

Where \$10,920 a month of AI spend went, and which \$2,180 of it was avoidable.

Worked example on synthetic data

SYNTHETIC DATA

Worked example on synthetic data — the workload, invoices and logs in this document are synthetic; the patterns and mechanisms are drawn from real audits.

Client profile

Client	Client A
Sector	B2B SaaS
Engineers	40
Geography	United States
Scope	Four providers, nine billed services, one 30-day invoice window.
Currency	USD as billed

Period and references

Measurement window	20 July 2026 to 18 August 2026 (30 days)
Invoice	INV-2026-08-0417
Verification window	1 September 2026 to 30 September 2026 (30 days)
Invoice	INV-2026-09-0431
Report date	2026-10-06
Price table captured	2 October 2026

Identified is not the same as realized. One of those numbers lives on a slide; the other has to survive the next invoice.

Diagnostic depth

- Structural.** Structural diagnosis covering all four providers: every invoice line was attributed to a service, a feature and a provider, and reconciled back to the invoice total.
- Behavioural.** Behavioural diagnosis on two features only — the support assistant and the agent orchestrator — where request-level gateway logs covered the full window.
- Not covered.** Prompt-content analysis, per-end-customer attribution, training and fine-tuning spend, and any workload that does not appear on an invoice are out of scope for this document.

DATA HANDLING

No credentials, no production access, no code

ACCESS

Read-only. Nothing was changed to produce this document.

REPRODUCIBILITY

Every figure is generated from one data file, published with the report.

Summary and contents

Client A spent \$10,920 on AI and LLM services in the 30 days ending 18 August 2026, across 4 providers and 9 billed services. We attributed all of it, found \$2,180 of avoidable spend in two mechanisms, and watched part of it come off the next invoice.

MONTHLY AI SPEND \$10,920 As invoiced, 4 providers, 9 services, one 30-day window.	IDENTIFIED WASTE \$2,180 20.0% of spend. Seven findings, two mechanisms.	IMPLEMENTED AND VERIFIED \$1,340 Confirmed on the following invoice, normalised for volume.
---	---	--

Read the two numbers separately. \$2,180 was identified. \$1,340 of it was implemented inside the engagement and then confirmed on the invoice from the verification window. The remaining \$840 is still identified and still on the bill; it is never added to the verified figure anywhere in this document, and it is never described as a saving.

Contents

1	Spend map Every invoice line attributed to a feature, a provider and a service.	Pages 3–4
2	Unit economics Three unit costs, each derived from one invoice line and one metered volume.	Page 5
3	Waste list Seven findings: phenomenon, evidence, amount and confidence.	Pages 6–8
4	Optimization roadmap Ordered by implementation cost, with what was actually verified.	Page 9
5	What we do not recommend Five things we looked at and declined, with the arithmetic.	Pages 10–11
6	Method appendix Sources, sampling, assumptions, limits and the reconciliation tables.	Pages 12–14

How to read this document

- ☒ Implemented and verified on the following invoice ☐ Identified, not yet implemented ☐ Preventive control, nothing at stake
- Every amount is a **monthly** figure for the measurement window unless it is labelled otherwise.
 - Every number is set in a monospace face so that it can be found and re-checked without reading the prose around it.
 - Confidence is stated per finding with the reason for it. Nothing here is rated Low; anything that would have been was left out instead.
 - Section 5 lists what we declined to recommend. It is the section that keeps section 3 honest.
 - The reconciliation tables in section 6 add this document's own numbers up in public.

1. Spend map

Client A runs four production workloads on someone else's GPUs. The map below starts from the invoice, because the invoice is the only list of AI spend that is guaranteed to be complete: it contains every line somebody is paying for, including the ones nobody remembers building.

By feature

FEATURE	PROVIDER(S)	MONTHLY	SHARE
Support chatbot Customer-facing assistant answering product questions inside the application.	Provider 1	\$4,180	38.3%
Batch summarisation Nightly summarisation of support tickets and release notes.	Provider 1, Provider 4	\$2,640	24.2%
Retrieval and embeddings Embedding of customer documentation plus query-time retrieval and reranking.	Provider 2	\$1,920	17.6%
Agent retries (avoidable) Spend generated by repeat attempts rather than by new work.	Provider 3	\$980	9.0%
Dead model routes Routes and endpoints that are billed but no longer serve a user-visible feature.	Provider 2, Provider 3, Provider 4	\$1,200	11.0%
Total	4 providers	\$10,920	100.0%

Where the money is concentrated



Bars are drawn to the share of monthly spend. The two amber bars are the mechanisms described in section 3; they are the only part of this chart that is not doing work for a customer.

