HITACHI

The tokenisation of work

From licences to metered execution

Vitor Domingos
Mahesh Raghunathan

Centre for Architecture & AI - May 2026

Hitachi Digital Services

Table of contents

Executive summary

AI vendors are moving from per-seat licences towards metered execution. Cost is now determined by how systems run, not by how many users hold access.

Frontier models, coding agents and hyperscalers each charge for some combination of seats, tokens, pooled credits and compute time. The structures and unit definitions differ, and the same workload can resolve to substantially different costs depending on which provider runs it. Credits are an internal accounting convenience. They are not a portable currency, and comparing vendors on credit price alone is misleading.

For engineering teams this means cost accumulates across multiple meters during execution: seat access, token consumption (input, cached and output), tool invocation, and underlying compute. A single change request can trigger all of them. Procurement controls applied at one meter often push spend into another. Cost management is now a system-design problem as well as a contracting problem.

> The unit of software delivery is changing. Where licences once governed who could use a tool, meters now govern how much that tool runs. Governance has to follow the meter, not the seat.

This creates a practical management problem; two development teams can deliver the same business outcome with agentic coding tools and still arrive at substantially different costs because they use different routing, context and orchestration patterns.

The call to action is clear: if organisations do not change cost controls, context strategy and routing policy, the same development team, producing the same output and quality, may face a sharp increase in agentic coding costs without a matching increase in output or quality. This whitepaper sets out the implications, proposes a control model that governs how work is routed and measured, and identifies the procurement and decision-rights questions that follow.

The shift from licences to metered execution

GitHub's pricing reflects the market direction. Copilot now supports long-running, multi-step tasks that span repositories. The request-based model that worked for autocomplete does not scale to agentic work, so the unit of charge has moved to tokens and compute, with additional metering for Actions minutes.

OpenAI and Anthropic have followed similar paths with different mechanics. OpenAI separates seat types and applies credit pools with spend controls. Anthropic charges for tokens directly, on top of access fees. Across all three vendors the seat now provides only access; the meter determines cost.

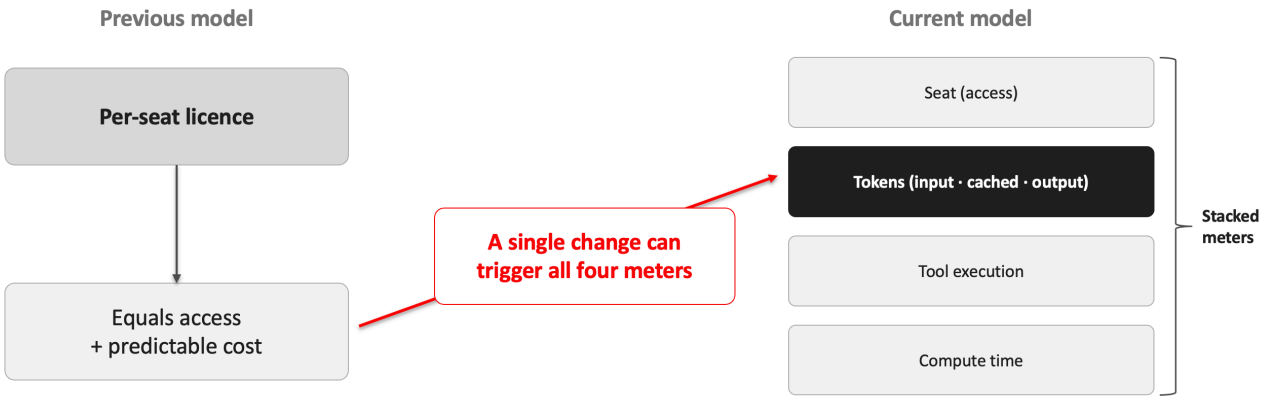


Figure 1. The pricing model has moved from a single licence cost to a stack of meters that accumulate during execution.

