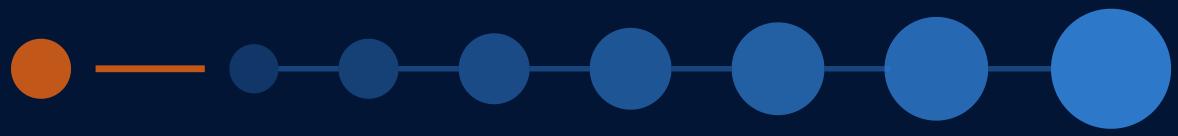
WHITEPAPER · AGENTIC AI FOR PRODUCT LEADERS

# What an Agent Actually Costs

Token economics and the build decision, for product managers



How to compute a defensible per-task cost for an agentic product, what the architecture does to that number, which levers move it and who pulls them, and how to compare the result against the alternative you are actually replacing.

**Yoram Friedman, MD**
Physician and enterprise product leader · SAP, Walmart, Microsoft, regulated health technology
Drawn from the Agentic AI for Product Leaders series · agenticaiproductmanagement.com

# The number that decides your product is one nobody will quote you.

Most teams cost an agent by looking up a price per million tokens, multiplying by volume, and comparing the result to an hourly rate. Every step of that is wrong, and the errors do not cancel.

The number that decides your product is not the model's price. It is the **architecture multiplier**. An agent does not call the model once. It re-enters the model repeatedly within a single task: to classify, to read what it retrieved, to weigh the result, to check policy, to draft, to retry the step that failed. For an ordinary enterprise agent that is five to fifty model calls per task. For an agent that writes code it is one hundred to five hundred. The same task on the same model can cost a cent or a dollar depending entirely on how the loop is built.

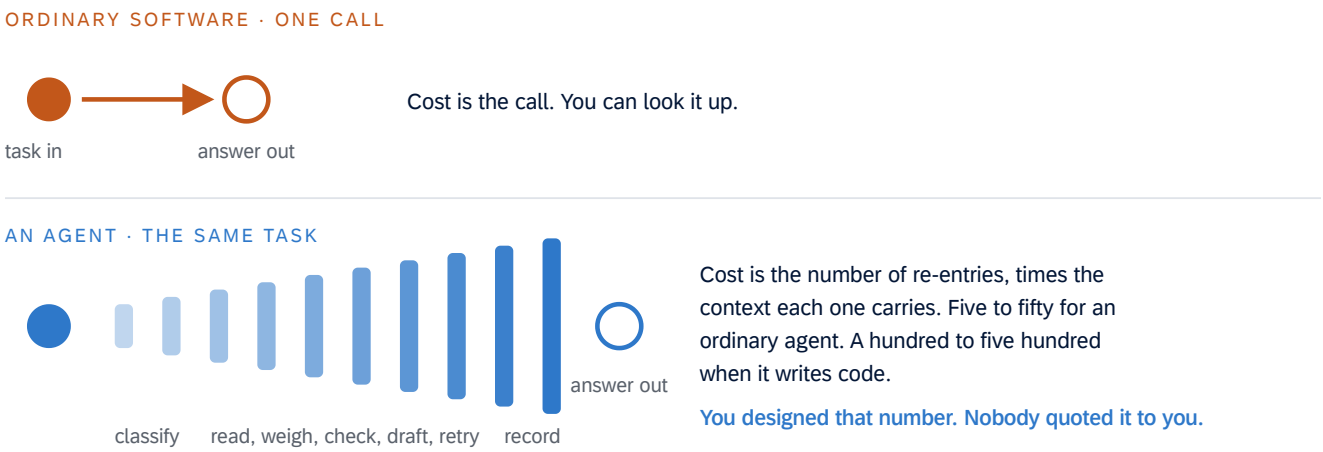



Figure 1. Where the cost actually comes from. In ordinary software the cost is the call, and you can look it up. In an agent the cost is the number of times the loop re-enters the model, times the context each pass carries. That number is a design decision, and it does not appear in any quote. Schematic, not measured.

> The cost question is not *which model is cheapest*. It is *how many times will this architecture re-enter the model, and how long is the context each time*. That is a design decision you own, not a price you shop.

