ARCHITECTURE · AI INFRASTRUCTURE · 2026

The *AI Execution Tax*: Why Intelligence Must Move Closer to the Data

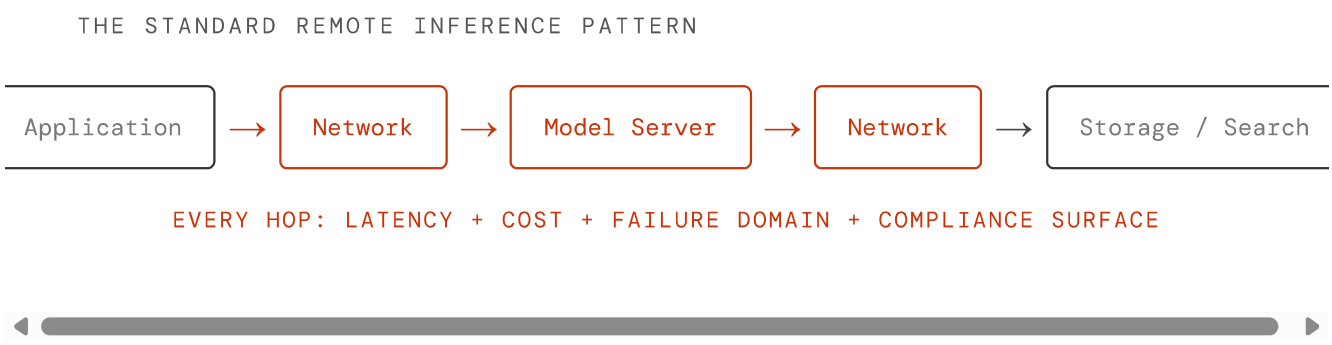
Every time your application ships data to an external model server, you're paying a toll — in milliseconds, in dollars, in risk. A quieter architectural shift is underway that eliminates it entirely.

Steven Lopez · Distributed Systems · AI Infrastructure · January 2026

For two years, enterprise AI has followed a comfortable script: attach a model-serving cluster to your existing stack, wrap it in an API, and call it real-time AI. It works. It also hides a compounding cost that most teams don't see until it's too late to fix cheaply.

That cost has a name: the **AI Execution Tax**. It isn't a line item. It's an architecture — one that treats inference as a remote service, ships your data across a network boundary, waits for a response, and then ships it back. At low throughput, this is invisible. At scale, it becomes the dominant operating cost of your entire AI stack.

The fix isn't a faster model server. It's a different mental model entirely.



Count the hops. Each one adds latency and jitter. Each one is a serialization round-trip. Each one is an independent failure domain you have to monitor, scale, and pay for. Each one expands the perimeter your security team has to defend. For a single inference call, this is a curiosity. For a system processing millions of events per hour, it's an architectural tax that compounds every time you add a workload.

> *The issue is not model quality. It's where inference executes — and the answer has been wrong for most production systems built in the last two years.*

What You're Actually Paying

The AI Execution Tax shows up in four places that most post-mortems miss until the architecture is already load-bearing and expensive to change.

LATENCY

Hundreds of ms per event — for a remote call you'll never see in a profiler

COST

Per-call pricing scales linearly with throughput — your AI bill grows with your success

EXPOSURE

Data leaves your boundary on every inference — multiplying your compliance surface

FRAGILITY

More clusters to scale, monitor, fail, and debug — independently

None of these are edge cases. They're the normal operating cost of the dominant remote-inference pattern. They were tolerable during the experimentation phase, when throughput was low and "good enough" was the bar. They become intolerable when AI moves from prototype to infrastructure.

How We Got Here

Distributed inference became the default for reasons that made sense at the time — and most of those reasons no longer apply.

ORIGINAL ASSUMPTION	STATUS IN 2026
Models require GPU-dense infrastructure	`Outdated` — CPU-optimized inference runtimes now cover a wide class of production workloads
JVM-native inference was impractical	`Outdated` — portable model formats and modern concurrency have changed the cost model
Isolation between systems is safer	`Inverted` — keeping data in-process is often more secure than externalizing it
Distributed always means more scalable	`Overgeneralized` — many AI workloads benefit most from locality, not distribution
External API services reduce complexity	`Partially wrong` — they shift complexity from code to operations, and add a bill

The industry was right to distribute inference when models were large, GPU-dependent, and JVM-native execution wasn't viable. Those constraints have shifted. Modern CPUs, purpose-built inference runtimes, and portable model formats have changed the trade-off surface significantly — particularly for CPU-bound inference workloads like embedding generation, classification, and scoring.

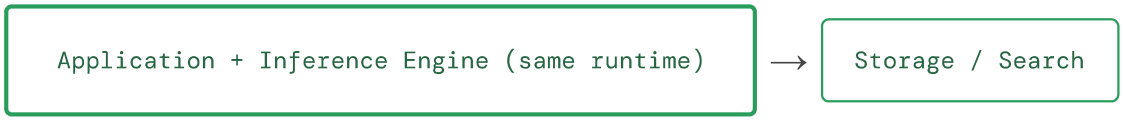
The Architectural Correction: In-Process AI

The idea is simple enough to state in a sentence: instead of shipping your data to the model, ship the model to your data — and run inference inside the same process that already owns the event stream, the state, and the business logic.