What the shape tells us

- **One line dominates.** The support chatbot is \$4,180, or 38.3% of spend. It is also the only line a customer can see, which is why section 5 refuses the obvious way to reduce it.
- **Two of the five lines produce nothing.** Agent retries and dead routes together account for \$2,180, 20.0% of the bill. One is work repeated; the other is capacity nobody calls.
- **Spend is spread over 4 providers.** No single vendor holds a majority in the sense that matters: the largest is 44.1% of the total, which is enough concentration to negotiate with and not enough to be trapped by.

1. Spend map (continued)

By provider

PROVIDER	ACCOUNT	FEATURE(S) SERVED	MONTHLY	SHARE
Provider 1 Frontier model API (redacted)	acct_••••7f2	Support chatbot / Batch summarisation	\$4,820	44.1%
Provider 2 Hyperscaler AI services (redacted)	acct_••••c19	Retrieval and embeddings / Dead model routes	\$2,440	22.3%
Provider 3 Inference platform (redacted)	acct_••••a04	Agent retries (avoidable) / Dead model routes	\$1,420	13.0%
Provider 4 Managed document AI (redacted)	acct_••••3b8	Batch summarisation / Dead model routes	\$2,240	20.5%
Total		9 services	\$10,920	100.0%

By service

SERVICE	ENDPOINT	PROVIDER / FEATURE	MONTHLY	SHARE
Chat completion, production Streaming chat with tool calls. 20,900 sessions in the window. This is the only line the customer would notice if it changed.	prod-chat-v3 (redacted)	Provider 1 Support chatbot	\$4,180	38.3%
Summarisation, on-demand batch Overflow jobs sent to the frontier API whenever the nightly window overruns.	batch-sum-a (redacted)	Provider 1 Batch summarisation	\$640	5.9%
Embeddings, index build Re-embedding of changed documents. A full index rebuild also ran twice inside the window.	embed-index (redacted)	Provider 2 Retrieval and embeddings	\$980	9.0%
Vector search, query time Per-query charges, including the reads consumed by the reranker.	vector-query (redacted)	Provider 2 Retrieval and embeddings	\$940	8.6%
Legacy completion routes Two deprecated entries were still present in the gateway route table and still receiving traffic.	legacy-completion-v1 and v2 (redacted)	Provider 2 Dead model routes	\$520	4.8%
Agent orchestrator Planner and executor loop for internal automations. All repeat attempts on this service are counted here.	agent-orchestrator (redacted)	Provider 3 Agent retries (avoidable)	\$980	9.0%
Provisioned throughput endpoint Billed on a monthly minimum. No feature has routed to it since the route table was rebuilt.	pt-endpoint-eu (redacted)	Provider 3 Dead model routes	\$440	4.0%
Managed document pipeline Per-page pricing covering the same corpus as the batch service above.	doc-pipeline-b (redacted)	Provider 4 Batch summarisation	\$2,000	18.3%
Idle cloned deployment Cloned for a project that was cancelled. Still reserved, still billed, never called.	deploy-clone-07 (redacted)	Provider 4 Dead model routes	\$240	2.2%
Total			\$10,920	100.0%

Identifiers are redacted; in a live engagement this column carries the client's own account and endpoint names. Each of the 9 services was attributed to exactly one feature and one provider, so no share of any line had to be apportioned by estimate.

2. Unit economics

Three unit costs, each derived by dividing one invoice line by a volume that the provider itself meters. Nothing here is modelled from product analytics, so the arithmetic can be repeated from the invoice and the usage export alone.

COST PER CHATBOT SESSION \$0.20 \$4,180 ÷ 20,900 chat sessions in the 30-day window	COST PER 1,000 DOCUMENTS SUMMARISED \$1.50 \$2,640 ÷ 1,760,000 documents summarised in the window	COST PER 1,000 RETRIEVALS \$0.60 \$1,920 ÷ 3,200,000 retrieval calls in the window
--	--	---

The derivation, line by line

UNIT	FEATURE	FEATURE SPEND	VOLUME	DERIVED PRICE	TOKENS / UNIT	PER MILLION TOKENS
Cost per chatbot session Feature spend divided by metered sessions.	Support chatbot	\$4,180	20,900	\$0.20	10,000	\$20.00
Cost per 1,000 documents summarised Feature spend divided by documents processed, then expressed per thousand.	Batch summarisation	\$2,640	1,760,000	\$1.50	800	\$1.88
Cost per 1,000 retrievals Feature spend divided by metered retrieval calls, then expressed per thousand.	Retrieval and embeddings	\$1,920	3,200,000	\$0.60	48	\$12.50

