FLUREE TOKENOMICS

The Tokenomics of Enterprise AI

Why a data layer beats a bigger context window

A total-cost-of-ownership story for the CDO who has to defend the AI budget to the CFO.

TL;DR · EXECUTIVE SUMMARY

The bill tripled while the price fell 99.7%. The cause is architecture — and so is the fix.

99.7% ↓
price per token, since GPT-3

320% ↑
total enterprise AI spend, 2024→2026

77%
lower cost per user with Fluree

$22M
saved over a 5-yr, 50→500-seat ramp

PROBLEM

Per-token prices collapsed, yet enterprise AI bills exploded — from ~$1.2M (2024) to ~$7M (2026). This is not a pricing problem. The conventional pattern treats the LLM as the data tier, re-loading data into context and re-reading a growing transcript on every step of an agentic loop — a quadratic tax that gets worse the more your best analysts use it.

SOLUTION

Put a governed knowledge graph with write-back memory between the agents and the data. Query, don't load. Schema on tap. Write-back memory. Build once, share. The data layer holds the long-term, growing context — ready for the LLM — so each question stays small and flat instead of re-paying for state on every turn.

IMPACT

~22× fewer tokens for the same questions and answers. 77% lower cost per user. ~$170 saved per seat every month. And roughly $22M in cumulative savings over a five-year ramp from 50 to 500 seats — ~$2.4M even if today's subsidized prices never move — delivered as a predictable line, not a curve that bends toward the ceiling.

BOTTOM LINE

This is not a cheaper way to do the same thing. It is the only architecture whose bill stays predictable as you scale — because cost tracks seats and queries, not the unbounded growth of accumulated context multiplied by an unknowable future token price. Provenance and a single source of truth come along as a side effect.

IN THIS PAPER

The bill nobody budgeted for

Here is the paradox sitting at the center of every 2026 AI budget meeting: the price of intelligence has collapsed, and the cost of using it has tripled.

PRICE PER TOKEN

−99.7%

since the GPT-3 era — and still falling

JEVONS PARADOX

TOTAL ENTERPRISE AI SPEND

+320%

~$1.2M (2024) → ~$7M (2026) per year

Per-token prices have fallen roughly 99.7% since the GPT-3 era,[1] and they keep falling. By any reasonable reading of the price sheet, enterprise AI should be getting dramatically cheaper. Instead, total enterprise AI spend has risen by an estimated 320% — average annual budgets ballooning from about $1.2M in 2024 to $7M in 2026, according to multiple industry analyses.[2] The FinOps Foundation's executive director described the mood shift bluntly: teams that were "tokenmaxxing" in early 2025 spent the spring of 2026 discovering they were 3x over their *entire annual* token budget by April.

This is not a pricing problem. It is an architecture problem. And it is the single most important thing a Chief Data Officer can explain to a CFO before the next renewal cycle.

The mechanism has a name economists will recognize — the **Jevons paradox**: when a resource gets cheaper, you consume so much more of it that total spend rises anyway. Cheaper tokens didn't save money.[3] They made it economical to point AI at every workflow in the building, and *the way* most enterprises wired those workflows quietly multiplies token consumption by one to three orders of magnitude per task.

The good news: the multiplier is a function of architecture, not destiny. Change where the data lives and the curve bends. This paper walks through exactly why agentic AI bills compound, why "just use a bigger context window" is the most expensive default in enterprise software, and what the total cost of ownership looks like when you put a governed data layer — a knowledge graph with memory — between your agents and your data.