This pattern — **In-Process AI** — eliminates the network boundary entirely. Inference happens in the same memory space, under the same scheduler, and

within the same security perimeter as the application. The result isn't just faster execution. It's *predictable* execution, with fundamentally different operational characteristics.

THE IN-PROCESS PATTERN



ZERO NETWORK HOPS. IN-MEMORY. SAME SCHEDULER. SAME SECURITY BOUNDARY.

KEY PROPERTIES

What In-Process AI delivers operationally

- **Latency:** Microseconds to low milliseconds for inference — not hundreds of milliseconds
- **Predictability:** Deterministic execution; no network jitter hiding in your p99
- **Security:** Zero-egress inference — data never leaves your memory space
- **Portability:** Same logic runs on cloud, on-prem, edge, and air-gapped systems
- **Cost:** No per-call pricing; no separately billed inference infrastructure
- **Correctness:** Inference and event delivery share the same transactional boundary

That last point deserves emphasis. When inference executes inside the same runtime that manages event flow and state transitions, correctness becomes *easier* to reason about — not harder. Failure semantics are local. Backpressure is immediate and observable. You don't lose an event to a network timeout and wonder whether the model saw it.

The Correctness Problem Nobody Talks About

Real-time AI pipelines don't just need to be fast. They need to be right. Duplicate or dropped events aren't just a data quality issue — in AI systems, they silently corrupt the artifacts that downstream systems depend on.

WHY IT MATTERS

What happens when inference and delivery semantics diverge

- Duplicate events produce duplicate vectors — search indices drift from ground truth
- Dropped events create gaps — embeddings disappear from the query surface silently
- Skewed analytics downstream follow the vectors, not the original events
- Model behavior becomes hard to debug when the training and serving distributions diverge

When inference executes inside the same runtime that enforces delivery guarantees — rather than as a separate external hop — these problems are structurally prevented rather than manually compensated for. The AI-generated artifact carries the same delivery semantics as the event that produced it. This matters especially when vectorized data becomes a first-class query surface in downstream systems.

Who This Is For — And Who It Isn't

In-Process AI is not a universal architecture. Recognizing its boundaries is part of what makes it credible.

WORKLOAD TYPE	RIGHT TOOL
Event enrichment, classification, embedding at ingest	In-Process AI
Fraud detection, cybersecurity scoring (per-event)	In-Process AI
RAG pre-processing, semantic search vectorization	In-Process AI
Large-scale cross-stream aggregation	Distributed Stream Processor
Continuous online model training	Distributed ML Platform
GPU-bound real-time inference at massive scale	Specialized GPU Infrastructure
Retrospective analytics, batch BI	Batch / Warehouse

Distributed compute frameworks aren't going away. Apache Flink is the right answer for workloads that require massive, cross-stream state and cluster-wide coordination. The goal of In-Process AI is **architectural precision** — using distribution where it creates value, and eliminating it where it creates overhead.

Where This Matters Most

Some environments have needs that make In-Process AI less of an optimization and more of a necessity. The pattern is especially consequential in three contexts.

Regulated Industries

Financial services, healthcare, and government environments face strict data residency and compliance requirements. Zero-egress inference keeps both data and derived intelligence within the controlled boundary — auditability included.

Defense & Aerospace

Systems that must sense, decide, and act without guaranteed connectivity cannot depend on centralized model servers. When the network is degraded or contested, inference autonomy is a survivability requirement, not a performance goal.

Sovereign & Edge Deployments

National data residency requirements, disconnected edge locations, and sovereign cloud mandates all point to the same conclusion: the architecture needs to carry its intelligence with it, independent of external connectivity.

In each of these cases, the limiting factor isn't model capability — it's where inference is allowed to execute and under whose control. That's an architectural question, not a model-selection question.

The 2026–2027 Outlook

Regulatory scrutiny is intensifying. Cost pressure on AI infrastructure is real and growing. Sovereignty requirements are expanding into sectors that previously had flexibility. These aren't speculative trends — they're shaping procurement decisions and platform roadmaps now.

ADOPTION TRAJECTORY — IN-PROCESS AI BY SECTOR (PROJECTED)

Defense	Dominant by 2027
Financial Svcs	Mainstream
Healthcare	Growing
Edge / IoT	Growing
General Cloud	Emerging

By 2027, In-Process AI is likely to be a baseline architectural expectation for federal, financial, healthcare, and critical-infrastructure systems — not because it's novel, but because the alternative will carry compliance and operational costs that are no longer defensible.

> *Organizations that internalize this shift early will define the next generation of autonomous, sovereign, and cost-efficient AI platforms. Those that don't will accumulate architectural debt that compounds with every workload they add.*

The Bottom Line

The industry has been paying an AI Execution Tax that was never itemized on any invoice but shows up plainly in latency numbers, infrastructure bills, compliance audits, and production incidents. It was a reasonable trade-off when models were large, runtimes were limiting, and AI was still in the lab. Those conditions have changed.

In-Process AI — running inference inside the same runtime that owns the data, the state, and the control flow — collapses complexity, reduces cost, strengthens security posture, and restores alignment between how AI systems are architected and how they actually behave under production load.

The architectural correction is underway. The question is whether your team leads it or follows it.

This article discusses general architectural patterns and system-level trade-offs. Implementation details are intentionally omitted. Latency figures are illustrative of class differences between in-process and remote inference models under representative workloads.

Steven Lopez

Distributed Systems · AI Infrastructure · Real-Time Architectures