Persistent

# Tokenmaxxing: The Great Token Heist

Your token dashboard is measuring waste and calling it adoption

Section 01

# Metric Everyone Trusts is a Lie

Walk into any enterprise AI steering committee in 2026 and the same slide appears: tokens consumed, up and to the right. Token burn has quietly become the industry's default proxy for AI adoption, a proof point that employees are "using" AI, evidence that the transformation is landing and ammunition for next year's budget. Vendors encourage it, boards applaud it and CIOs get promoted on it.

That number is a smokescreen. A rising token count cannot distinguish between an agent producing a decision and an agent re-transmitting the same tool results five times. It cannot tell whether a workflow succeeded or merely ran. Tokens measure consumption, not contribution, and celebrating token growth is celebrating CPU cycles. A CFO would never report "our software burned 40% more compute this quarter" as a win. Yet that is precisely what every AI adoption dashboard does.

Worse: much of that burn is not even user demand. It is forced consumption, manufactured by the architecture itself. An agent re-sends its entire conversation history on every turn, rediscovers a database schema that hasn't changed in months, transmits pretty-printed JSON whose whitespace alone costs money and breaks its provider's cache with careless prompt structure, while paying full price for repetition. None of this is an employee getting value. The system is generating its own demand and the dashboard scores it as adoption.

How do AI leaders decide which tokens bought an outcome, and which bought motion? There is no straight answer.

Section 02

## What Got Us Here: We Solved the Wrong Problem First

The first era of enterprise AI organized itself around model accuracy. It gave us retrieval for grounding, review loops, citation requirements, governance boards, in other words, the entire Responsible AI discipline. But every fix for reliability made the economics worse: every grounding layer increased what the model must process, every retrieval call added latency, every reasoning trace carried forward more history. We learned to make AI reliable; we never learned - with equal urgency - to make it economical.

> The systems struggling in production today are not failing because they hallucinate. They are failing because they work expensively. The first enterprise AI problem was probability. The next one is profitability.

Section 03

## Rise of Tokenmaxxing

Tokenmaxxing is the predictable consequence of treating context as free and of measuring usage instead of value.

In traditional software, inefficiency is visible in dashboards and alerts. In agentic AI, it hides inside the interaction. A user asks one question and receives one answer; beneath it sits a chain of prompts, retrievals, tool calls, retries and most insidiously, context retransmissions. An agent does not merely answer; it investigates, plans, retrieves, observes and revises, and in each loop the cost of every prior loop travels forward. Turn 5 re-pays for everything Turn 2 already processed.

That is tokenmaxxing; not more intelligence, but more consumption masquerading as intelligence.

This is why agents represent an economic regime change. A chatbot's cost is bounded by one prompt and one response; an agent pursues an outcome, so its cost is bounded only by workflow depth. The danger is not that agents fail, it is that they succeed inefficiently by spending money on their own path to an answer while the meter reads "adoption."

Section 04

# We Built a System that Fights Tokenmaxxing

This is not a thought experiment. We built and instrumented a Context Graph architecture for financial decision intelligence, which is a system preserving decision history, causality and precedent so that when an agent is asked "Should we approve this $25,000 credit limit increase?" for a customer, it can see the identical request denied three months ago, the fraud flag raised last week and the precedents that anchor sound judgment.

The value is real. In side-by-side testing, a customer with a spotless risk score gets a 95% approval inclination from a naive agent and 45% from the Context Graph agent, which sees the prior escalations, cites decision IDs and quotes risk factors verbatim.

> The Context Graph agent reasons like an institution instead of a stranger.