Basis of each unit

- **Cost per chatbot session.** 10,000 tokens per session, input plus output, averaged across the window.
- **Cost per 1,000 documents summarised.** 800 tokens per document, input plus output, averaged across both summarisation services.
- **Cost per 1,000 retrievals.** 48 tokens per retrieval: 32 embedding tokens plus 16 reranker tokens, averaged across the window.

Cross-checks

- The unit price is derived from the invoice line rather than measured separately, so multiplying it back by the volume reproduces the line exactly. That is a consistency check on our arithmetic, not independent evidence that the volume is right.
- The two summarisation services carry very different prices per unit of work, but they are metered differently, on different page scopes. We deliberately claim no waste from that gap; the reasoning is in the next section.
- The blended cost per million tokens differs between the frontier assistant and the batch pipeline by a factor of about 10.7. That is the expected shape when one workload needs tool use and the other does not.

Limits of these three numbers

- Unit costs are per feature and per provider. A workload that spans two providers has one blended price in this document and no per-provider split.
- Token counts are the providers' own metered counts as they appear on the invoice. We did not re-tokenise prompts to verify them.
- A workload that is not metered does not appear here at all. This is a spend map, not an inventory of everything the company runs.

3. Waste list

Seven findings, grouped into two mechanisms. Each finding carries the evidence that produced it, the amount it represents, and a confidence level with the reason for that level. Amounts are monthly and are stated at the unit prices captured on the date in the appendix.

Two mechanisms carry the whole of it: **Agent retries**, where work was repeated that could not have succeeded, and **Dead model routes**, where capacity is billed with nothing routed to it. Each finding below states the phenomenon, the evidence that produced it, the amount, and a confidence level with its reason.

Category A: Agent retries (avoidable) — \$980

A retry that eventually succeeds is cheaper than a failed customer request, so retries are not waste by definition. We counted only attempts that could not have succeeded, and attempts that repeated work the system had already completed.

✓ \$580 implemented and verified · ▲ \$400 identified, not yet implemented · 45.0% of all identified waste

F1 Retry storm with no backoff and no cap on Agent orchestrator (agent-orchestrator (redacted)), \$980 of this line		\$320
PHENOMENON	Failed attempts were re-sent immediately, with no delay and no ceiling on the number of tries. When one downstream tool slowed down, the orchestrator responded by pushing more requests into the same degraded path, so a queue formed and then drained at full price.	
EVIDENCE	Gateway request logs with attempt counters, plus the orchestrator configuration read as text. - Gateway logs, full window: 1,240 orchestrator sessions averaged 6.1 attempts per logical task against 1.4 attempts on quiet days. - The two worst days, 14 and 15 August, carry 32.7% of this service's monthly spend, or \$320. - Neither exponential backoff nor a retry budget appears anywhere in the orchestrator configuration.	
CONFIDENCE	High Attempt counts are logged per request ID and the cost per attempt is a published per-token price; the two agree to within 1.8%.	
FIX	Exponential backoff with jitter, a hard cap of two attempts per logical task, and a per-session retry budget.	
STATUS	✓ Implemented and verified on the following invoice Attempts per logical task fell to 1.5 after the change. The reduction was measured on the September invoice with volume held flat.	

F2 Retries issued on responses that could never succeed on Agent orchestrator (agent-orchestrator (redacted)), \$980 of this line		\$260
PHENOMENON	The retry policy never read the response class. Client errors that cannot succeed on a second attempt, such as malformed tool arguments and requests rejected for their own shape, were re-sent as if they had been transient failures.	
EVIDENCE	Gateway response-class histogram joined to attempt counters. 8.4% of orchestrator calls returned a non-retryable client error and were nevertheless re-sent. - Those repeats consumed 26.5% of this service's monthly spend, or \$260, while producing no successful work. - The retry wrapper was applied at the HTTP client, upstream of any response-class check, so no caller could have prevented it.	
CONFIDENCE	High Response classes and attempt counts are both logged directly; the join is deterministic, with no sampling.	
FIX	Classify before retrying: retry transport failures and rate limits only, fail fast on everything else and surface the error to the calling feature.	
STATUS	✓ Implemented and verified on the following invoice Non-retryable repeats stopped in the same release as the backoff change and stayed stopped through the verification window.	

