCE

- ✓ Cache repeated prompt prefixes
- ✓ Structure prompts: static first, dynamic last
- ✓ Add semantic caching for Q&A; / support
- ✓ Route tasks to the cheapest capable model
- ✓ Batch all non-urgent workloads
- ✓ Cap output length + use structured JSON
- ✓ Summarize context; don't resend full history
- ✓ Set hard token budgets + loop breakers
- ✓ Use OSS/small models for narrow tasks
- ✓ Tag every call; track cost-per-success

SAVINGS AT A GLANCE
Prompt caching: up to 90% • Routing: 50-80% • Batching: 50% • Output control: 40-60

RED FLAGS IN YOUR STACK
One model for everything • No caching layer • Unbounded agent loops • Full transcripts between ag

ONE-LINE RULES
• Don't send trivial work to your priciest model. • Cache what repeats. • Batch what can wait. • Cap what can run away. • Measure success, not tokens.

Cheaper tokens won't save you. Better engineering will.

More deep-dives & free resources at **www.koushikmaji.com** • Subscribe to **Tech With Koushik**