Three consequences follow for the person making the build decision. The number in your business case is produced by your architecture, which means it is negotiable by design and not by procurement. Most of the available savings come from four or five mechanisms that are architectural rather than commercial, and none of them appear in a vendor quote. And the bill has no natural ceiling, because a loop that re-enters the model has no reason to stop

unless something stops it, which makes cost control a runtime safety control rather than a finance report.

One more, and it sets the frame for everything after it. **Do not model a price decline.** Per-token list prices have fallen steeply for three years, and it is tempting to let that carry a marginal business case. It should not. The capital committed to AI infrastructure has to be recovered, efficiency gains are being spent on more capable and more expensive models rather than on serving today's capability more cheaply, and tokens consumed per task are rising independently of price through longer reasoning traces and more verbose output. Model per-task cost as flat. The bet is asymmetric: model flat and prices fall, you get a windfall; model a decline that does not arrive, and your case fails after you have staffed it.

PART 1

## Two ratios, and the reason the price list is not the story

The **prompt** is everything you hand the model to work from: the instruction, the question, the documents you retrieved, and the running conversation. The **token** is the unit the model reads and writes in, a chunk of text smaller than a word, roughly three-quarters of a word. A page runs several hundred tokens. Tokens matter for one blunt reason. They are the unit you pay in and the unit the model's attention is spent on.

Every call has a cost going in and a cost coming out, and those are not the same size. Two ratios carry almost all of the practical content, and unlike absolute prices they have held across vendors and across three years of price movement. Carry the ratios, not the cents.

**Output costs five to six times input.** The reason is mechanical rather than commercial: input is processed in parallel, output is generated one token at a time. This is the single most consequential number most cost models omit. Self-hosting reprices the asymmetry rather than removing it: decoding is still sequential, so it reappears as throughput cost instead of a line on an invoice.

**A small model costs about five times less than the flagship in the same family**, on both input and output. That ratio is what should drive routing decisions. A tier has also opened above the flagship at roughly twice flagship rates, which puts its output at ten times the flagship's input, comparing across units deliberately to make the point. The question that tier forces is not whether it is better; it is whether the step it takes is worth ten flagship steps, and for most production traffic the answer is no.

| TIER | MODEL | INPUT /1M | OUTPUT /1M | OUT : IN | WHAT THIS TIER IS FOR |
|---|---|---|---|---|---|
| Research | Anthropic Fable 5 | $10.00 | $50.00 | 5.0x | Decisions where being wrong costs more than the run. Its output runs ten times a flagship's input and fifty times a small model's. Not production traffic. |
| Flagship | Anthropic Opus 5 | $5.00 | $25.00 | 5.0x | Genuine multi-step judgment, real ambiguity, work a senior person would have to stop and think about. The default most teams over-use. |
| | OpenAI GPT-5.6 Sol | $5.00 | $30.00 | 6.0x | |
| | Anthropic Sonnet 5 | $3.00 | $15.00 | 5.0x | |
| Mid | OpenAI GPT-5.6 Terra | $2.50 | $15.00 | 6.0x | The workhorse band. Bounded multi-step tasks with tool use, drafting against a spec, structured extraction that needs some judgment. |
| | Google Gemini 3.1 Pro | $2.00 | $12.00 | 6.0x | |
| | Google Gemini 3.6 Flash | $1.50 | $7.50 | 5.0x | |
| Small | OpenAI GPT-5.6 Luna | $1.00 | $6.00 | 6.0x | Classification, routing, extraction, summarization, and the reading steps inside an agent loop. Where most of the saving lives, because most steps are these. |
| | Anthropic Haiku 4.5 | $1.00 | $5.00 | 5.0x | |

Table 1. Illustrative mid-2026 snapshot. Prices change often; verify at the provider's page before building a case on any cell. Sonnet 5 carries introductory pricing of $2.00 and $10.00 through 31 August 2026, after which the figures above apply. Self-hosted open-weight models are absent because their cost structure is fixed infrastructure rather than per token.

Three takes from that table, and they outlast the numbers in it. **The output-to-input ratio is five to six times everywhere**, across three vendors and four tiers, which is why it is safe to build a mental model on. **Price is a tier-selection criterion, not a vendor-selection one.** Within a tier the vendors sit within noise of each other, so choosing a provider on cents is optimizing a rounding error. Choose on eval performance for your task and on latency, then take cost out by moving work down a tier. And **the step between adjacent tiers is roughly five times and the span from top to bottom is roughly ten, like for like**, which means the decision that moves your bill is not which flagship you use. It is how much of your traffic

never needs one. The larger multiples quoted around this market, thirty or fifty times, are almost always comparing one tier's output price against another tier's input price. Check the units before you repeat one.