3. Waste list (continued)

F3 Duplicate tool execution, including replay after a broken stream on Agent orchestrator (agent-orchestrator (redacted)), \$980 of this line		\$400
PHENOMENON	Turns interrupted mid-stream were replayed from the beginning rather than resumed. Any tool the assistant had already called ran a second time with the whole context attached again. The customer saw one answer; the invoice saw two or three.	
EVIDENCE	Turn-level traces with tool-call fingerprints, joined to stream-interruption events. • 3.1% of completed turns contain at least one duplicated tool call with an identical argument payload. • Replay after a stream interruption accounts for 40.8% of this service's monthly spend, or \$400. • There is no idempotency key on the tool-call path, so a second call cannot be recognised as a duplicate of the first.	
CONFIDENCE	Medium Duplicates are countable, but separating replay from a legitimate second call relies on matching argument payloads, which is probabilistic rather than deterministic.	
FIX	Idempotency keys on every tool call, and resume from the last checkpoint instead of replaying the turn when a stream breaks.	
STATUS	⚠ Identified, not yet implemented This one needs a change to the orchestrator rather than a configuration edit, so it is carried as identified spend, not recovered spend.	
F4 Deprecated route still receiving live traffic on Legacy completion routes (legacy-completion-v1 and v2 (redacted)), \$520 of this line		\$300
PHENOMENON	A completion route for a model generation the vendor has since superseded was still listed in the gateway route table, and was still being selected for a small share of production chat traffic at a worse price per token than its replacement.	
EVIDENCE	Gateway route table export plus the matching invoice lines. • Route table snapshot taken on the last day of the window: two deprecated entries present, both older than six months. • The larger of the two carries 25.0% of this category's spend, or \$300, while being selected for only 1.1% of chat sessions. • Selection came from a fallback rule written before the current model catalogue existed.	
CONFIDENCE	High The route is named on the invoice and in the configuration, and both name the same endpoint.	
FIX	Remove the entry from the route table and delete the fallback rule that reintroduces it.	
STATUS	✓ Implemented and verified on the following invoice The endpoint stopped appearing on the invoice in the verification window.	
F5 Superseded batch route left in the table on Legacy completion routes (legacy-completion-v1 and v2 (redacted)), \$520 of this line		\$220
PHENOMENON	A second deprecated entry existed only for batch work and had been superseded by an earlier routing change. Nothing referenced it, but the same fallback rule kept it warm enough to bill every month.	
EVIDENCE	Gateway caller attribution joined to the invoice line for the same endpoint. • 340 calls in the window, every one of them from the fallback path rather than from a named caller. • Accounts for 18.3% of this category's spend, or \$220.	
CONFIDENCE	High Every call is attributable to a single rule, and the endpoint appears on exactly one invoice line.	
FIX	Delete the route and rewrite the fallback rule in the same change so the entry cannot come back.	
STATUS	✓ Implemented and verified on the following invoice Both deprecated entries were removed together; the invoice line disappeared with them.	

3. Waste list (continued)

Category B: Dead model routes — \$1,200

Routes and endpoints that are billed on schedule but no longer serve a user-visible feature. This category is the cheapest to recover and the easiest to reintroduce, which is why the roadmap ends with a control rather than with a cleanup.

✓ \$760 implemented and verified · ▲ \$440 identified, not yet implemented · 55.0% of all identified waste

F6 Reserved endpoint with no routed traffic on Provisioned throughput endpoint (pt-endpoint-eu (redacted)), \$440 of this line		\$440
PHENOMENON	One provider bills a monthly minimum for reserved capacity. The endpoint is still reserved and still billed on schedule, but no feature routes to it and its request counter has not moved since the route table was rebuilt.	
EVIDENCE	Invoice line for the reservation, plus the provider's own metered-request report for the same endpoint. - The monthly minimum appears on the invoice for the full window: \$440, or 36.7% of this category. - Metered requests recorded against the endpoint over the whole window: zero. - The reservation sits inside a term commitment, so it cannot be cancelled unilaterally before the term ends.	
CONFIDENCE	Medium The billing is unambiguous and the request counter is authoritative. What we cannot see is the termination terms, so the recoverable amount depends on a commercial conversation this document was not party to.	
FIX	Re-scope the reservation to a workload that exists, or let it lapse at the end of term. If it must be paid for until then, move the overflow batch job onto it so the money buys something.	
STATUS	▲ Identified, not yet implemented The technical half is a one-line routing change. The half that unlocks the amount sits in a contract.	