But Context Graphs are also the first real stress test of AI economics, because they are among the most context-intensive architectures in the enterprise. Our unoptimized baseline, for a three-query session about one customer took:

**9**
LLM turns and 14 tool calls

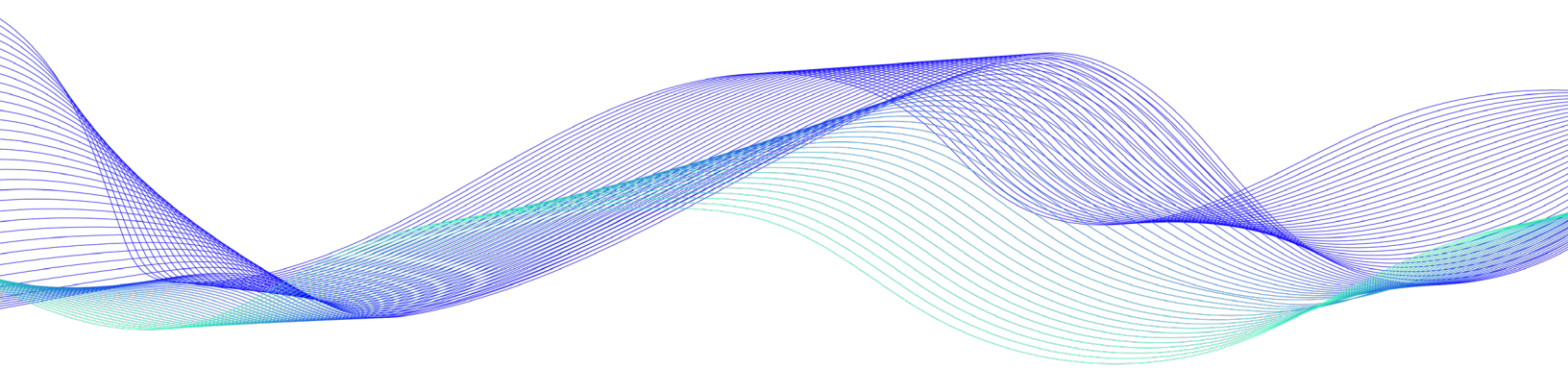
**103,102 Total tokens**
Total tokens most of them re-transmitted data the model had already read

**85 Seconds**
of wall-clock time

Trace it turn by turn and the pathology is obvious: Turn 1 costs ~700 tokens; by Turn 6 the agent is re-sending ~40,000 tokens per turn — the system prompt, every prior tool result, the full history — to answer a question it has effectively already answered. It pays for the same data five times.

The paradox is that the same context that makes AI useful is what makes it economically unsustainable.

> The future of AI is a memory problem. The future of AI economics is a memory management problem.



Section 05

# Blind Spot: Safety Guardrails Without Economic Guardrails

Enterprises have matured the language for Responsible AI — fairness, risk, compliance, trust. Ask the same organizations how many tokens a workflow consumed, which steps drove the cost, how much context was reused versus rediscovered or what the cost per successful outcome was — and the room goes quiet.

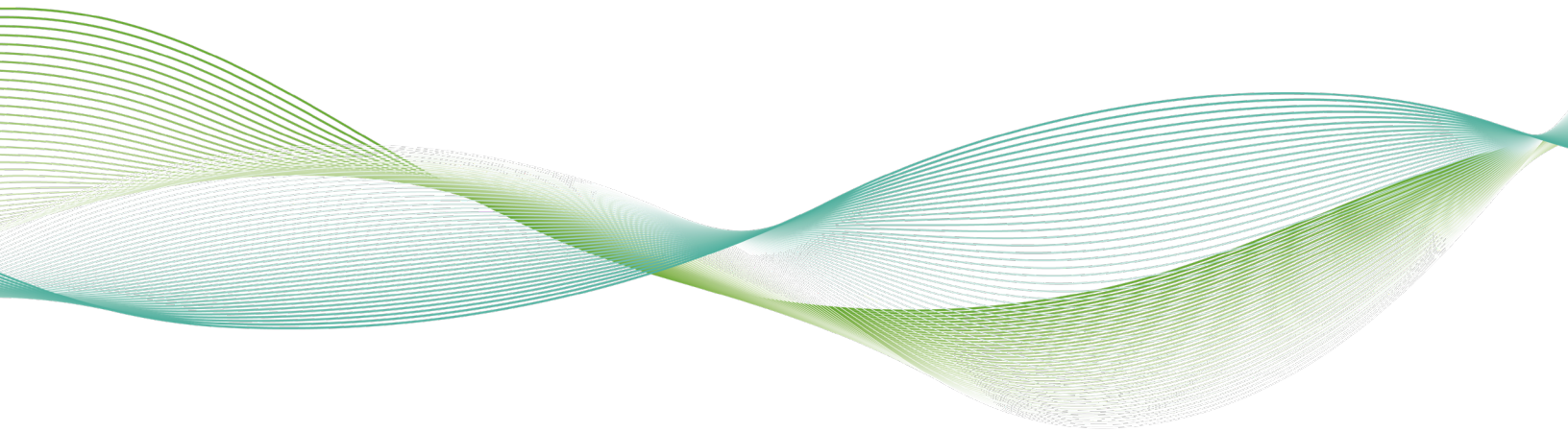
> There is no token governance, no context governance, no cost-per-outcome discipline.
>
> We built safety guardrails and forgot to build economic ones.