Where the platform exposes a reasoning-effort setting alongside the model choice, treat the pair as the unit of the decision. A mid-tier model at high effort and a flagship at low effort are different points on the same curve, and for a task class you have eval coverage on, the cheaper point is often good enough.

Two things do not show up on any price list and cost real money. **Reasoning tokens**: a reasoning model, or a standard model set to extended thinking, generates an internal chain of thought billed at full output rates that the user may never see; some providers expose the trace, all of them bill it. A dispute requiring the agent to work through contract terms, transaction history and policy exceptions can generate thousands of invisible tokens before a single visible word. And **images**: a scanned invoice is processed as an image, not as text, often equivalent to several hundred to several thousand text tokens depending on resolution. A customer who uploads a photo, a PDF and a handwritten note has consumed an order of magnitude more input than a text-only interaction before the agent has reasoned at all.

PART 2

## An agent does not call the model once

**Reasoning depth.** Extended thinking is not a free quality upgrade. It is an output-rate multiplier, and it should be decided per task class rather than set once for the product.

**Context accumulation.** Retrieved documents are billed as input on *every* call in the session. Retrieve a ten-page policy at the start of an eight-turn conversation and you pay for it eight times. Retrieval-augmented generation is not free retrieval; it is paid context injection, and the bill compounds with every turn.

**Multimodal inputs.** See above. The failure pattern is a business case modeled on text conversations meeting real customer behavior, which increasingly involves attachments.

**Retries.** The failed attempt usually stays in the context stream, so the second attempt costs more than the first and the third more than the second. The expensive part of a retry storm is the model re-invocation, not the failed downstream call.

**Coordination.** Published figures put roughly thirty-seven percent of tokens in a multi-agent system into coordination: re-establishing context, handing off state, resolving disagreement. A

four-agent system can run about three and a half times the tokens of a single-agent system for the same outcome. Adding agents does not divide the work.

These do not arrive one at a time. A reasoning model handling a complex dispute across ten turns, with retrieved documents, customer attachments and a retried tool call, operates at a cost per interaction that bears no resemblance to the average anyone modeled. Model the compound case as a named scenario with its own frequency, not as a tail you will notice later.

| COST LAYER | TYPICAL RANGE | WHAT IT MEANS FOR THE DECISION |
|---|---|---|
| Floor cost, year one | $175K to $450K for enterprise deployments; evaluation and monitoring alone $60K to $120K | The floor exists whether volume hits projection or half of it. Model both. |
| TCO underestimate | Actual runs 40 to 60 percent above the initial case | Multiply the initial case by 1.5 before presenting it to finance |
| Multi-agent coordination | ~37% of total tokens; ~3.5x for a four-agent system vs. single-agent, same outcome | Agent count is a cost decision, not only a design one |
| Agentic vs. single-shot, same task | 5x to 50x more tokens to complete one case than a single completion of the same task; 100x to 500x for code | Unit economics hold only if the agent closes the case end to end |
| Automation vs. agent, routing | $0.001 vs. $0.10 to $0.20 per task | For routing, an agent is 100 to 200x more expensive per task |
| Automation vs. agent, scheduling | $0.01 vs. $0.60 to $1.20 per task | The differential for routine work is structural, not marginal |
| Cost per task by complexity | Simple $0.002 to $0.04; moderate $0.09 to $0.26; complex $0.50 to $2.00+ | The complexity tax is where most agentic business cases break |

Table 2. Cost layers in enterprise agentic deployments, mid-2026. A mix of published industry ranges and figures from this series; treat every row as a shape to replace with your own measurement rather than as a citable benchmark.