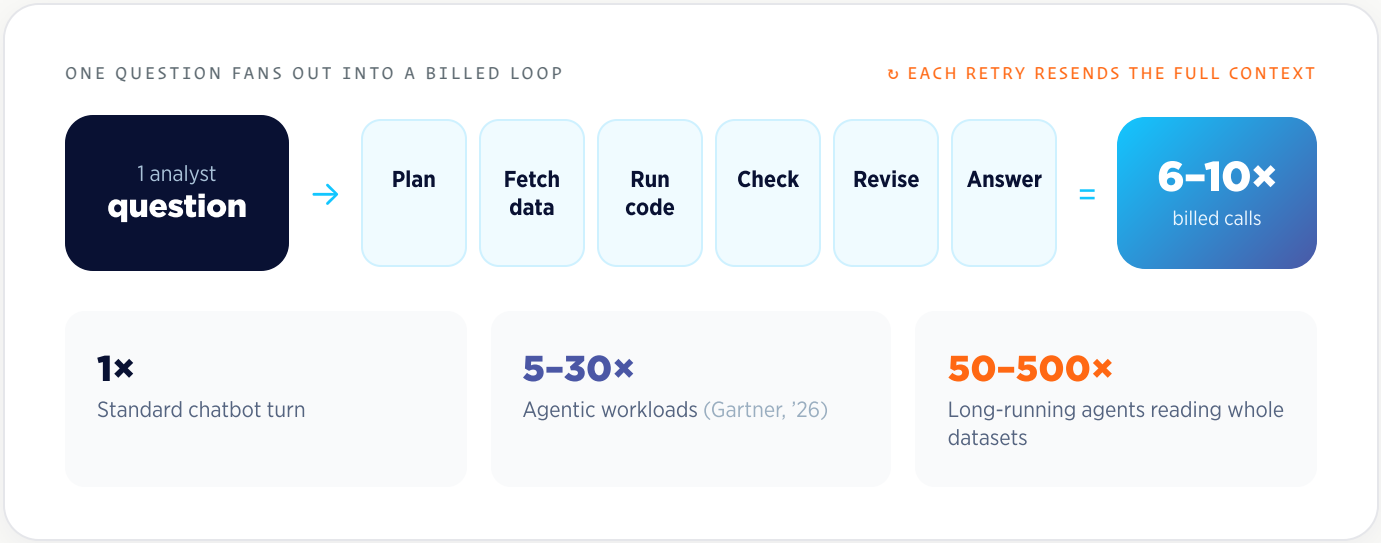
SOURCES1 Navya AI, "AI Cost Report: Token Prices vs. the AI Bill" (2026) · 2 The Next Web, on FinOps Foundation commentary (2026) · 3 AuthorityTech (2026) — full citations on p. 14

HOW THE BILL COMPOUNDS

Why one question is no longer one answer

Start with the unit of work that actually generates the bill. In a 2024 chatbot, a question was a single model call: prompt in, answer out, one invoice line. That mental model is what nearly every enterprise AI business case was built on. It is also obsolete.

In an agentic system, **a single analyst question fans out into 6–10 internal model calls** — plan, fetch data, run code, check the result, revise, answer. You pay for every step in that loop, not just the reply the user sees. Gartner's March 2026 analysis[4] puts agentic workloads at 5–30x the tokens of a standard chatbot turn; at the heavy end, long-running agents that read entire datasets land in the 50–500x range.[5] EY's "Total Cost of Agents" series[6] quantifies the same shift in dollar terms: roughly **30 times more** per interaction in 2026 than in 2023, for the same business question.



A simple linear workflow that cost about **$0.04** per interaction in 2023 costs roughly **$1.20** in 2026 — about 30 times more for the same business question. (EY, "Total Cost of Agents")[6]

~30×
$0.04 → $1.20 / interaction

SOURCES4 Gartner analysis (Mar 2026), via Oplexa · 5 Fortune, Jun 25, 2026 · 6 EY, "The Total Cost of Agents" — full citations on p. 14

THE COMPOUNDING MECHANISM

Three forces, stacking multiplicatively

Three forces drive the compounding, and they stack multiplicatively:

01

Agentic loops

The plan-fetch-code-check-answer cycle means one user prompt triggers many billed inferences. Each retry to fix a bad output resends the full working context. An agent running ten correction cycles can burn 50x the tokens of a single clean pass — and retries are architecturally necessary for quality, so you can't simply switch them off.

02

Data pulled into the prompt

To analyze enterprise data — ERP rows, PLM records, a CRM export — the conventional pattern loads the *raw rows* into the model's context. Then the agentic loop re-reads those same rows on every step. The data isn't analyzed once; it's re-attended five, eight, ten times per question.

03

A transcript that only grows

Every new turn re-sends the entire conversation so far. The model has no memory between calls — it must be told everything, every time. Context grows with each message, and because the model re-reads the whole thread each turn, **cost grows with roughly the square of the conversation length.**

72% of production AI cost sits *outside* the visible model invoice — in orchestration, retrieval, retries, and re-attended context.[1] The number on the vendor's pricing page didn't change. The architecture around it did all the damage.