This shift makes execution measurable at fine granularity. Context size, model selection and workflow design now have cost consequences. They also have quality consequences, since the same architectural decisions that drive cost also drive latency and output reliability.

What changed in the vendor market

Vendors now combine access and usage in different ways. These differences are commercial and operational: they affect which workloads each provider is economical for, and they affect how much cost forecasting is possible before commitments are signed.

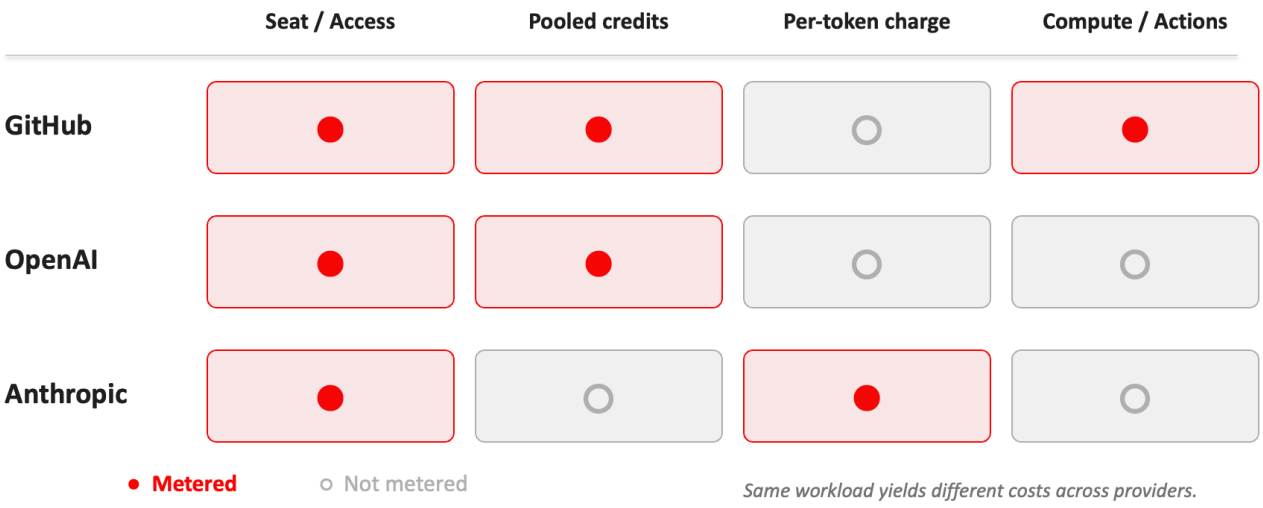


Figure 2. Vendor pricing structures compared across the four main meters.

A summary of the structures

Vendor	Access basis	Usage basis	Notable extras
GitHub	Per-seat plus pooled credits	Premium request tokens, plus compute	Actions minutes metered separately
OpenAI	Seat tiers (Plus, Team, Enterprise)	Workspace credit pool with role-based caps	No universal credit conversion across products
Anthropic	Subscription access fee	Direct per-token billing on top of access	Cached input tokens priced separately

The result is a fragmented commercial market in which pricing structure is as important as model capability. A workload that suits Anthropic's per-token model on cost grounds may suit GitHub's pooled credits on operational grounds, and the right answer is rarely the same across teams within a single organisation.

Agentic FinOps: from cloud spend to reasoning spend

So, vendors now combine access and usage in different ways;

- **Cloud FinOps** is now focused on compute utilisation, storage growth, network egress, reserved instances, VM sprawl and cost per workload.
- **Agentic FinOps** applies the same discipline to token and reasoning utilisation, context growth, tool-call and orchestration overhead, governed reasoning capacity, agent and workflow sprawl, and cost per outcome.

Finance and procurement teams will need to move beyond seat counts and negotiated discounts into workload-level controls over tokens, model routing, context reuse, autonomous retries and compute-linked execution.