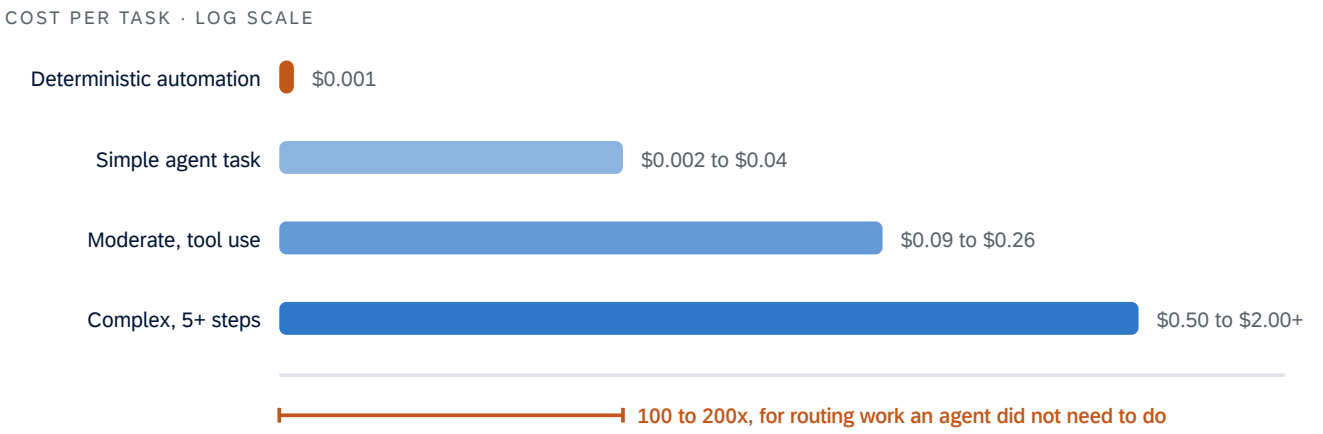



Figure 2. Cost per task by complexity, against the deterministic baseline, on a logarithmic scale. The complexity tax is where most agentic business cases break. The bracket is the number worth carrying into a build decision: for routing work, an agent runs a hundred to two hundred times the cost of logic you could have written down. Mid-2026 industry ranges.

PART 3

# What actually moves the number, and whose call each one is

| LEVER | TYPICAL EFFECT | WHOSE CALL |
|---|---|---|
| **Prompt caching.** The agent replays its system prompt, tool definitions and context on every step; that repeated prefix is what caching is for. | Cached reads at ~1/10 base input price; break-even after one or two reads | Engineering builds it. You own prefix stability. |
| **Model routing.** Send each task to the cheapest model that can do it; cheap model classifies, frontier model judges. | ~5x on the routed share | You define the task classes |
| **Batch processing.** Work with nobody waiting runs on queued endpoints. | ~50% on both input and output | You decide what is truly interactive |
| **Shortening the loop.** Fewer model re-entries per task; deterministic steps replace model calls. | Often 30 to 60%, highest ceiling of any lever | You and the architect, jointly |
| **Context engineering.** Retrieve spans not documents; compact the running conversation; drop what the step does not need. | Large and compounding on long sessions | You, with the data team |
| **Gating reasoning mode.** Extended thinking on per task class, not per product. | Removes an output-rate premium from the easy majority | Yours |

| LEVER | TYPICAL EFFECT | WHOSE CALL |
|---|---|---|
| **Output shape.** Ask for the decision, not the essay; structured output over prose you will parse. | Direct, at the expensive end of the bill | Yours |
| **Retry hygiene.** Circuit breaker per tool, backoff with jitter, summarize failed attempts rather than carrying them. | Bounds the tail rather than the average | Engineering; you set the thresholds |
| **Multimodal discipline.** Extract once at the resolution the task needs; carry text forward. | Order of magnitude on attachment-heavy traffic | Shared |
| **Not using an agent.** Deterministic logic has no inference cost at all. | 100 to 200x on the tasks it applies to | Yours, and it is the build decision itself |

Table 3. Ordered roughly by savings-to-effort for a typical enterprise agent.

Four of these are worth a sentence more, because they are the ones a PM decides rather than reviews.

**Caching pays on traffic shape, not on architecture alone.** The saving is real for sustained interactive sessions and largely absent for bursty, low-frequency work, because caches expire on short windows measured in minutes. The arithmetic is worth checking rather than taking on trust: cached reads run at roughly a tenth of base input price, while writing to the cache costs a premium over it, a little above list for a short window and roughly double for a long one, which is what puts break-even at the first or second read. It also requires a stable prefix: a system prompt that interpolates a timestamp, session ID or user name at the top invalidates the cache on every call. Put volatile content at the end, and do not book the saving in an estimate before someone has checked your inter-call gap.

