TOTEM LTD. · FIELD GUIDE AND WORKSHEET

What AI actually costs, and what to do about it

August 2026. Complete as published; nothing withheld. Public and forwardable.

Per-token prices fell sharply through 2025 and 2026. Enterprise AI bills went up anyway. This guide explains why, sets out what to do in the order worth doing it, and is honest about which widely cited savings do not survive contact with a real workload. It ends with a worksheet you can fill in for your own organization.

Everything here is public knowledge, assembled. If you work through it and find nothing new, that is the correct outcome and you are further along than most.

Why the bill rises while the price falls

A chatbot query triggers one model call. An agentic workflow plans, calls tools, reads results, checks its own work, and corrects, which can mean ten to twenty calls for a single request a person made once. Gartner's March 2026 analysis put agentic workloads at five to thirty times the tokens of a standard chatbot query. EY modelled a customer service interaction at roughly four cents in 2023 against about a dollar twenty in 2026 for an orchestrated equivalent, around thirty times higher.

Most organizations met that multiplier when the invoice arrived rather than when the system was designed. The FinOps Foundation's 2026 findings, as analyzed by Optimum Partners, put the share of enterprises whose AI costs exceeded projections at seventy-three percent. Over the same period the share of FinOps practitioners responsible for AI spend went from about a third to nearly all of them.

The lesson from the most public case is not that AI is expensive. Uber rolled coding agents out fast, exhausted its annual AI budget in roughly four months, and its own leadership said publicly that higher token consumption had not visibly produced higher output. The organization was rewarding volume while nobody could draw a line from the spend to finished work.

The unit that matters

Cost per token is not a business metric. Cost per completed task is, where a completed task means a discrete request that produced an output a named person accepted as usable.

This distinction does real work. A workflow whose token bill doubled while handling triple the volume got cheaper, and raw spend without a denominator produces panicked and wrong decisions. It also speaks the language the person approving the budget already uses, because no chief financial officer thinks in tokens.

If your AI reporting counts prompts, agents, or tokens, you are measuring effort. Measure finished work.

What to do first, in order

Turn on provider-side prompt caching before anything else. Stable content, system prompts, policy text, schemas, and long instructions are re-sent on every call and are exactly what caching is for. It is the highest return for the least effort in the entire list.

Then control output length. Across leading 2026 models, output tokens price at roughly three to eight times input, with a median near four to one. Applications that generate long responses pay far more per generated character than per character of context they send. Asking for a shorter answer is often a larger saving than trimming the prompt.

Then right-size the model per task. The price differential across capability tiers runs twenty to fifty times. The common expensive mistake is not a technical error; it is defaulting everything to the strongest available model.

Then put retrieval on a fixed token budget. A retriever that returns whatever it found forces the model to pay for material it did not need. A budget forces relevance over volume.

Then cache exact matches before attempting anything cleverer. Exact caching is a hash lookup with zero false positives and no correctness risk.

Then instrument per workflow, per team, and per model, with anomaly alerts. A cost regression should be caught like a performance regression, in days, not at the monthly invoice.

The agentic layer, where the money actually goes

If you run agents, the biggest lever is what enters the context window rather than which model reads it.

Four patterns cover most of it. Write, meaning keep durable state outside the context in notes or checkpoints rather than in the transcript. Select, meaning load tools and documents when needed instead of declaring everything up front. Compress, meaning summarize resolved turns and trim raw tool output. Isolate, meaning delegate to sub-agents with their own context windows so one task's noise never contaminates another's.

Two specific techniques are worth knowing by name. Tool-call collapsing, sometimes called code mode, lets an agent write code that fetches and filters data before any of it enters context; Cloudflare's February 2026 implementation collapsed over 2,500 endpoints into two tools using roughly a thousand tokens, against 1.17 million for a conventional server definition. And sub-agent isolation has been measured at roughly 9,000 tokens for a multi-domain query against 15,000 for an accumulating pattern, with the honest tradeoff that isolated sub-agents reset each time and so give no benefit on repeated similar queries.

Where the published numbers mislead

This section is the reason the guide exists. Three widely repeated savings behave differently in production than in the papers.

Routing and cascades. The famous figures are real and they are benchmark figures: FrugalGPT reported up to ninety-eight percent cost reduction, and RouteLLM over eighty-five percent on one benchmark while retaining ninety-five percent of the strong model's performance. A June 2026 evaluation found cascade approaches landing below the accuracy of simply always using the strong model once rejected escalation calls were amortized into the cost. Router benchmarking has separately found that aggregate metrics conceal domain and difficulty differences. Routing works when your query distribution is genuinely skewed toward easy questions. Whether yours is, is a measurement, not a purchase.

Semantic caching. Hit rates of thirty to seventy percent are commonly reported, and the failure mode is silent. One financial-services team found their cache returning matches at similarity scores high enough to look safe on questions that differed in ways that mattered, serving an answer generated without a stress qualifier to a question that included one. The research position is blunt: a cache returning false hits is worse than no cache, because users receive confidently presented wrong answers. Prefer misses. Run exact matching first and semantic only underneath it, tune the threshold on real traffic in shadow mode before serving anything, and if the semantic layer adds only a few points over exact matching, the risk is not worth the complexity.

Context compression. Trimming helps until it does not. One study observed a context of roughly eighteen thousand tokens collapsing to a hundred and twenty under iterative rewriting, with accuracy falling below doing nothing at all. Compression needs a floor and a test.

The pattern across all three: every one of these techniques fails quietly. Nothing throws an error. You find out from a user, or you never find out.

Measurement that survives a finance conversation

Baseline before you change anything, because a baseline reconstructed afterward will not be believed and should not be.

Track cost per completed task, first-pass acceptance, cycle time from request to accepted output, and adoption measured as distinct people initiating work. Keep human review time and setup effort out of the headline cost number and report them separately, so the payback period stays visible rather than buried.

Write down in advance what result would mean the effort failed. A pilot that cannot fail is a demonstration.

The worksheet

Fill this in for one workflow, not for your whole organization. If you cannot answer a row, that gap is your next task.

Question	Your answer
What is one completed task in this workflow, in one sentence?	
Who accepts it as usable, by name?	
What does one completed task cost today?	
How many model calls does one task trigger?	
What fraction of returns are accepted on first pass?	
How long from request to accepted output?	
Is provider-side prompt caching on?	
What is the typical output length, and is it capped?	
Which model tier does each step use, and why that one?	
Is retrieval on a fixed token budget?	
If you route, has routing been tested on your own query distribution?	
If you cache semantically, when was the threshold last validated on real traffic?	
What does the governance or instruction layer cost in tokens, separately?	
What result would tell you this is not working?	

The gap this guide does not close

Everything above is about the mechanics of spend. The larger question underneath it is whether the instructions you give a system, the standards, constraints, and boundaries an organization has usually

already written down, change how often its output is accepted the first time.

The governance industry asserts that link constantly. As of this writing we could not find it measured and published anywhere with a stated effect size. TOTEM is running that measurement and will publish the result, including if the result is that there is no effect.

TOTEM Ltd. operates as an industry infrastructure execution partner. This guide is offered without registration, gate, or follow-up obligation.

totemltd.com · hello@totemltd.com