We call the missing discipline AI Hygiene: the architectural, operational and governance practices that ensure AI systems consume only the intelligence required to produce a business outcome. Responsible AI asks whether the system is safe, fair and trustworthy. AI Hygiene asks whether it is efficient, economical and disciplined. It does not replace Responsible AI; it completes it.

AI Hygiene begins by replacing the vanity metric with a real one: a token P&L. Instrument every workflow, tag every token to the outcome it served - be it a decision rendered, a case resolved, an exception handled - and report effective cost per successful outcome alongside accuracy. Only then can leaders separate the two populations hiding inside every token dashboard: tokens that bought judgment and tokens that bought retransmission, rediscovery and whitespace.

In our baseline, the majority of a 103K-token session was re-sent data the model had already read. That is not adoption. That is a leak.

And the central insight of AI Hygiene is, in itself, contrarian: agent costs are not fixed by the model. They are driven by architecture, comprising the tool calls the agent makes, the data it re-transmits per turn, the prompt structure and whether the provider can cache repeated prefixes. These are engineering choices. And engineering choices compound.



Section 06

# Five-Layer Optimization Framework

We validated a five-layer framework on the Context Graph system described above. The result was a 45% lower effective token cost, 57% lower latency, 71% fewer redundant tool calls — without changing the model and without degrading response quality.

Each layer targets a different source of waste. Each is independent and composable.

## 1 Hybrid Retrieval — retrieve precisely, not repeatedly.

Most systems retrieve too much because they retrieve imprecisely. Semantic search finds textual similarity, but enterprise decisions often turn on structural similarity — cases described in different language that share the same topology of accounts, relationships and risk.

We combine semantic retrieval (ChromaDB text embeddings) with structural retrieval (Neo4j FastRP graph embeddings) in a single scored pass, eliminating the retry turns burned when a semantic search misses a precedent. Heavy text embeddings live in ChromaDB, never in graph query results, preventing* thousands of tokens of vector bloat per node.

## 2 Intelligent Tool Caching — if the answer hasn't changed, the cost shouldn't recur.

Agents constantly rediscover static information: schemas, policy structures, customer context, embeddings. We cache at four levels — a TTL server-side schema cache, a session-level client-side context store, an LRU embedding cache and cache-aware tools that resolve instantly even when the LLM decides to call them. Recurring computation becomes reusable knowledge.

## 3 Context-Aware Prefetching — stop paying the agent to discover what you already know.

The highest-impact layer and the most counterintuitive. Instead of letting the agent spend three turns discovering the customer's profile, history and fraud signals through tool calls, we front-load that data into the system prompt via fast parallel REST calls before the conversation begins and instruct the agent not to re-fetch it. Tool calls fell from 14 to 4; LLM turns from 9 to 5; latency by 57%.