**Shortening the loop has the highest ceiling and no vendor.** Nobody sells it. It requires reading the trace of one real task, naming every model re-entry, and arguing with each one. Classification steps often do not need a model. Formatting steps almost never do. Steps that exist because an earlier step returned something unusable are a schema problem wearing a cost costume.

**How you write the success criterion is a cost driver.** An unmeasurable or unachievable goal is a loop generator: an agent told to reach a target it cannot verify will keep trying, and an agent told to reach one that is unattainable will keep trying forever. Make the criterion measurable, make sure the agent has a tool that can measure it, and give it an explicit budget, because the agent has no price model and cannot self-regulate on cost. This is Executable Brief work, and it belongs to you.

**Deciding not to use an agent is the largest number here.** Deterministic automation costs about a tenth of a cent for routing against ten to twenty cents for an agent. When a problem can be written down in advance, writing it down is not the lesser solution. This applies inside a product as well as to the build decision: a hybrid that handles the predictable eighty percent deterministically and routes the residue to an agent is usually the correct shape, and it is rarely the shape a demo produces.

Two things you cannot model. Volume and committed-use discounts are negotiated case by case and unpublished, so treat any you win as margin rather than as an assumption. And pinning inference to a data-residency region costs about ten percent more, which is the one place a governance choice shows up as a line on an invoice.

PART 4

## The bill has no ceiling unless you build one

Two agents in a production research system started talking to each other. One analyzed, the other verified, and a misclassified error taught the pair to treat "we disagree" as "retry with different parameters" rather than "stop." They retried week after week, passing the full conversation back and forth, the weekly bill climbing from pocket change into five figures. Nothing malfunctioned and every dashboard stayed green, because the system was doing exactly what it was built to do at a rate nobody had capped.

An agent is not a batch job that runs a bounded operation and halts. It is a loop that re-enters the model, carries growing context, retries on failure, and calls tools that fail in ways that make it retry harder. Uncapped, it spends faster than a human review cycle can react.

> A budget read after the call is an accountant. A budget checked before the call is a brake. Every runaway shares one mechanic: hundreds or thousands of paid calls go out before a human sees an alert.

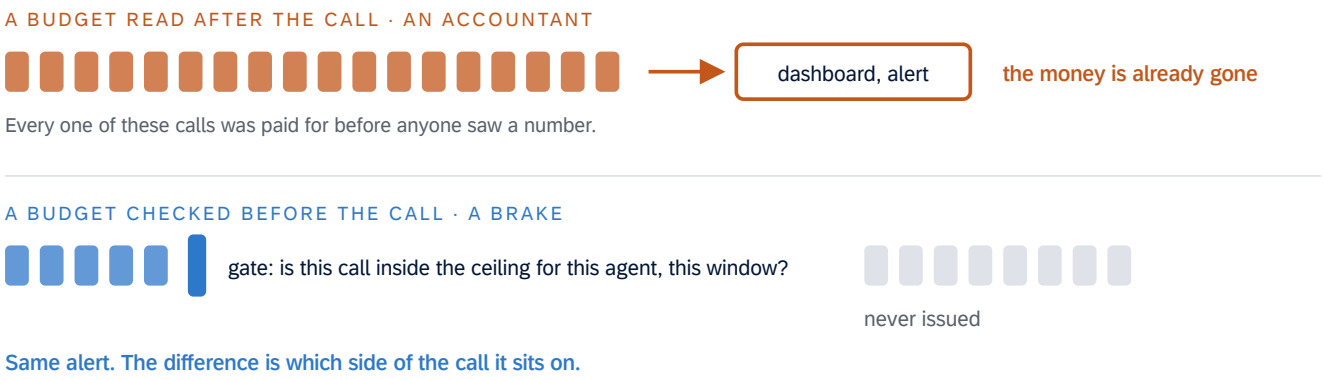



Figure 3. The same alert, on two sides of the call. Read afterwards it is a record of spending that already happened. Checked before the call, against the running total, it is the only thing that stops a runaway while the money is still in the room.

## The four controls, and who owns them

**Hard ceilings** on tokens, requests and wall-clock, per agent per window, checked before the call. Not the per-call cap engineering already sets; these cap cumulative behavior across a run and across runs, which is where the money goes.