THE FAILURE MODE

This is the failure mode Fluree calls **context-in-the-loop**: agents fail not because they lack intelligence, but because the architecture forces them to re-acquire context from scratch on every step.

SOURCES1 Navya AI, "AI Cost Report" (2026) · "Context-in-the-loop" is detailed at flur.ee/blog — full citations on p. 14

The expensive default: “just give it more context”

When teams hit a quality wall, the reflex is to feed the model more — more rows, more documents, a bigger context window, more of the transcript retained. This feels like the cheap fix because the marginal price per token is tiny. It is, in fact, the most expensive decision in the stack, for two compounding reasons.

1 REPRICING EXPOSURE

First, **long context and caching are precisely the patterns most exposed to repricing.** The attention cost that makes long context expensive scales roughly quadratically with context length. Today’s token prices are widely understood to be subsidized below cost[4] — venture capital and hyperscaler cross-subsidies are footing part of the bill, and “normalization upward is a *when*, not an *if*.”[4] When that subsidy rolls off, the inefficient pattern — long context, heavy caching — is repriced hardest. You are not just paying more tokens today; you are accumulating exposure to tomorrow’s correction.

2 COST-OF-PASS

Second, **the bigger window doesn’t even buy you better answers past a point.** Stanford researchers introduced a metric called *Cost-of-Pass*[7] — the expected monetary cost of generating a *correct* solution, not just the cost of generating tokens. Their finding cuts directly against the reflex: common inference-time techniques that throw more compute at the problem typically *raise* the cost-of-pass rather than lower it. Once you measure cost per correct answer rather than cost per token, “stuff more into the window” stops looking like optimization and starts looking like what it is: paying a quadratic tax to re-read data the model has already seen.

THE DEEPER POINT FOR A DATA LEADER

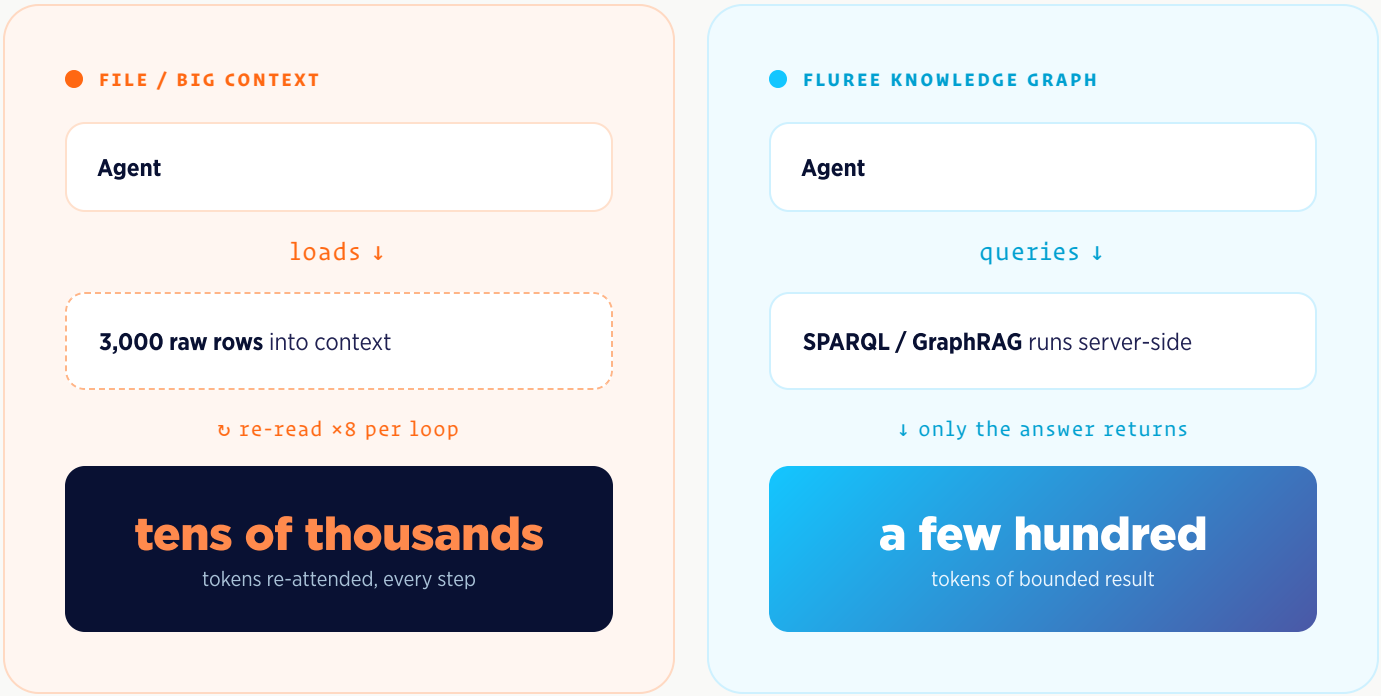
Shoving context into the model treats the LLM as your data tier. Every question re-loads, re-reads, and re-pays for state that should live somewhere persistent, governed, and queryable.