Here is the part skeptics should sit with: total input tokens barely moved (+1.8%), because the unoptimized agent transmits the same data anyway — through tool results that accumulate and get re-sent every turn. Prefetching doesn't reduce what the model reads. It changes where the tokens come from. That difference is what unlocks Layer 5.

## 4 Compact Serialization — whitespace is pure waste.

Pretty-printed JSON is a subsidy paid to no one. Compact serialization, stripped embedding vectors, truncated long strings and capped result sets reduced token counts by 15–25% across tool outputs. Unglamorous. Free. Compounding.

## 5 Provider Prompt Caching — make the provider pay for repetition.

Providers cache the longest-matching token prefix across calls, and most teams accidentally destroy it by inserting variable customer data mid-prompt, thereby breaking the cache and re-processing the static guidelines that follow at full price on every request.

The fix is prompt architecture: static first, variable last. Instructions and scoring rubrics form a stable ~6K-token prefix; customer data goes at the end. For multi-turn conversations about the same customer, the entire ~19K system prompt becomes cache-identical. In our benchmark, 93% of input tokens were cache hits, cutting the effective cost 46.6%, with zero changes to the LLM call itself.

The fix is prompt architecture: static first, variable last. Instructions and scoring rubrics form a stable ~6K-token prefix; customer data goes at the end. For multi-turn conversations about the same customer, the entire ~19K system prompt becomes cache-identical. In our benchmark, 93% of input tokens were cache hits, cutting the effective cost 46.6%, with zero changes to the LLM call itself.

Section 07

# Beyond Systems We Own: TokenOps for External Agent Harnesses

Everything above assumes an architecture you control. But a growing share of enterprise token spend now runs through harnesses you don't build and can't instrument by editing source code — Claude Code, GitHub Copilot and similar coding and productivity agents that developers invoke directly. This is where AI Hygiene meets TokenOps: the same discipline, applied to a system whose internals are a black box.

The same four failure categories that break traditional FinOps for owned systems show up here, just wearing a different costume.

- **Consumption mechanics go opaque:** a harness silently routes a task to a larger model or lets an autonomous loop run 40 tool calls when 8 would do, and the invoice arrives with no line item explaining why.
- **Architecture and design hygiene erodes by default:** a bloated CLAUDE.md or copilot-instructions.md gets dumped into every single turn, whole-repo context gets stuffed in instead of task-scoped retrieval, and subagent delegation — genuinely useful for parallelizing work — quietly amplifies cost 5–20× per task if nothing bounds its depth.
- **Observability is close to zero:** most engineering teams cannot tell the cost per pull request, cost per ticket or which developer's sessions are burning the budget.
- **Operational waste** compounds silently through retry loops on a failing test, orphaned long-running sessions, redundant file reads across turns that a human would never notice because the harness's UI shows one clean transcript.

Because we cannot rewrite these harnesses, our approach here is not to touch the model call. It's to govern the harness from the outside, in two layers.

- **Governance layer:** Scoped, repo-level instruction files that replace sprawling context with task-relevant policy; explicit tool and MCP-server allowlists so an agent can't reach for expensive capability it doesn't need; hard context budgets — maximum file reads, maximum subagent depth, maximum turns before human checkpoint; and prompt templates inside those instruction files that follow the same static-first, variable-last discipline as Layer 5, so the harness's own provider-side caching actually engages instead of resetting every session.
- **Telemetry gateway:** A proxy sitting between the harness and the model API that captures what the harness itself won't show, such as tokens per session, per developer, per repository; cache hit rate; tool-call counts; and anomaly flags when a session's tool-call count or token burn crosses a threshold with no resolved outcome. Every captured session gets tagged to the ticket or task it served, so the same token-P&L discipline, including cost per successful outcome, extends to coding agents, not just backend systems.

This is precisely the **Assess & Inventory → Foundation Build → Optimization progression:** inventory what these harnesses are actually costing before governing them, stand up the gateway and instruction layer, then ship the routing, caching and waste fixes once the data shows where they're needed.

AI Hygiene is not a property of the systems you build. It is a discipline you can — and must — impose on the ones you merely use.