**A kill switch** outside the agent's runtime, so a misbehaving agent cannot suppress it. **A circuit breaker** per tool, tripping on infrastructure failures only. **A burn-rate alert** on spending velocity, not on the monthly total.

Engineering owns the mechanisms. You own the numbers and the escalation: what the three ceilings are, what velocity counts as abnormal, and who gets paged at each tier. Those are product decisions disguised as infrastructure settings, and left implicit they default to whatever the framework shipped, which for cost is usually nothing.

Be precise about what the platforms do and do not give you, because the loose version of this claim is easy to refute. They do offer account-level monthly spend ceilings and budget alerts, and on two of the large clouds a budget policy can deny further calls once a threshold is crossed. What none of them ships is a per-agent, per-window ceiling evaluated before the call executes, which is the only control that stops one misbehaving agent rather than the whole account after the money is gone. Platforms enforce throughput, requests and tokens per minute; cost and consequence controls need business logic only you can write, because only you know what this agent is worth and what number should make a phone ring. The danger is not the gap. It is assuming the infrastructure is protecting you, shipping on that assumption, and finding out on a Saturday. A useful tell at a launch review: ask what the daily ceiling is and who gets paged when hourly burn triples. "We have a monthly budget alert" and a shrug means the agent has no brakes, only a receipt.

PART 5

# The comparison that decides it, in six lines

**The break-even.** An agent is worth its cost when the value of the task, times how often it runs, exceeds the fully loaded per-task cost: tokens times the architecture multiplier, plus the human supervision the agent still requires. Teams compare sticker price to an hourly rate and conclude the agent is obviously cheaper. Often it still wins. Sometimes, counted properly, it does not, and that is a finding worth having before you build.

**The comparison is against the best alternative, not against nothing.** The alternative might be traditional automation, a constrained non-agentic AI feature, or a human with better tooling. The agent has to beat that, not beat the empty field.

## The calculation, in six lines

You cannot do this from a price list. Lines 1 to 3 come from reading the trace of one real task, which means the first version of this number arrives after the prototype and not before it. Estimate them before you build, then replace them with measurements at the first opportunity.

| # | LINE | WHERE IT COMES FROM |
|---|---|---|
| 1 | Model re-entries per task, *N* | Count them in a trace. Name each one and ask whether it needs a model at all. |
| 2 | Average input tokens carried per re-entry, *C* | System prompt, tool definitions, retrieved content, conversation so far. This grows through the task; use the average, not the first call. |
| 3 | Average output tokens per re-entry, *O* | Include reasoning tokens if extended thinking is on. They bill at output rates and do not appear in the response. |
| 4 | Compute per task = *N* × (*C* × input price + *O* × output price) | Priced per tier, per the routing you actually intend. Apply the caching discount only where the prefix is stable and the traffic is dense enough to hit a live cache. |
| 5 | Loaded per task = compute + supervision + downstream rework | Supervision is the escalation rate times the cost of a review. Rework is the correction the output needs before it is usable. |
| 6 | Case = loaded × volume × 1.5, plus the year-one floor | Compare against the best alternative at the same volume. Hold per-task cost flat across the horizon. |

Line 4 is where teams stop. Lines 5 and 6 are where the answer actually changes.

**Which line dominates depends on volume, and it flips.** At low volume the year-one floor dominates and the token bill is noise; arguing about model choice on a thousand tasks a month is arguing about the wrong line, and the real question is whether the fixed cost of running an agentic product properly is worth carrying at all. At high volume the multiplier dominates and the floor amortizes to nothing; there, architecture is the entire decision. Run the case at your projection and at half of it, because the two halves of that sentence give opposite advice and you need to know which side of the crossover you are on.

**The 1.5x correction.** Actual total cost of ownership runs forty to sixty percent above the initial case in typical deployments, because teams anchor on the vendor quote and miss orchestration, oversight, eval maintenance, and the model-update cycle that arrives every three to six months and costs a re-validation each time. Multiply before you present. If that makes the case fail, it was already failing and you found out cheaply.

**Count the downstream rework.** This is the line most often missing entirely. Where the agent's output requires more verification, correction or refactoring than the alternative's, that cost is real, it lands on a team that did not sign the business case, and no token meter reports it. For code-generating agents it can exceed the generation cost. Put a number on it or state explicitly that you have assumed it to be zero.