You would never run analytics by re-streaming your entire warehouse through the application layer on every query. Yet that is exactly the shape of file-based and “big context” agentic AI.

SOURCES4 Oplexa, “AI Inference Cost Crisis” (2026) · 7 Erol et al. (Stanford), “Cost-of-Pass,” arXiv:2504.13359 — full citations on p. 14

Fluree's answer: keep the data out of the prompt, keep each question small

The alternative is structurally simple to state and architecturally deep to execute: **put a governed knowledge graph between the agents and the data, and let agents both read from it and write back to it.** The data layer holds the long-term, growing context — ready for the LLM — instead of the LLM holding it, badly, one expensive turn at a time.



Four mechanisms do the work.

01 **Query, don't load**

02 **Schema on tap**

03 **Write-back memory**

04 **Build once, share**

FOUR MECHANISMS

How the data layer breaks the curve

Query, don't load

Agents run SPARQL or GraphRAG queries **server-side** and get back only the answer rows. The raw data never enters the model's context. Instead of loading 3,000 rows and re-reading them eight times, the agent asks a precise question and receives a bounded result — a few hundred tokens of answer, not tens of thousands of tokens of raw input re-attended on every loop.

Schema on tap

A compressed, searchable ontology — Fluree's semantic layer — lets the agent understand the *data model* without reading all of the data. It knows what's queryable and how the entities relate, then writes a precise query rather than ingesting everything to figure out what's there.

Write-back memory

BREAKS THE SQUARE

This is the part that breaks the quadratic curve. Agents save findings, validated queries, and guidance to a **memory graph** — agentic memory that lives in the graph, not the prompt. The next question retrieves a compact summary instead of replaying the whole transcript — context resets to near-zero at each new question rather than growing without bound. The conversation stops paying the square.

Build once, share

An analysis is saved as a governed artifact and reused across teams, plants, and regions — not rebuilt, and re-paid for, by every user who needs the same answer. This is the **AI Agent Mesh** pattern. Contrast the file-based alternative: a copilot-style app built on a local extract often can't even be shared without IT re-platforming it, so its build cost never amortizes across the team. A governed graph artifact does.

