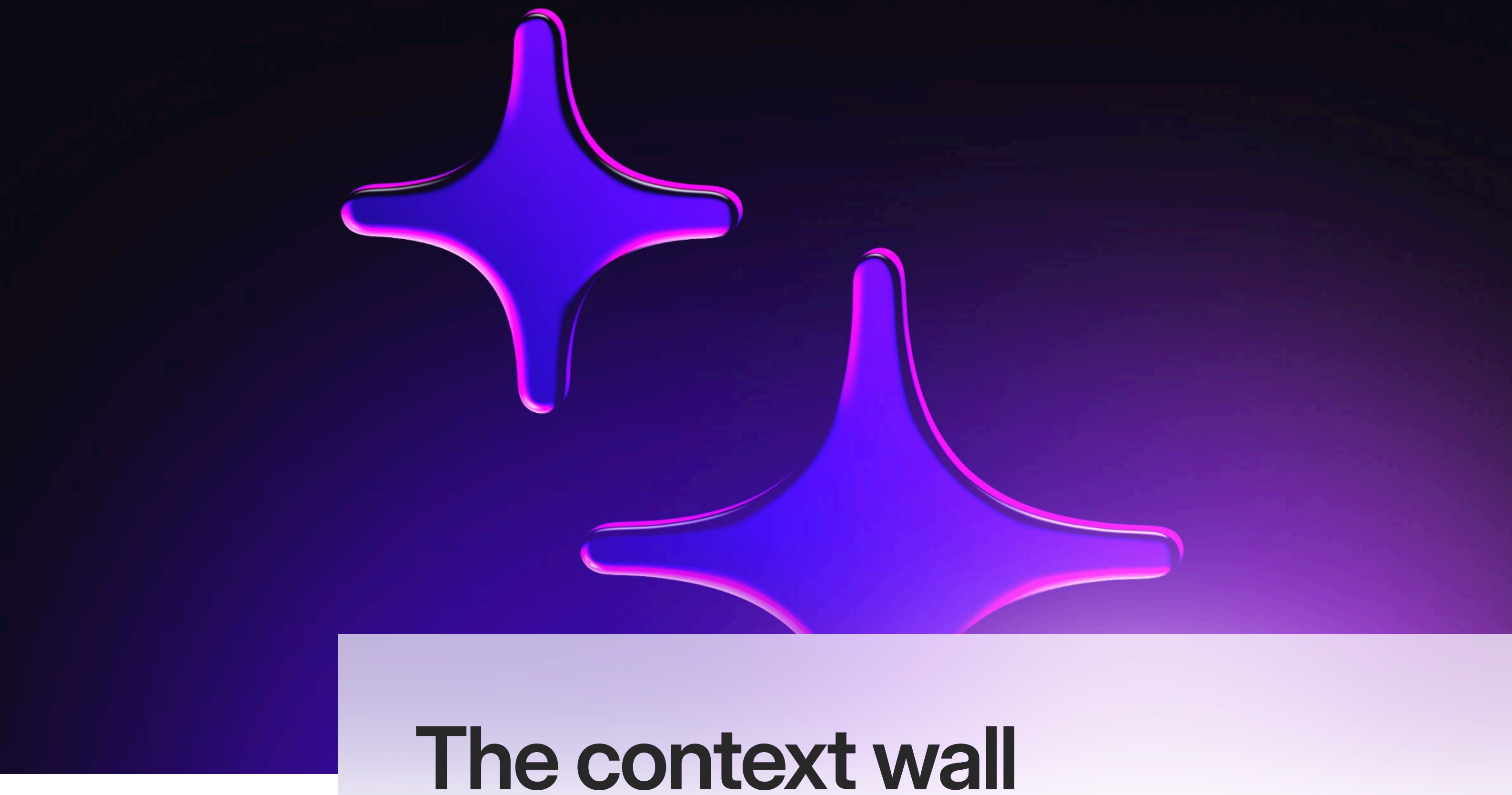


The context wall
Why AI agents require a multi-model semantic foundation

Beyond knowledge graphs to real-time contextual reasoning

This paper introduces the Enterprise Semantic Foundation (ESF), a unified, multi-model substrate that merges the stability of Knowledge Graphs with the dynamism of Context Graphs. It argues that the next leap in AI utility will not come from larger models, but from a shift in data architecture.