F7 Cloned deployment from a cancelled project on Idle cloned deployment (deploy-clone-07 (redacted)), \$240 of this line		\$240
PHENOMENON	A deployment was cloned for a pilot that was cancelled before it served any traffic. The clone was never torn down and has been billed for reserved capacity ever since.	
EVIDENCE	Read-only deployment inventory joined to the reserved-capacity invoice line. - Deployment inventory read as text: the clone has no traffic history at any point in its life. - Reserved capacity cost for the window: \$240, or 20.0% of this category.	
CONFIDENCE	High A deployment with no traffic history that appears on an invoice cannot be doing anything else.	
FIX	Delete the deployment, and add a monthly inventory diff so the next clone cannot outlive its project.	
STATUS	✓ Implemented and verified on the following invoice Deleted during the engagement; the reserved-capacity line dropped in the verification window.	

Closing the section

MECHANISM	FINDINGS	IDENTIFIED	VERIFIED	STILL OPEN	SHARE OF SPEND
Agent retries (avoidable)	3	\$980	\$580	\$400	9.0%
Dead model routes	4	\$1,200	\$760	\$440	11.0%
Identified waste	7	\$2,180	\$1,340	\$840	20.0%

\$1,340 of the \$2,180 identified waste was implemented and confirmed on the following invoice; \$840 is still identified and still on the bill. Section 6 reconciles the 7 findings against the 5 invoice lines, and section 4 shows what happened when each one was acted on. Two things are deliberately not claimed as waste anywhere in this document: the difference between the two services that summarise the same corpus, and any vendor price movement inside the period.

4. Optimization roadmap

Seven items, ordered by what it costs to implement. Configuration-only items sit above anything that needs a code change, and both sit above anything that needs a contract. Each item names the findings it closes and the spend at stake.

1	Stop retrying responses that cannot succeed Move the retry decision below the response-class check so that anything which is not a transport failure or a rate limit fails immediately and surfaces the error to the calling feature. Verification: Attempt counters for the orchestrator, plus the corresponding invoice line on the following invoice. · Configuration-only · About 1 hour of engineering time · findings F2 · ✓ Implemented and verified on the following invoice	\$260
2	Delete the two deprecated routes and the fallback rule that keeps them warm Remove both entries from the gateway route table and rewrite the fallback rule in the same change, so that the entries cannot be reintroduced by a rule nobody is looking at. Verification: Route table snapshot after the change, plus the disappearance of the endpoint's invoice line. · Configuration-only · About 1 hour of engineering time · findings F4, F5 · ✓ Implemented and verified on the following invoice	\$520
3	Add backoff, jitter and a retry budget Exponential backoff with jitter, a hard cap of two attempts per logical task, and a per-session retry budget. The budget is what stops a degraded downstream service from being answered with more load. Verification: Mean attempts per logical task, compared against the same statistic in the baseline window. · Configuration-only · About 2 hours of engineering time · findings F1 · ✓ Implemented and verified on the following invoice	\$320
4	Delete the cloned deployment, and add a monthly inventory diff Delete the deployment outright. The diff is the part that lasts: a monthly comparison of reserved capacity against traffic history, reviewed by a named person. Verification: Reserved-capacity invoice line, plus the first inventory diff run. · Configuration-only · About 2 hours of engineering time · findings F7 · ✓ Implemented and verified on the following invoice	\$240
5	Idempotency keys and resume-from-checkpoint on the tool path Attach an idempotency key to every tool call and resume an interrupted turn from its last checkpoint instead of replaying it. The replay is what turns one customer answer into three billed executions. Verification: Duplicate tool-call rate per completed turn, measured over a full week after release. · Small project · 1 to 2 weeks · findings F3 · ⚠ Identified, not yet implemented	\$400
6	Re-scope or release the reserved endpoint Ask the provider to re-scope the reservation to the workload that actually exists, or let it lapse at the end of term. Until either happens, route the overflow batch job onto it so that the reservation carries real work. Verification: Contract amendment or lapse date, plus the metered-request counter on the endpoint. · Commercial and configuration · 2 to 4 weeks, mostly waiting · findings F6 · ⚠ Identified, not yet implemented	\$440
7	A route hygiene control, so the cleanup does not have to be repeated Flag any route with no traffic for 14 days, and any reserved capacity with no traffic for 30 days, into the same review that already looks at cost. Canary every route change so that a fallback rule cannot silently keep an old entry alive. Verification: The control's own output, reviewed monthly. · Project, ongoing · A few days to set up, then minutes a month · findings none · 🛡 Preventive control. No incremental dollars at stake	\$0

What came off the bill. 4 items worth \$1,340 were implemented inside the engagement and confirmed on the following invoice. 2 items worth \$840 remain identified and are not counted anywhere as a saving. Total at stake: \$2,180, 20.0% of the \$10,920 spent in the window.