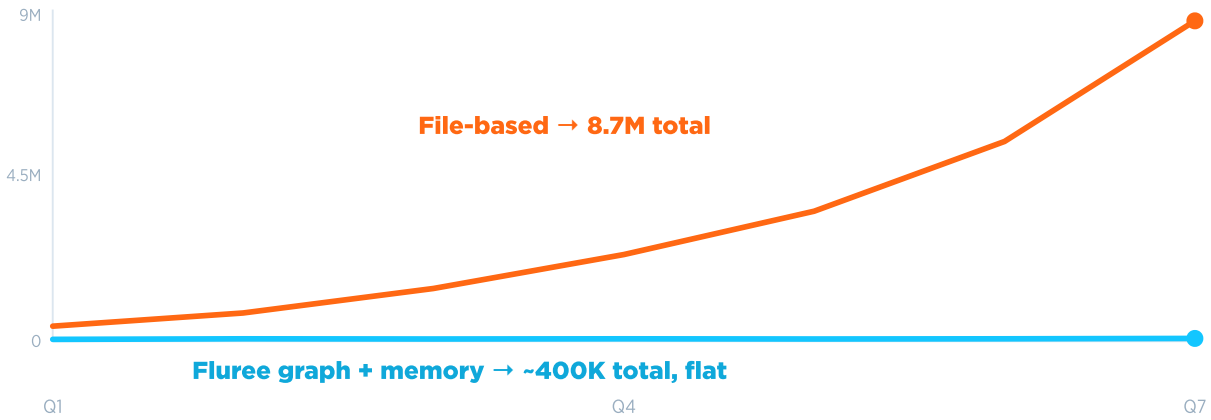
THE NET EFFECT

Same questions, same answers, ~22× fewer tokens

The net effect, modeled across a realistic seven-question analyst session: the file-based approach climbs from 0.4M tokens at question one to 2.1M by question seven as the transcript accumulates — **8.7M tokens over the session.** The graph-plus-memory approach stays flat at roughly **57K tokens per question regardless of session length** — about 400,000 total.

CUMULATIVE TOKENS ACROSS A 7-QUESTION ANALYST SESSION

per question Q1–Q7



~22×

That's **~22x fewer tokens for the same questions and the same answers.** Reasoning and output barely move between the two; the input context is the entire story. One architecture re-reads a growing thread; the other resets it. And why do ~22× fewer tokens become 77% — not 95% — fewer dollars? Because the file pattern's re-read input is billed at discounted cache rates, while output tokens — priced ~5× input — barely differ between the two. The dollars compress the gap; the repricing exposure doesn't.

0.4M → 2.1M
File-based tokens per question, Q1 → Q7

~57K, flat
Fluree tokens per question, at any session length

8.7M vs 0.4M
Session totals for the same seven answers

NOTE Token counts are modeled estimates of a seven-question analyst session, not customer telemetry — assumptions on p. 14

What it means per user, per month

Token ratios are interesting. Dollars are persuasive. So here is the same per-question behavior, priced at today's rates and modeled across three honest usage tiers — because a procurement reviewer who logs in twice a week and a heavy agentic investigator who builds reusable apps do not consume alike.

ANALYST ARCHETYPE	FLUREE / MO	FILE-BASED / MO	SAVED / MO	SAVED %
Light occasional reviewer	**$18**	**$61**	**$43**	**70%**
Core the typical daily analyst	**$71**	**$289**	**$217**	**75%**
Power heavy agentic, builds apps	**$195**	**$983**	**$787**	**80%**
Blended across the mix	**$52**	**$222**	**$170**	**77%**

The headline for the typical user is the one to put on the slide: **a Core analyst costs $289/month on the file-based approach and $71 on Fluree** — the same questions, the same answers, only the re-reading differs. Blended across the mix, that's **$170 saved per user every month, a 77% reduction.**

Notice the pattern in the "Saved %" column: the savings rate *rises* with intensity, from 70% for light users to 80% for power users. This is the opposite of how most enterprise software scales. Heavier use of a file-based system means more loops, more re-reads, more accumulation — so the worst offenders are your most valuable analysts. The data-layer architecture turns that on its head: the more agentic the work, the bigger the structural advantage.

NOTE Priced at published mid-2026 API rates; usage tiers are modeled — assumptions on p. 14 · "File-based" prices a copilot-style, load-files-into-context usage pattern — not any vendor's product or seat pricing · columns rounded independently

THE FIVE-YEAR STORY

The value compounds at scale — twice

A single user's $170/month is a clean number. But CDOs aren't deploying to one user, and CFOs aren't budgeting for one year. The TCO story is a five-year story, and over that horizon **two forces push in the same direction.**