What it means for teams, developers and the enterprise

For developers

Model selection and prompt design now influence cost directly. The same task can vary by an order of magnitude or more depending on context size and the model invoked. When developers treat token usage as invisible, engineering management may not see the variance until it appears on an invoice or on-screen premium usage notices.

For teams

Increased agentic activity does not automatically improve outcomes. Field reports show productivity gains alongside mixed effects on system stability, particularly where teams loosen review practices in exchange for speed. Spend has to be tied to delivered quality, not just to volume of activity.

For the enterprise

Three concerns converge at the executive level: cost control across multiple meters, visibility into how usage relates to outcomes, and governance over which execution paths are permitted for which kinds of work. Organisations that do not coordinate this end up applying caps in one place and watching spend reappear elsewhere.

> Dual metering is already common; costs combine identity (seats), execution (tokens), and infrastructure (compute). Without coordination across all three, controls in one area redirect rather than reduce spend.

Practical outcome-cost example

We understand that token spend alone is the wrong comparison, so the useful measure is cost per outcome. In an internal development team, two squads may close the same number of tickets, merge a similar number of changes and maintain the same quality bar, yet consume very different levels of agentic spend. The difference comes from how each squad routes work, manages context and controls autonomous execution.

The management question is not only whether agentic improves productivity, but whether each additional unit of reasoning spend improves delivery, quality or risk reduction. In practice, the same output can carry substantially different costs depending on whether a team uses broad context and open-ended retries, or scoped context, tiered routing and checkpointed agentic runs.

The practical lesson is that delivery volume is not enough. An open-ended agentic route can hide higher spend behind good delivery, while a controlled route can keep cost, quality and throughput visible together.

A control model for token-based execution

A token-based environment needs a control model for execution. The model has six elements between demand and execution.

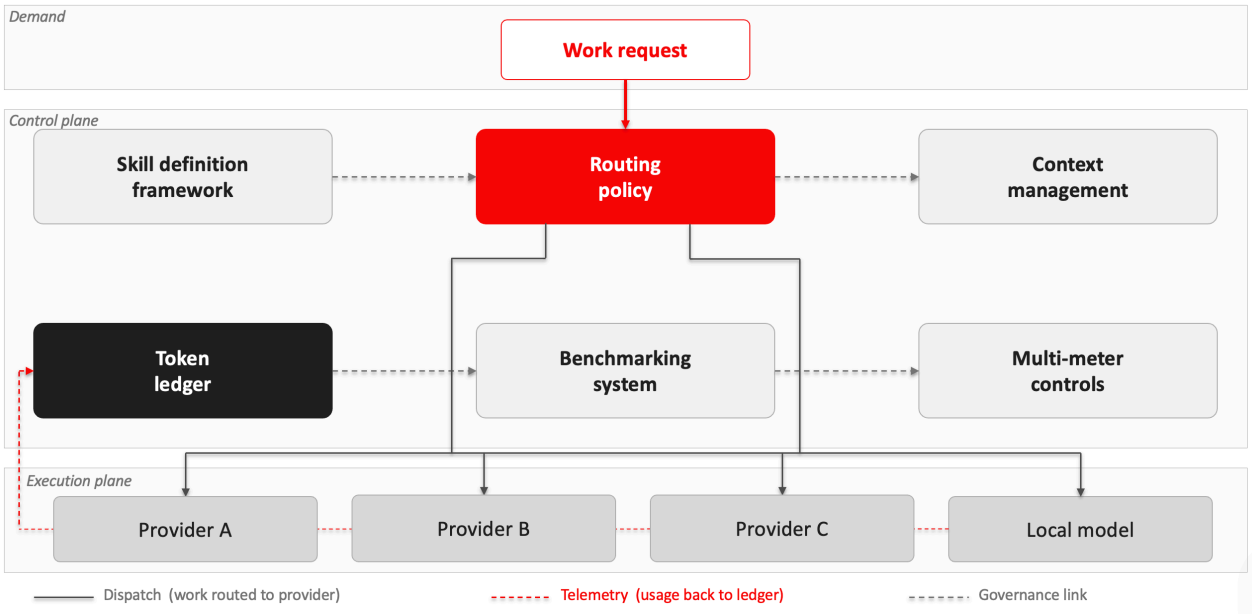


Figure 3. The control model places governance between work requests and execution targets, with telemetry returning to a central ledger.