**The multiplier does not transfer between task classes.** Five to fifty for ordinary work, one hundred to five hundred for code. Benchmark your classification agent, extrapolate to your drafting agent, and you can be wrong by an order of magnitude. Measure per class. This mistake looks like diligence, which is why it survives review.

**The brownfield inversion.** In greenfield the floor is whatever the agent costs to run. In brownfield, replacing a step in a system that already works, the agent must beat the marginal cost of the human step *and* justify integration, change management and supervision that a greenfield build never paid. Meanwhile the licence component of that floor has been falling as platform vendors fold AI capability into base subscriptions, so the licence question and the loaded-cost question now move in opposite directions and have to be modeled separately.

**Outcome-based pricing,** where a vendor bills per resolved case rather than per token, simplifies the question to: at what resolution rate does this vendor stay profitable, and does that match what the workflow can accept? Where it is not on offer, you are holding the failure cost yourself, and that is its own line.

### Worked example: the refund agent

The bare model cost per refund decision is a couple of cents, far below the loaded cost of a human's time. That comparison is the trap.

*Count the re-entries.* One refund decision re-enters the model about eight times: classify the request, pull and read the order, weigh the account history, check the policy, run the fraud signal, draft the decision, format the audit record, and produce the escalation package when it escalates. Each pass carries a few thousand tokens of context. The multiplier is roughly ten times the bare cost, so a decision whose model cost is two cents has a real compute cost nearer twenty. That is before a human touches it.

*Add the supervision.* Say the agent resolves seventy percent autonomously and escalates thirty. The seventy percent cost cents each. The thirty percent still cost a human review, so the agent does not remove that cost; it concentrates the human on the cases where judgment pays. Break-even is not "agent cheaper than human per task." It is "fully loaded cost, including supervising the escalated minority, below the cost of humans reviewing all of it, with the wrongful-refund risk priced in." Here it clears, but only because the escalation rate is low enough that the supervised minority is truly a minority.

The figures are illustrative and will move. The method is durable: count the model re-entries and the context each one carries, and you have the multiplier, which is the number that actually decides the case.

PART 6

# What to ask, and when

### At the build decision

- What is the best non-agent alternative, and by how much must the agent beat it?
- How many model re-entries for a typical task? For the worst one?
- Which of those could be deterministic?
- What is the loaded per-task cost, including supervision and downstream rework?
- Has the case been multiplied by 1.5, with per-task cost held flat rather than declining?
- Greenfield or brownfield, and which floor applies?

### At the design review

- Is the prefix stable enough to cache, and does our traffic shape realize the saving?
- Which task classes route to a small model, and on what signal?
- Which traffic is interactive by requirement rather than by habit?
- Is extended thinking on per class, or on by default?
- Does retrieval inject spans or whole documents, and does anything ever remove context?
- Is every success criterion measurable, with a tool that can measure it?

### At the launch review

- The three ceilings, as three numbers: tokens per run and per window, requests per window, wall-clock per run. Enforced before the call or logged after it?
- Does the kill switch live outside the agent's runtime, and who is paged when hourly burn triples, by name?
- What is per-task cost at the trajectory level, including coordination overhead and human review time? If nobody can produce it, that is a finding about a surface nobody designed.

List price is the wrong number in both directions. Too high, because caching, batching and routing move real traffic well below it. Too low, because the architecture multiplier, the supervision, the floor cost and the downstream rework are all absent from it. The number that decides your product is produced by your design, which is the good news: it is yours to change.

---

**Sources and status.** Drawn from the *Agentic AI for Product Leaders* series by Yoram Friedman: *Agentic AI for Busy Product Managers* (second edition), chapter 4 on suitability and cost; *Why Agentic AI Products Fail*, chapter 3 on the token bill and chapter 10 on operational guardrails; *The Agentic AI Team*, on where the cost decision sits in the organization; and *The Agentic AI Practitioner* (forthcoming, September 2026), on the loaded cost model in the Human Brief. Benchmark ranges are industry figures stated as ranges and stamped to mid-2026, and should be replaced with your own measurements. The model prices in Table 1 are a snapshot and will be wrong before this document is. The ratios beside them have held across vendors and across three years of price movement, which is why the argument is built on the ratios and the table is offered only as a starting point. The runaway-loop and coordination figures are published incident and research data. The refund agent is an illustrative composite.