FORCE ONE · MORE USERS

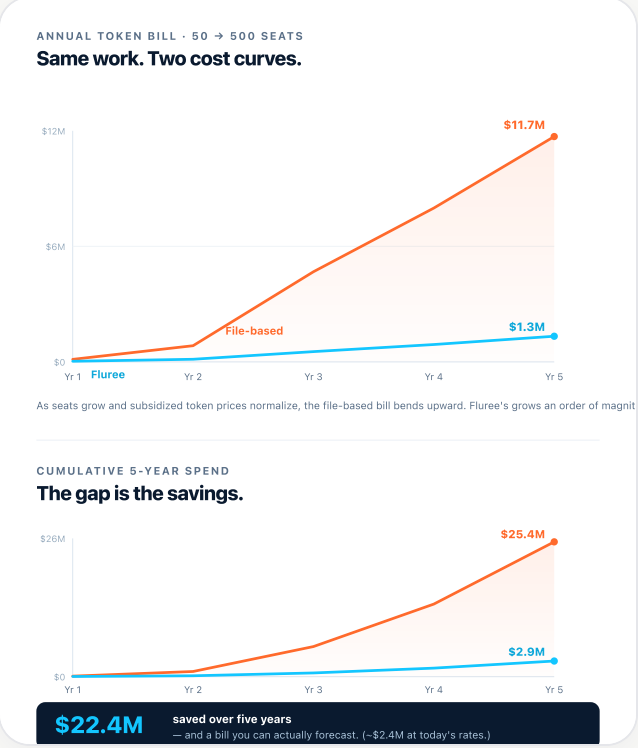
A 50-seat pilot becomes 500 seats. The per-seat gap doesn't shrink with scale — it scales *with* the seats, linearly, every one of them.

FORCE TWO · TIGHTER TOKEN ECONOMICS

Today's prices are subsidized. As they normalize upward[4] — and as cached-token pricing rises from ~10% of the input rate toward 25–50% — the *inefficient* pattern is repriced hardest. Seat growth and repricing compound together rather than cancelling out.

Modeled across a five-year ramp from 50 to 500 seats, with the subsidy rolling off over years three through five:

	YR 1	YR 3	YR 5	5-YR CUM.
Seats	50	200	500	—
File-based	$133K	$4.68M	$11.70M	$25.4M
Fluree	$31K	$0.53M	$1.33M	$2.94M
Saved / yr	$102K	$4.15M	$10.37M	$22.4M



SOURCES4 Oplexa (2026) · dollar figures are modeled estimates — assumptions on p. 14 · at today's rates held flat, the same ramp still saves ~$2.4M (77%)

By year five, the file-based architecture is running an $11.7M annual token bill; the data-layer architecture runs $1.3M for the same work. **Cumulative five-year savings land around $22M under the repricing scenario — ~$2.4M with today's rates held flat** — and, just as importantly for the person who has to forecast it, the Fluree line is a gentle slope while the file-based line bends sharply upward as repricing bites.

That last point is the one CFOs actually buy. It isn't only that the bill is smaller. It's that the bill is **predictable.**

An architecture that keeps each question's context small is the only one whose cost you can forecast as you scale, because cost tracks seats and queries — not the unbounded growth of accumulated context multiplied by an unknowable future token price.

BEYOND THE TOKEN LINE

The part that doesn't show up in the API invoice

Token math wins the cost argument. But a CDO selling this internally has a second, equally important story — the one about *trust* — and it happens to reinforce the economics.

A governed knowledge graph isn't just cheaper to query; it changes what an answer *is*. Because every fact carries its lineage, answers become provable, reproducible, and cited every time. The data is combined, organized, and governed into one source of truth — golden records the whole organization shares — rather than scattered across files that three different AI tools will summarize three different ways.

WHAT CHANGES UNDERNEATH THEM

One governed answer, every single time

This is what the people closest to the work describe when the architecture changes underneath them — illustrative composites drawn from data-leader conversations, not customer quotes.

CHIEF DATA OFFICER

Previously couldn't prove where a number came from — can now trace every figure back to its source.

VP OF FINANCE

Spent a week verifying each board number — now trusts each figure because it carries its own receipt.

HEAD OF DATA

Used to get three answers from three AI tools — now gets one governed answer, every single time.

SVP OF SALES

Catches every at-risk account the moment the data shifts, rather than learning of it days late.

Those are not soft benefits floating free of the cost case — they're the same architecture viewed from the business side. The reason the answer is cheap (query the graph, don't re-load the data) is the same reason it's trustworthy (the graph is governed, versioned, and carries provenance). Cost discipline and auditability come from one design decision, not two.

THE BOARD HAS MOVED ON FROM TOKEN CHARTS

The 2026 board wants **efficiency ratios**[4,6] — cost per resolved ticket, cost per correct answer, revenue per workflow against the inference it consumed. A data layer that returns bounded, governed, reusable answers is the architecture that makes those ratios *good*, because it minimizes the denominator (tokens spent) while maximizing the numerator (answers you can actually act on and defend).

SOURCES4 Oplexa (2026) · 6 EY, "The Total Cost of Agents" (2026) — full citations on p. 14

A NOTE ON INTELLECTUAL HONESTY

Name the assumptions. The direction holds.