The six elements

Token ledger
A single record of usage and cost across providers, models, tokens and compute. Without a unified ledger, every other element of the model relies on partial data.

Skill definition framework
Reusable units of work, each with explicit ownership, allowed models, expected inputs and outputs, and cost boundaries. Skills turn ad hoc requests into governed components.

Routing policy
Rules that assign each request to a model, provider and execution path based on cost, risk and performance. Policy sits between intent and invocation. A practical routing policy should define default models for routine work, premium-model escalation criteria, maximum context boundaries, retry limits, approval thresholds and fallback routes. It should also distinguish developer convenience from enterprise value: not every task deserves frontier reasoning, but some high-risk work should be routed there immediately rather than after a sequence of failed low-cost attempts.

Context management
Reusable, structured context applied consistently across tasks. Cached and shared context lowers token usage and improves consistency; unmanaged context inflates both cost and variance.

Benchmarking system
Measurement at the level of individual tasks and workflows: cost, quality, latency and rework. Benchmarks turn cost-per-call into cost-per-outcome.

Multi-meter controls
Coordinated limits applied across tokens, seats and compute together. A cap on one meter can move spend to another. Multi-meter controls reduce that risk.

Concrete routing example

Consider a hypothetical product squad using GitHub Copilot for IDE assistance and pull-request review, and Cursor for deeper agentic development.

Without routing policy, the team may default to premium reasoning models for routine edits, keep full repository context open, allow long-lived sessions to accumulate history, and let agents retry or replan without a spend boundary.

With routing policy, the same squad can use lightweight models for autocomplete and small edits, reserve premium models for architectural decisions or difficult debugging, scope Cursor context to the relevant module, and require human checkpoints before expensive multi-step execution.

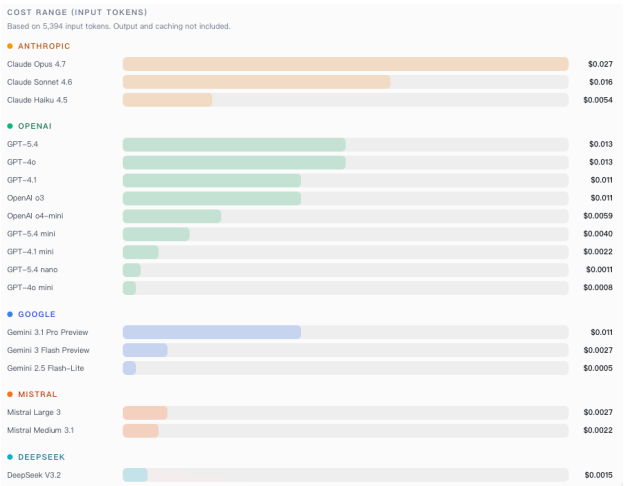
Work type	Uncontrolled route	Routed route	Cost implication
Autocomplete and small edits	Premium model available by default	Low-cost model or included completion route	Avoids premium spend on low-risk work
PR explanation and review	Full repository context sent repeatedly	Scoped diff, linked tickets and cached standards	Reduces repeated input-token load
Refactoring	Cursor agent runs against broad workspace context	Module-bounded task with explicit files and acceptance criteria	Limits context growth and retry loops
Architecture or production defect	Ad hoc escalation after several failed attempts	Immediate premium model route with evidence requirements	Spends more per run but reduces wasted cycles

Context as an economic asset

Context affects both quality and cost. Reusable, well-structured context reduces token consumption and improves consistency. Poorly managed context increases both cost and variance, and frequently increases time to a usable answer.

