The Future of AI Might Look More Like Databases Than Chatbots

How In-Process Inference and Operational Databases Build the Foundation for World-Model AI

By Steven Lopez

Yann LeCun has been making an argument for years that the AI mainstream tends to ignore. In talks at places like the Collège de France and his published work on Joint Embedding Predictive Architectures (JEPA), he contends that large language models — however impressive — are not a path to genuine machine intelligence. What's missing isn't scale. It's grounding. LLMs learn statistical patterns over tokens. What intelligent systems actually need, he argues, are world models: internal representations of cause and effect, physical reality, and how actions produce outcomes. Systems that don't just predict the next word, but anticipate the next state of the world.

That distinction — between language models and world models — has a quiet but significant implication for how we should be thinking about AI infrastructure.

Intelligence Requires Memory

A world model isn't a static artifact. It's a continuously updated representation of a system's environment. To maintain that representation, an AI system needs something that LLMs famously lack: persistent, queryable, updateable state.

This is why the database conversation is more important than it looks. When people talk about AI infrastructure, they tend to mean GPUs, model serving, and inference optimization. But none of that addresses the fundamental requirement of a world model: somewhere to put what the system has learned, in a form it can retrieve and reason over later.

This is the problem modern operational databases already solve well. A document store like MongoDB doesn't just hold records — it maintains application state, event histories, vector embeddings, and relationship graphs in formats that are fast to query and easy to update. That's not coincidentally useful for AI. It's structurally necessary for any system that needs to maintain context across time.

The Pipeline Problem

Most production AI systems today are built around a familiar pattern: data arrives, gets shipped to a model server via API call, a result comes back, and that result gets written somewhere. The model is a remote service, consulted on request.

This architecture has a fundamental tension with the world model concept. If intelligence is supposed to be embedded in the system — continuously updating, responding to new events as they happen — then a request/response model that calls out to a separate

inference server isn't really embedded. It's bolted on. Every API call is an admission that the intelligence lives somewhere else.

The latency cost is real. Serialization overhead alone can represent **35–45% of total pipeline latency** in typical AI pipelines. But the architectural cost is more significant: you've designed a system where intelligence is a consulted oracle rather than an intrinsic property of the data layer. (For a full breakdown of where this overhead comes from, see *The Serialization Tax*.)

A Different Architecture

What changes when inference runs inside the streaming runtime itself?

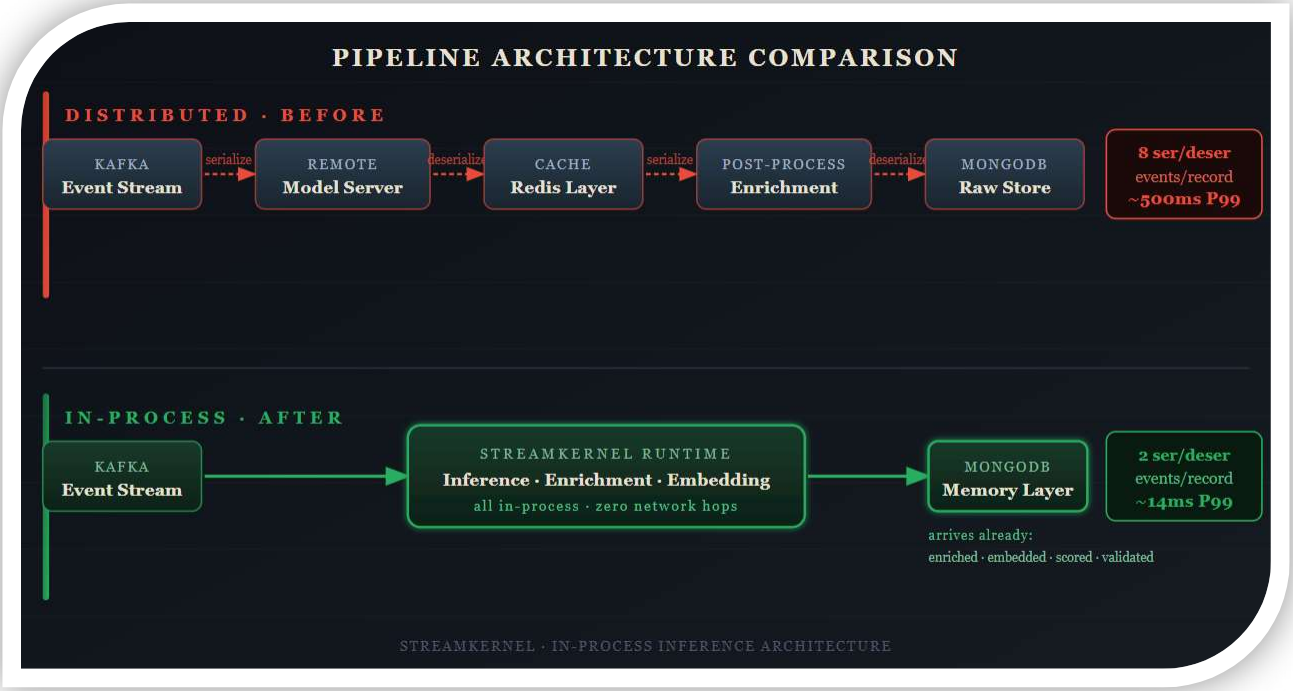
In the standard model, the pipeline looks like this: `Event → Model Server → Database`. Each arrow is a network call — a serialization event, a round-trip, a point of failure. With in-process inference, two of those three hops disappear entirely.