A credible TCO story names its own assumptions, so here are the load-bearing ones.

The token counts in the model are *modeled estimates* of these flows, calibrated against a realistic analytic usage pattern — not telemetry from a specific customer. The exact dollar figures move with your model tier, your seat mix, and how aggressively token prices normalize. If your alternative already uses a tuned retrieval index rather than dumping raw extracts into context, the file-based numbers come down; if your users dump whole spreadsheets into the chat, they go up.

THE DURABLE INSIGHT IS ARCHITECTURAL

Retrieve versus re-read.

But the *direction* is not in dispute, and it doesn't depend on any single cell in the spreadsheet. Any pattern that re-loads data into the model and re-reads a growing transcript pays a multiplicative tax that compounds with usage and with time. Any pattern that queries a governed data layer server-side and resets context per question does not.

EVERY EXTERNAL DATA POINT POINTS THE SAME WAY

99.7% ↓[1,2]
price collapse vs 3x bill increase

5–500×[4,5]
agentic token multiplier

Subsidy[4]
roll-off thesis on token prices

Ratios[6]
board-level shift to efficiency

THE LOAD-BEARING ASSUMPTIONS

Usage mix — 60/30/10 light/core/power analysts, ramping 50→500 seats over five years.

Repricing scenario (Yrs 3–5) — input ×3, output ×4, cache-read fraction →50%. Held at today's rates instead, five-year savings are ~$2.4M (77%) rather than ~$22M — the direction is unchanged.

Prices — published mid-2026 frontier API list rates (≈$5/M input, ≈$25/M output; cache reads ≈10% of the input rate).

Deliberately conservative — cache writes are modeled as free for both architectures (real caches charge ~1.25× to write); the file-based pattern caches ~36× more tokens, so the simplification flatters the alternative, not Fluree.

REFERENCES

1 **Navya AI** — "AI Cost Report: Token Prices vs. the AI Bill" (2026) · navyaai.com

2 **The Next Web** — "Token prices fell 98%, enterprise AI bills tripled" (2026), citing the FinOps Foundation · thenextweb.com

3 **AuthorityTech** — "How Inference Economics Changed Enterprise AI Buying" (2026) · authoritytech.io

4 **Oplexa** — "The AI Inference Cost Crisis of 2026," citing Gartner's March 2026 analysis · oplexa.com

5 **Fortune** — report on GPU & token economics, June 25, 2026 · fortune.com

6 **EY** — "The Total Cost of Agents" insight series (2026) · ey.com

7 **Erol, El, Suzgun, Yuksekgonul & Zou** (Stanford) — "Cost-of-Pass: An Economic Framework for Evaluating Language Models" (2025) · arXiv:2504.13359

Fluree figures on pp. 07–12 are modeled estimates at published mid-2026 API rates — not customer telemetry.

THE TAKEAWAY

For the CDO who has to defend the number

The question your CFO is going to ask is not "is AI worth it." It's "why did the bill triple when the price fell 99.7%, and what makes you confident it won't happen again next year."

The answer is that the bill tripled because the conventional architecture treats the LLM as the data tier — re-loading enterprise data into the model's context and re-reading an ever-growing transcript on every step of an agentic loop, paying a quadratic tax that gets *worse* the more your best analysts use it and *worse again* as subsidized token prices normalize upward.

A governed knowledge graph with write-back memory removes the very factor driving the sticker shock. It keeps the data out of the prompt, keeps each question's context small and flat, and turns analyses into governed artifacts the whole organization reuses instead of rebuilding.

77%
lower cost per user

~$170
saved per seat per month

~$22M
cumulative 5-yr savings · ~$2.4M at today's rates

That's not a cheaper way to do the same thing. It's the only architecture whose bill stays predictable as you scale — and the one that hands you provenance and a single source of truth as a side effect of getting the economics right.

GO DEEPER

Context-in-the-Loop: the architecture for autonomous AI →

Inside FlureeDB: how the knowledge graph stays verifiable →

Semantic GraphRAG: the complete guide to retrieval, knowledge graphs & LLMs →

Start free

Talk to our team

Pressure-test these numbers against your own seat mix and usage pattern — flur.ee/solo · flur.ee/contact