Sequencing. Items 1 to 4 are configuration and recover \$1,340 a month for hours of engineering time rather than weeks, which is why they come before anything that needs a sprint. Item 5 is the only finding that needs code, and its fix also removes a class of user-visible double answers. Item 6 unlocks a \$440 finding but is gated by a contract, so it is scheduled as a commercial conversation with a technical half that can ship immediately. Item 7 carries no dollars; it is the reason items 2 and 4 will still be true in a year.

Items 1 to 4 are verified: they were implemented inside the engagement and the reduction appears on the invoice from the verification window, normalised for the 2.4% volume drift. Items 5 and 6 are carried as identified spend at \$840. We are not counting them as savings, and we are not counting any provider price movement in the period as a result of our work.

5. What we do not recommend

This is the part of the audit that saves the most money, and it is the part nobody puts in a proposal. Each item below was looked at properly and then declined, with the arithmetic that produced the decision.

Do not replace the support assistant with a smaller or self-hosted model to cut the largest line

Verdict. Declined. The largest line is the largest because it is the only line a customer can see.

1. The assistant is 38.3% of monthly spend, so it is the obvious target and the first thing a cost tool will flag.

2. The saving would come entirely from a lower price per token, paired with a quality change that this dataset cannot measure. We do not price those.

3. Two things would decide it, and neither exists here: an offline evaluation set with acceptance thresholds, and the measured cost of a bad answer in escalation time.

4. Both are inexpensive to build. Neither existed when we looked, which is itself a finding about the organisation rather than about the model.

WHAT WE WOULD DO INSTEAD

Fix the retry path first and re-measure. If the assistant is still the biggest line afterwards, build the evaluation set before touching the model, so that the next decision has evidence on both sides of it.

Do not self-host inference to escape per-token pricing

Verdict. Declined. Your portable spend is below the break-even point, and the comparison is already generous to self-hosting.

1. Two nodes for availability at \$2.75 per hour across 730 hours is \$4,015 a month. One node is not an option: a single accelerator with no second copy turns a hardware fault into an outage.

2. A quarter of an engineer's fully loaded time is \$3,000 a month, and platform overhead for networking, storage and monitoring adds \$450.

3. That puts the fixed cost at \$7,465 a month before the first token is served.

4. The workloads that could genuinely move to open weights are batch summarisation and retrieval, \$4,560 a month. The support assistant needs a frontier model with dependable tool use, so it is not portable at any realistic quality bar.

5. Break-even therefore sits at \$7,465 a month of portable spend. You are at \$4,560, a shortfall of \$2,905.

6. Utilisation says the same thing from the other side: your batch volume would occupy about 9% of a single node, so most of what you paid for would sit idle.

Inputs used: nodes for availability 2 · node price per hour 2.75 · hours per month 730 · engineer time, in FTE 0.25 · fully loaded engineer cost 12000 · platform overhead 450 · workloads treated as portable batch-summarisation, retrieval-embeddings · utilisation of one node 9

WHAT WE WOULD DO INSTEAD

Revisit this only when portable spend is above the break-even figure, and only after the retry and route fixes have landed, because those reduce the very spend that self-hosting is supposed to replace.

Also examined, also declined

Declined on purpose. We are reporting zero waste from a difference that looks like a finding.

5. What we do not recommend (continued)

Do not buy a semantic cache for the summarisation pipeline

Verdict.

Declined. The ceiling is a tenth of the vendor fee, and the ceiling is optimistic.

1. Ceiling: \$95 a month, which assumes every reusable document hits the cache at a perfect hit rate.

2. Vendor fee: \$900 a month. Net position: -\$805 a month.

3. The ceiling is small for a good reason. The pipeline already deduplicates its own input, and only 3.6% of documents in the window were repeats. A cache would be paid to rediscover that.

4. Caching also changes invalidation semantics. Someone has to own that, and the ownership has a cost even when the licence does not.