Section 08

## Combined Result

Measured on the same model, database and three-query scenario:

| Metric | Baseline | Optimized | Change |
|---|---|---|---|
| LLM turns | 9 | 5 | −44% |
| Tool calls | 14 | 4 | −71% |
| Cache hit rate | 0% | 93% | - |
| Effective cost tokens | 103,102 | 56,805 | −45% |
| Wall-clock time | 85s | 36s | −57% |

One query — a fraud pattern analysis — cut cost 61% and answered in a single turn with zero tool calls: the data was already prefetched and 99% of input tokens were cached. The optimized agent also produced slightly longer, more detailed answers (+2.7% output tokens), because it started every turn with richer context. Efficiency and quality were not traded off. They rose together.

Honesty demands the caveats: prefetching pays off in entity-centric, multi-query sessions where the data fits in context; it hurts exploratory or single-shot interactions. Cache-friendly design requires discipline — deterministic ordering, long stable prefixes, no casual prompt reshuffling. This is architecture, not a toggle.

Section 09

## The Vision: Stop Counting Tokens. Start Pricing Outcomes.

The industry's reflex when AI underperforms economically is to wait for a cheaper model. That reflex is wrong: we cut effective cost nearly in half and latency by more than half on the same model, through architectural discipline any engineering team can implement layer by layer.

> Notice what the framework did to the metrics themselves. Raw token volume barely changed (+1.8%) — the adoption dashboard would call the optimized system a flat quarter. Yet effective cost fell 45%, latency fell 57% and answers got better.

The vanity metric and the value metric moved in opposite directions. That single fact should end token-count-as-adoption.

The organizations that win the agentic era will treat context as a managed resource, memory as an engineered asset and cost-per-successful-outcome as the number that goes to the board — governing an agent's economic behavior as rigorously as its ethical behavior. Tokenmaxxing is not inevitable. It is a design failure and design failures have design solutions. Five layers. No model change. 45% of the cost, gone.

The stack is the strategy.

# Author


## Sadashiva Borkar

Senior Data Scientest, CTO AI Research


Sadashiva Borkar is an integral part of the CTO AI Research team. He is currently focused on uncovering business value through the application of Knowledge Graphs, Machine Learning and Generative AI princiles.



### About Persistent

Persistent Systems (BSE: 533179 and NSE: PERSISTENT) is a global services and solutions company delivering AI-led, platform-driven Digital Engineering and Enterprise Modernization to businesses across industries. With over 27,500 employees located in 21 countries, the Company is committed to innovation and client success. Persistent offers a comprehensive suite of services, including software engineering, product development, data and analytics, CX transformation, cloud computing, and intelligent automation. The Company is part of the MSCI India Index and is included in key indices of the National Stock Exchange of India, including the Nifty Midcap 50, Nifty IT, and Nifty MidCap Liquid 15, as well as several on the BSE such as the S&P BSE 100 and S&P BSE SENSEX Next 50. Persistent is also a constituent of the Dow Jones Best-in-Class World Index. The Company has achieved carbon neutrality, reinforcing its commitment to sustainability and responsible business practices. Persistent has also been named one of America's Greatest Workplaces for Inclusion & Diversity 2025 by Newsweek and Plant A Insights Group. As a participant of the United Nations Global Compact, the Company is committed to aligning strategies and operations with universal principles on human rights, labor, environment, and anti-corruption, as well as take actions that advance societal goals. With 468% growth in brand value since 2020, Persistent is the fastest-growing IT services brand in 'Brand Finance India 100' 2025 Report.


**USA**

Persistent Systems, Inc.
1731 Technology Drive
Suite 700, San Jose
CA 95110
Tel: +1 (650) 481 9180
Email: Info@persistent.com

**India**

Persistent Systems Limited
Bhageerath, 402
Senapati Bapat Road
Pune 411016
Tel: +91 (20) 6703 3000
Fax: +91 (20) 6703 6003


Persistent
Re(AI)magining™ the World