Consider a financial transaction pipeline. In the standard architecture, an event arrives in Kafka, gets forwarded to a model server for fraud scoring, the score comes back, and the enriched record lands in the database. The model sees one transaction at a time, in isolation, with no memory of what came before.

In an in-process architecture — something StreamKernel is built to enable — the inference logic runs inside the runtime that's consuming the stream. The model has access to the full context of the pipeline: prior events, aggregated state, embeddings stored in MongoDB from previous sessions. When the enriched record lands in the database, it arrives already scored, already contextualized, already embedded — not raw data waiting to be processed, but data that has already been reasoned over.

The pipeline becomes:

```
Event Stream → StreamKernel Runtime → Context Lookup (MongoDB)
  → In-Process Inference → Enriched Event → MongoDB
```



In this architecture, MongoDB isn't just a passive store. It's the memory layer that makes continuous learning possible. The runtime reads from it to contextualize new events. It writes back to it to update what the system knows. Over time, the system's behavior improves not because the model was retrained offline, but because its operational context — stored in MongoDB — grew richer.

Why This Matters for LeCun's Argument

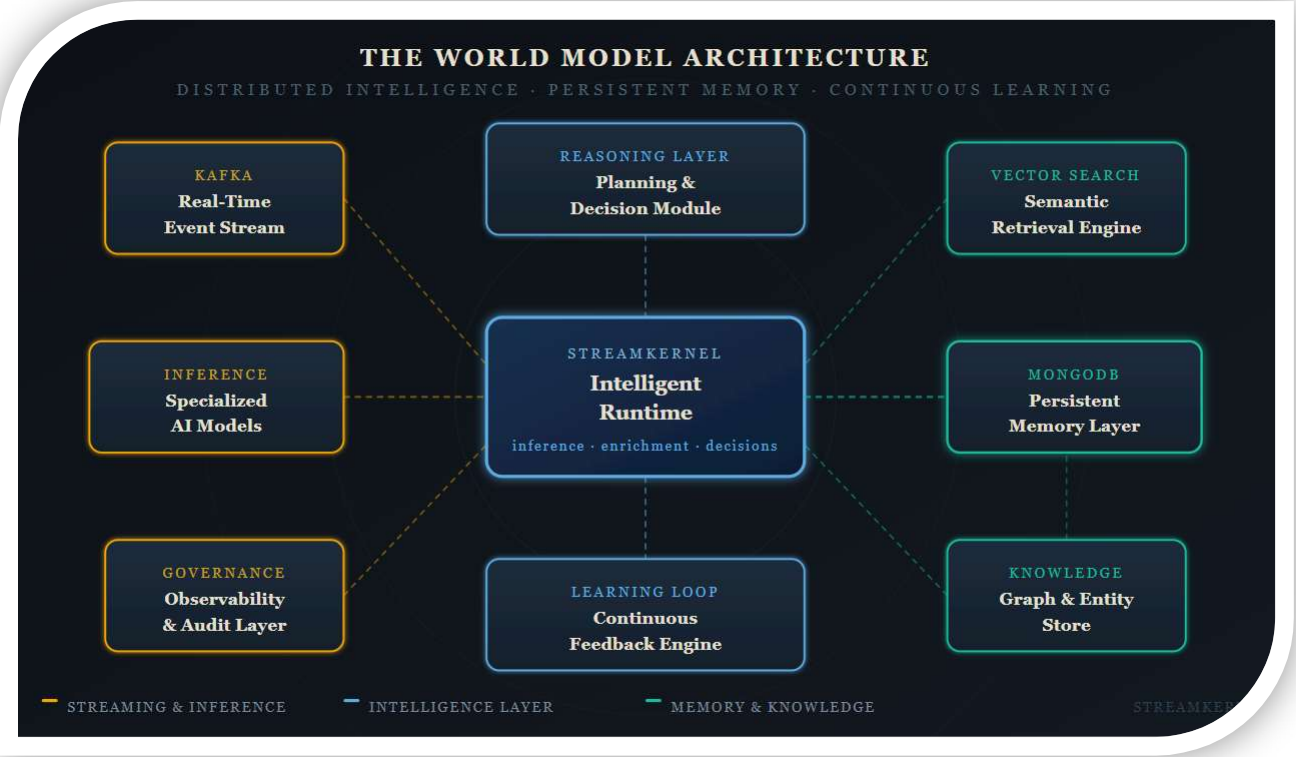
LeCun's world model thesis requires three things that current AI architectures struggle to provide: real-time feedback from the environment, persistent memory that survives across interactions, and inference that's embedded in the system rather than wrapped around it.

The architecture above addresses all three. The streaming runtime provides real-time feedback. MongoDB provides persistent memory. In-process inference means the intelligence is part of the data pipeline, not an external service it depends on.

This is speculative at the frontier — nobody has built a true world model yet, and LeCun would be the first to say we're far from it. But the architectural direction is clear. Systems that can learn continuously, maintain context, and embed intelligence close to the data will be better positioned for whatever comes next than systems that treat inference as a remote call.

The Bigger Picture

The future LeCun describes isn't a single massive model that knows everything. It's a distributed system of specialized components — real-time runtimes, persistent knowledge stores, vector search engines, planning modules — each handling a specific capability, collectively producing something that behaves intelligently.



In that picture, the competition isn't just about who trains the best model. It's about who builds the best system around intelligence. And systems are, fundamentally, an infrastructure problem.

The teams building for that future aren't just thinking about GPUs and inference speed. They're thinking about memory, continuity, and where in the stack intelligence actually lives.

That’s a database problem just as much as it’s a model problem. And it's one worth taking seriously now.