Inputs used: vendor fee 900 · reusable share measured 3.6 · feature batch-summarisation

WHAT WE WOULD DO INSTEAD

Deduplicate upstream where the documents are collected, which costs nothing per month, and re-measure the reusable share in a quarter. If it rises above roughly a third, the calculus changes.

Do not chase off-peak or batch discount windows by delaying work

Verdict.

Declined. We measured the hourly distribution first, and it does not support the recommendation.

1. \$2,640 of spend could in principle move to a discounted window, since it is not interactive.

2. Only 18.1% of it already lands in that window, which caps the achievable reduction at \$239 a month even if everything movable is moved.

3. Earning that would mean delaying summaries by up to 12 hours, which changes when people see their own data.

4. The interactive assistant cannot move at all. It is used during business hours, and that is the point of it.

5. We would not have been able to say any of this without the hourly distribution, which is why we do not make timing recommendations without it.

Inputs used: discount offered 50 · share already inside the window 18.1 · workloads treated as portable batch-summarisation · maximum delay introduced 12

WHAT WE WOULD DO INSTEAD

Leave the nightly window where it is and fix the route table, which is worth more per hour of engineering time than a rescheduling project.

Do not consolidate to a single provider to reach the next volume tier

Verdict.

Declined. The tier is smaller than the redundancy it would cost.

1. Consolidating would move \$3,660 of monthly spend to the largest provider, which reaches a 6% tier: \$220 a month.

2. Against that, the window contains two provider-side degradations, of 41 and 96 minutes, which a second provider absorbed without a customer-visible incident.

3. The same change removes the second provider's independent capacity, its separate rate limits, and the only meaningful exit from the first provider's terms.

4. We can price the tier. We cannot price the outage that arrives on the day the reduction is largest.

Inputs used: tier discount 6 · services that would move s6, s7, s8, s9 · provider degradations, minutes 41, 96

WHAT WE WOULD DO INSTEAD

Keep two providers, and put the consolidation question in front of whoever owns availability rather than whoever owns the invoice.

6. Method appendix

What was read, how it was sampled, what was assumed, and what this document can and cannot support. The reconciliation tables at the end of this section are generated from the same data file as every number above.

Data sources

SOURCE	WHAT WAS READ
Invoice, measurement window	Line-level invoice export from four providers, covering the 30 days ending 18 August 2026.
Invoice, verification window	The following month's export for the same four providers, used only to verify items that were implemented.
Provider usage reports	Metered volumes per endpoint: sessions, documents, retrieval calls and reserved-capacity availability.
Gateway and application logs	Request-level logs with attempt counters, response classes and tool-call fingerprints, for the chatbot and the orchestrator.
Read-only configuration and inventory	Gateway route table, orchestrator retry configuration, and the reserved-capacity and deployment inventory. Read as text; nothing was changed.
Not collected	Prompt bodies, customer content, source code and any production access. None of it was requested or received.

Sampling and attribution

1. Window: 30 days ending 18 August 2026, chosen to match the invoice period exactly so that no spend had to be apportioned across months.
2. Attribution order: invoice line to service to feature to provider. Spend that could not be attributed was capped and reported rather than spread across features.
3. Retry analysis: 100% of orchestrator requests in the window, because attempt counters are cheap to count.
4. Turn-level analysis: a 12% sample of completed turns for duplicate tool calls, weighted back to the full window. A full trace of every turn would have been disproportionate to the amount at stake.
5. Document volumes: seven consecutive days extrapolated to 30 days for the summarisation feature.
6. Verification: the same method re-run against the following invoice, with volumes normalised to the baseline so that a volume change cannot be read as a saving.

Unit price table

Captured **2 October 2026**. Unit prices were captured on the date above and archived with the working papers. Providers change prices; a report without a capture date is not checkable six months later.

6. Method appendix (continued)

Assumptions

ASSUMPTION	VALUE USED	WHY IT MATTERS
Retry attribution	One hundred percent to the owning feature	A repeat attempt is charged to the feature that owns the request identifier and is never split across features.
Volume drift	2.4% between windows	Verified reductions are normalised to baseline volume. Without this, volume growth would silently cancel part of a real reduction.
Document volume	Seven days extrapolated by 30 divided by 7	The largest single uncertainty in this document, and the reason the batch unit cost is stated with more caution than the other two.
Currency	USD as billed	No currency conversion was applied and no list prices were reconstructed.
Discounts	Net amounts as invoiced	The enterprise discount was already applied upstream, so this document cannot be used to reason about list price.
One-off events	Included, not stripped out	Two full index rebuilds sit inside the retrieval line. They recur, so treating them as accidents would flatter the baseline.

