Big-T Notation

Engineering for Token Efficiency in the Age of Enterprise AI

Dan Neff & **Brian Scott**

Principal Architects · Adobe

Enterprise AI Summit 2026 · IT Revolution

April 24–25 | San Jose, CA

Every era of computing has a unit of scarcity

1960s	Memory bytes	— programmers counted every allocation
1990s	Network bandwidth	— shaped every client-server design
2010s	API calls & egress	— cloud unit economics
2025+	**Tokens**	**— the unit of AI-native computing**

Engineers who understood the new unit of scarcity built systems that scaled.
The ones who ignored it built systems that bankrupted their sponsors.

The Jevons Paradox of AI

280x

per-token cost reduction
in 2 years (GPT-3.5 level)

$106B

inference market 2025
→ $255B by 2030

> **More miles per gallon — but 50x more gallons burned.**

Inference = 80–90% of lifetime AI product costs. HBM, energy, and data center costs are pushing that floor back up.

Four demand waves — stacking simultaneously

1 **Frontier models are more capable — and more expensive per task**
Reasoning models demand up to 100x more compute. A single code review can cost 5–10x more with chain-of-thought.

2 **Larger context windows invite larger prompts**
4K → 200K+ tokens in under 3 years. When the window is big, the path of least resistance is to dump everything in.

3 **Enterprise adoption is compounding, not linear**
88% of orgs using AI. 12% of users consume 50% of tokens. Power users are 100x more token-intensive.

4 **Infrastructure costs are rising**
Blackwell cabinets: $2–3M each. HBM prices doubled. Data center spend → $650B in 2026.

Introducing

Big-T Notation

Big-O tells you how **runtime grows** as input scales.

Big-T tells you how **token consumption grows** as usage scales.

$$T(n \cdot k \cdot a)$$

n	×	**k**	×	**a**
requests or input size		model calls per request		agent depth (sub-agents)

Not a formal mathematical system. A **thinking tool** — a shared vocabulary for reasoning about token cost trajectories before committing to an architecture.

The complexity classes

`T(1)`	**Constant.** Cached response, embedding lookup, pre-computed result. Model not invoked per request.
`T(log n)`	**Sublinear.** Code filters input before the model sees it. SQL, regex, embeddings — tokens eliminated **before** inference. The retrieval promise done right.
`T(n)`	**Linear.** One model call per request, proportional to input size. The baseline everyone budgets for.
`T(n·k)`	**Multiplicative.** k model calls per request — RAG chains, multi-turn, reasoning tokens. k is often **invisible**. Extended thinking: 128K tokens consumed, 500 visible.
`T(n·k·a)`	**Agent-multiplicative.** Orchestrator → sub-agents → tool calls → more model calls. The $O(n^2)$ of AI: architecturally powerful, economically dangerous.
`T(∞)`	**Unbounded.** Autonomous loops with no termination. Retry logic that re-prompts on every failure. The infinite loops of the token economy.

T(log n) — Code is the new cache layer

Deterministic code reduces the input set **before** inference. Tokens leave the pipeline entirely.

92%
token reduction — Flexpa
SQL views: 240K → 19K tokens

99.95%
token reduction — MIT CSAIL RAGO
1M corpus → 512 tokens to LLM

99.98%
elimination — Legion Intel
2M → 250–350 tokens via retrieval

> As the data corpus grows, tokens consumed by the model grow slowly or stay nearly constant. This is what RAG was supposed to do — **keep token consumption from scaling with data volume**.

Complexity is a property of the interface, not the task

Naive: 30 individual tools

Agent loads 30 tool schemas into context.
Chains 5 tools — each round trip replays full context + growing history.

T(n·k) with large constant c

→

Refactored: single composed interface

One `execute_script` tool accepts a command sequence.
Agent composes full 5-step operation in one call.

T(n) — order of magnitude drop

Same task. Same output. Same tools doing the same work. **The interface drove the token cost, not the task.**

```
Think:  find . -name "*.log" | grep ERROR | sort | head
vs. asking someone to run each command separately and read you the output between each one.
```

Code generation as native compression

Tool-calling pattern

Create 31 calendar events → 31 individual tool calls.
Each is a round trip with context replay.

`T(n·b) — b operations, each a full agent turn`

Code Mode

Agent generates a program — a loop that constructs payloads and makes API calls.
One generation step. Executes in sandboxed runtime.

`T(n) — b operations run outside the token economy`

81%

token reduction on batch tasks
(Cloudflare Code Mode, Dec 2025)

> Asking an LLM to perform work through tool-calling schemas is like asking Shakespeare to take a crash course in Mandarin and then write a play in it.
>
> — Kenton Varda, Cloudflare

Five levers of token efficiency

1. MODEL ROUTING
Route the right task to the right model. 1,750x price spread across the landscape.

2. PROMPT ARCHITECTURE
Serialization, compression, output format constraints. Prompt structure impacts cost as much as model selection.