Vendors already support this through prompt caching and reusable skill systems. Organisations that manage context with ownership, version control and reuse policies should see lower cost per task and more predictable outcomes than those that allow context to be assembled inside individual workflows.

For example, Hitachi Digital Services tracks how much tokens agentic skills consume across different models:



How cost accumulates across meters

Cost depends on three variables: token pricing, included usage, and workload shape. The same task can differ in cost by an order of magnitude depending on model selection and context handling. Caching and reuse materially affect cost. Recomputing the same inputs increases spend without producing new value.

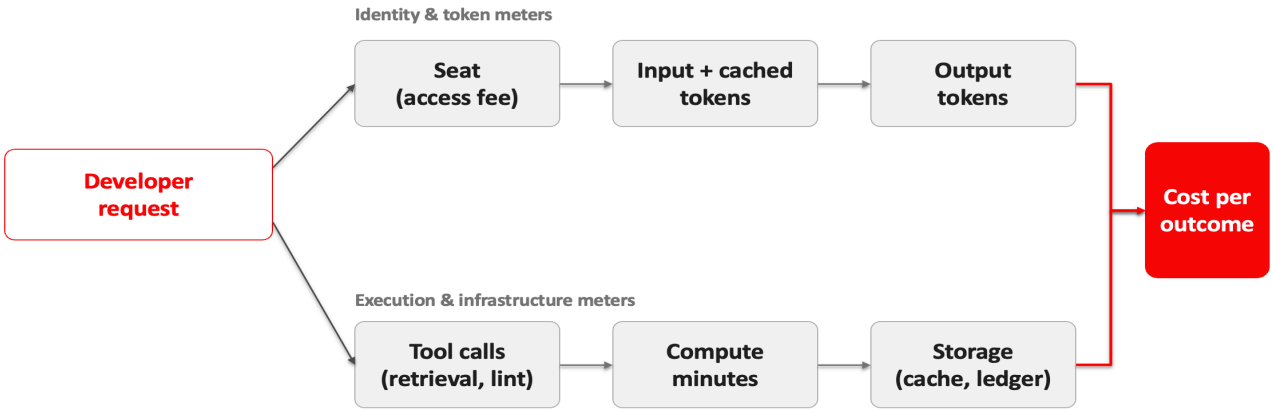


Figure 4. A single change can accumulate cost across access, token, tool-call and infrastructure meters before it reaches a cost-per-outcome figure.

The implication for governance is that meter-level limits are necessary but not sufficient. Cost accountability has to track the full execution path, not the single line item that procurement happens to monitor most closely.

Metering levers that must be governed

Metering lever	Why it matters
Context window size	Large prompts and broad repository context increase input-token consumption quickly
Frontier model usage	Premium reasoning models can materially change marginal cost per task
Agent recursion depth	Retries, replanning and self-review loops multiply inference consumption
Autonomous retries	Failed or repeated attempts can silently inflate spend without creating new value
Retrieval and tool calls	External search, MCP calls and tool execution add latency, tokens and sometimes compute cost
Human approval checkpoints	Approvals before expensive execution paths prevent runaway agent spend

Migration and implementation approach

Migration to a control model should begin with measurement, not procurement. The following sequence preserves delivery capability while bringing cost under coordinated control.

1. Establish a baseline of usage, cost and delivery outcomes.
2. Create a unified ledger linking execution to cost.
3. Classify work by risk and cost sensitivity.
4. Route work to appropriate models based on that classification.
5. Convert repeatable processes into governed, reusable units.
6. Apply policy-based budget controls across meters.

Procurement considerations

Procurement must extend beyond seat pricing to cover execution economics. Before making a commitment, procurement should ask for:

- Detailed pricing across tokens, caching and compute
- Model substitution terms, including price-protection clauses
- Credit rules: expiration, rollover and pooling boundaries
- Access to usage and audit data at the granularity needed for chargeback
- Multi-vendor responsibility boundaries when workloads span providers
- Coverage of local versus cloud execution under the same agreement
- AI FinOps reporting rights, including exports that connect user, model, skill, context pack, tool execution and outcome identifier.