Confidence scale

LEVEL	MEANING
High	The amount is visible on the invoice and corroborated by a second independent source, such as a log or an inventory read. Two sources, consistent, no sampling.
Medium	The billing is unambiguous but either the attribution is inferential or a term we cannot see affects how much is recoverable.
Low	Evidence exists but depends on an assumption we could not test. Nothing in this document is rated Low; anything that would have been was left out instead.

Evidence types

- Invoice line: appears on the provider's invoice for the stated period.
- Provider usage report: metered volume from the provider's own reporting.
- Gateway or application log: request-level evidence under our counting rules.
- Read-only configuration or inventory: configuration or capacity read as text, never modified.

What this document can conclude

- Attribute 30 days of invoice spend to a feature, a service and a provider, and reconcile that attribution back to the invoice total.
- Identify routes, endpoints and deployments that are billed but carry no user-visible traffic.
- Count repeat attempts that could not have succeeded and price them at the provider's own rates.
- State a per-unit cost for three features from measured spend and metered volume.
- Say which reductions appeared on the following invoice, normalised for volume.

What it cannot conclude

- We cannot state savings for items that were not implemented. Those appear as identified spend and are never added to the verified figure.
- We cannot say what removing a deprecated route did to answer quality. We can say the endpoint stopped appearing on the invoice; we cannot say that no customer noticed.
- We cannot attribute spend to individual end customers, and we did not attempt it.
- We cannot see pre-discount list prices, so this document cannot support a renegotiation of the enterprise tier from first principles.
- We cannot forecast prices. The capture date above is the boundary of what we know.
- We cannot see the contract behind the reserved endpoint, which is why that finding is capped at Medium confidence rather than High.

6. Method appendix (continued)

Reconciliation

Each table below is arithmetic rather than narrative. The middle column is generated from the same data file that produced every other number in this document, so it can be recomputed independently.

1. Five line items to the monthly total

LINE ITEM	SERVICES	AMOUNT	RECONCILES TO
Support chatbot	1	\$4,180	s1
Batch summarisation	2	\$2,640	s2, s8
Retrieval and embeddings	2	\$1,920	s3, s4
Agent retries (avoidable)	1	\$980	s6
Dead model routes	3	\$1,200	s5, s7, s9
Sum of 5 line items		\$10,920	✓ \$10,920

2. Seven findings to identified waste

FINDING	AMOUNT	VERIFIED	PENDING
F1 Retry storm with no backoff and no cap Agent retries (avoidable)	\$320	\$320	—
F2 Retries issued on responses that could never succeed Agent retries (avoidable)	\$260	\$260	—
F3 Duplicate tool execution, including replay after a broken stream Agent retries (avoidable)	\$400	—	\$400
F4 Deprecated route still receiving live traffic Dead model routes	\$300	\$300	—
F5 Superseded batch route left in the table Dead model routes	\$220	\$220	—
F6 Reserved endpoint with no routed traffic Dead model routes	\$440	—	\$440
F7 Cloned deployment from a cancelled project Dead model routes	\$240	\$240	—
Subtotal: Agent retries (avoidable), 3 findings	\$980	\$580	\$400
Subtotal: Dead model routes, 4 findings	\$1,200	\$760	\$440
Identified waste, 7 findings in 2 mechanisms	\$2,180	\$1,340	\$840

✓ \$1,340 realized and verified plus \$840 still identified equals the \$2,180 identified waste, which is 20.0% of the \$10,920 spent in the window: ✓ \$1,340, 12.3% of spend, and ✓ \$840, 7.7% of spend. Rounding is applied for display only. Reconciliation uses unrounded values, which is why a column of displayed figures can appear to be off by a tenth of a percent.

On redaction. In a real engagement this appendix carries the client's own account identifiers, endpoint names and route table keys. In this worked example every identifier has been redacted to a form such as acct_••••7f2 or prod-chat-v3 (redacted). The redaction is cosmetic: it removes names, not amounts, and the arithmetic is unaffected.

The two mechanisms that mattered here were a retry policy that never read the response class, and a route table that outlived the models in it. Neither needed a rewrite, and neither would have been found by a tool that only reads the invoice. That is the difference between a spend report and an audit.

Lapisli Audit · LAP-SAMPLE-001 · 6 October 2026 · Worked example on synthetic data — the workload, invoices and logs in this document are synthetic; the patterns and mechanisms are drawn from real audits.