3. CACHING
Prompt caching, semantic caching. Turn T(n) workloads into T(1) for the common case.

4. ABSTRACTION TRANSPARENCY
You cannot optimize what you cannot measure. Credits and seat licenses hide the token.

5. WORKLOAD CLASSIFICATION & GOVERNANCE
Make sure the workloads burning the most tokens are the ones producing the most value. Circuit breakers on agentic workloads. Budget thresholds that trigger review.

Lever 1 — Model routing is architecture

1,750x

price spread across available models
$35/MTok (o3-pro) → $0.02/MTok (Qwen 0.8B)

> **UC Berkeley RouteLLM (ICLR 2025)**
> 85% cost reduction on MT Bench, 95% of GPT-4 quality retained. Only 26% of queries needed the frontier model.

> **Stanford FrugalGPT**
> 98% cost reduction on financial news classification, matching GPT-4 accuracy. Cheaper models sometimes **outperformed** expensive ones.

Nuance: Routing reduces **cost per token**, not **volume**. Less capable models often need more verbose prompting and retries. **`Total Cost = T(volume) × price(model)`** — reducing one can increase the other.

Lever 2 — Prompt structure impacts cost as much as model selection

Input: serialization is the hidden cost driver

CSV: 56% fewer tokens than JSON
— no quality loss (GetCrux, 10K questions)

TOON format: 40–60% reduction vs JSON
— declare field names once, list values row-by-row

Poor serialization routinely consumes 40–70% of available tokens through formatting overhead alone.

Output: the 4x pricing asymmetry

Output tokens cost **3–8x more** than input tokens across all major providers.

Function calling with constrained decoding:
42% output token reduction
— Microsoft DS team

Deterministic schema elements don't benefit from LLM generation. Let code handle what code handles better.

Lever 3 — Caching is a design principle

If a workload is T(n) today, check whether caching can make it T(1) for the common case.

90%

input cost reduction
prompt prefix caching (stable system prompts)

73%

cost reduction
Redis LangCache (semantic caching, high-rep workloads)

42%

semantic cache hit rate
~3x higher than exact-match, 15ms overhead

> **The mental model:** This is identical to how web engineers think about CDNs and database query caches. `T(1)` for cache hits, `T(n)` as fallback for misses. The principles are the same — only the medium has changed.

Lever 4 — Opaque abstractions hide cost

Any abstraction that prevents you from seeing token-level economics also prevents you from optimizing them.

Credits	No fixed relationship to tokens. Can't determine which model processed a prompt, how many tokens it consumed, or effective per-token cost. Monthly limits consumed in 15-minute sessions.
Hybrid	Token-based rate cards layered on credit systems. Per-message estimates, token rates, and credits coexist. Abstraction layers accumulate rather than simplify.
Direct	Published per-token rates, no intermediate credit system. BYOK with full telemetry. Every API call at provider's published rate.

If your tooling doesn't expose token-level telemetry, you're paying for the privilege of not knowing what you're paying for.

The Token Efficiency Pipeline

INSTRUMENTATION & GOVERNANCE — Telemetry · Cost Attribution · Circuit Breakers

Raw Workload	`T(n·k·a)`
↓	
L1 — Deterministic Preprocessing SQL · Embeddings · Regex	`→ T(log n)`
↓	
L2 — Cache Check Semantic + prefix	`hit = T(1)`
↓ miss	
L3 — Model Router Classify → route to tier	`cost/tok ↓`
↓	
L4 — Prompt Architecture Serialization · dedup	`volume ↓`
↓	
L5 — Inference Minimum necessary tokens	`optimized`
↓	
L6 — Output Mode Direct response or code gen	`→ T(n)`

Layers 1–2 operate **outside** inference — largest absolute savings. Layer 3 reduces cost, not volume. Layers 4–6 optimize within inference.

The question to take back

> What is the token complexity class of this workload, and is that complexity justified by the value it produces?

Start Monday

- Instrument token consumption by workload
- Identify your T(n·k) and T(n·k·a) systems
- Ask: can deterministic code reduce the input set?

Start this quarter

- Model routing — stop running everything on frontier
- Caching layer for high-repetition workloads
- Abstraction audit on your AI vendor contracts

Where We Need Help

Two open problems we haven't solved yet

"Show us your invoices"

We've analyzed a large volume of developer and production data internally, but we don't want to calibrate this framework in a silo. If you're tracking token spend by task type and model tier, we'd like to collaborate — compare patterns, pressure-test the complexity classes, and benefit from collective wisdom across industries.

"Who's solved the downstream problem?"

When agents call systems of record at machine speed, the bottleneck moves to API gateways, rate limits, and service account governance. If you've built the infrastructure to handle agent-speed access patterns against ERP, CRM, or ITSM backends — we want to learn from you.

10x organizations won't use fewer AI tokens.

They'll use them better.

Efficiency is the architecture that makes ambition sustainable.

Dan Neff & **Brian Scott** · Principal Architects, Adobe
Enterprise AI Summit 2026 · IT Revolution