Decision rights and governance

Clear ownership across functions matters more than any single tool choice. Token-based execution cuts across procurement, engineering, security and platform engineering, and each function holds part of the control model.

Finance and procurement should own the AI FinOps envelope: budget policy, spend thresholds, chargeback and contractual protections. Engineering should own routing, context, benchmarks and the evidence that spend improves delivery outcomes. Platform teams should own the telemetry substrate that joins those two views.

Function	Primary responsibility
Finance	Spend policy and budget controls
Engineering	Routing, execution design and benchmarks
Security & Legal	Data handling, residency and compliance
Platform teams	Control systems, ledger and telemetry

The primary metric is cost per outcome, such as cost per merged change or cost per ticket resolved, alongside quality and reliability measures. Reporting on token spend in isolation will produce the wrong incentives.

Where to start

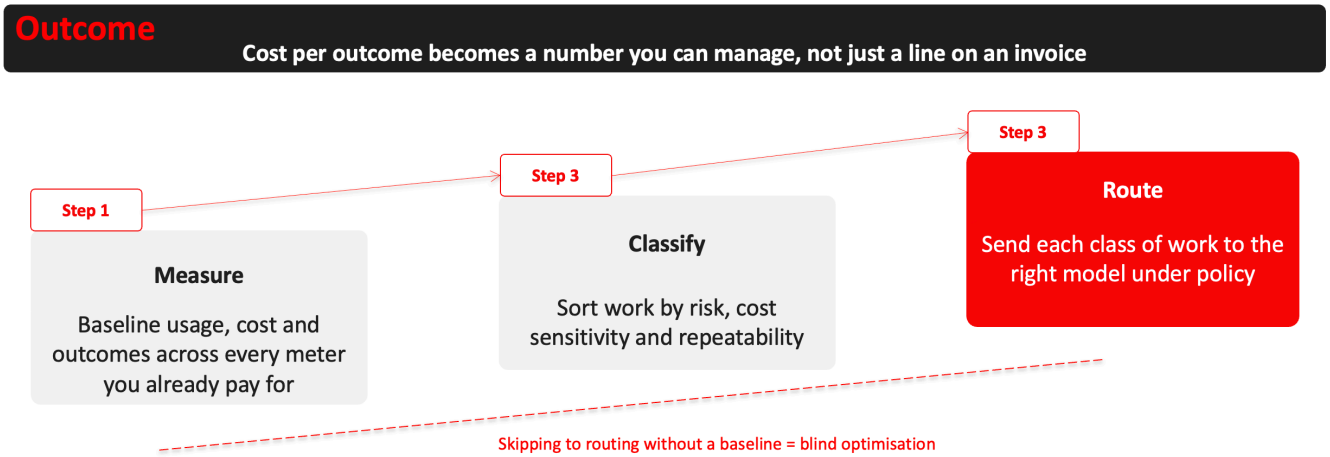
The fastest route to control starts with measurement: measurement first, classification second, routing third. Organisations that try to skip routing without a baseline find themselves optimising blindfold.

Avoid uncontrolled usage costs

The near-term risk is not that agentic coding tools fail to deliver value. The near-term risk is that they deliver value through unmanaged consumption.

Organisations should baseline current AI-assisted development usage before usage-based billing takes effect, classify their highest-volume workflows, define routing policy for Copilot, Cursor and equivalent tools, and connect spend to delivery outcomes.

Without these changes, the same development team can ship the same work at the same quality level and still face a substantially larger bill.



> Leaders should be able to answer five questions: what is being metered, how work is routed, who owns the context, what each outcome costs, and whether the spend improves quality, speed or risk reduction.