SurrealDB

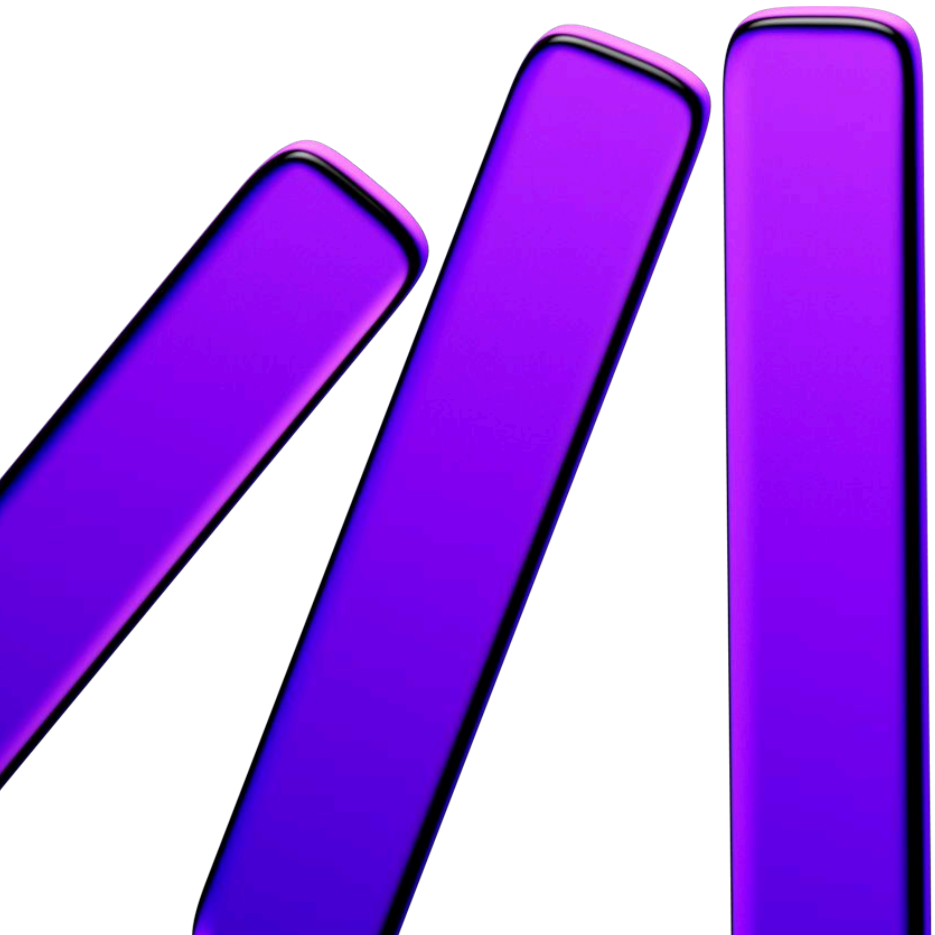
Table of contents

Executive abstract
The context wall

Enterprise AI initiatives in 2026 share a common failure pattern. The cause of this failure is not the model. It is the architecture beneath the model. As the initial hype surrounding Large Language Models (LLMs) matures into the deployment of autonomous agents, enterprises are hitting a "Context Wall." While LLMs provide the reasoning engine, they lack a persistent, structured, and real-time memory layer.

Current attempts to bridge this gap via Retrieval-Augmented Generation (RAG) often rely on fragmented "frankenstacks" - disparate vector, graph, and document databases stitched together with complex glue code. This paper argues that AI agents require a new kind of database foundation: one that can represent entities and their relationships as a native graph, store and retrieve unstructured content as documents, support semantic search across embeddings, maintain structured records, and do all of this within a unified query model and consistent transactional guarantees.

We introduce the concept of the **Enterprise Semantic Foundation (ESF)** - a unified, multi-model substrate that merges the stability of **Knowledge Graphs** with the dynamism of **Context Graphs**. By moving beyond the limitations of data platforms and single-model stores, SurrealDB provides the sub-millisecond, ACID-compliant environment, with native graph support, required for agents to reason, collaborate, and evolve in real time.



Executive summary

The problem

LLMs are stateless; RAG is a temporary patch. Agents need "living" memory.

The thesis

Traditional single-model stores and fragmented data platforms are structurally misaligned with the requirements of agentic state.

The solution

A unified "Context Graph" built on a multi-model foundation that treats storage as an active participant in reasoning.

1. The 2026 enterprise reality

Failed pilots

Most enterprise AI agent deployments in 2024 and 2025 failed not because models were too small, but because agents lacked proper context. Brittle workflows, stale data, and misaligned business definitions made agents unreliable in production.

Cost and governance pressure

Enterprises are now scrutinising AI infrastructure spend. A “frankenstack” of four databases is not just architecturally fragile; it is expensive to operate, difficult to govern, and hard to audit. Procurement teams are asking for consolidation.

The missing ingredient

It is not a better model. It is a data architecture designed for agency from the ground up.

2. The architecture of agency

The latency of learning

Modern data platforms are optimised for analytical throughput, not the sub-second read-think-write loops that agents require.

The "state" problem

Agents manage goals, sub-tasks and working memory. This requires ACID-compliant transactional control that analytical platforms cannot provide.

The missing layer

Every existing data architecture lacks a dedicated context layer - a low-latency, transactional substrate that holds the live state of an agent's understanding and pushes updates rather than waiting to be polled.

How context gets built

Context enters the system through automated pipelines, agent-generated observations, human-curated business rules, and self-correcting feedback loops. A multi-model foundation supports all of these patterns natively.

3. The context layer: why multi-model wins

The fragmented memory tax

When memory is split across a vector store, a graph database, and a document store, the “semantic join” happens in application code. Every retrieval compounds latency and introduces the risk of semantic drift.

Why memory middleware is not enough

Products such as MemO provide a useful abstraction layer, but are built on top of the same fragmented infrastructure. A memory layer that sits above four inconsistent stores inherits their synchronisation problems.

Atomic context

In a multi-model system, a graph edge isn't just a pointer; it’s a document. It can hold the weight of a relationship, the timestamp of the last interaction, and the vector embedding of the context, all retrievable in a single query.

Reduced cognitive load

Agents fetch a ‘complete thought’ in one operation rather than reconstructing it from multiple API calls.

Introduction
From static graphs to active context

Enterprise infrastructure has evolved through three distinct generations.

The first was the **semantic layer**: hand-built YAML definitions and LookML models that mapped business terms like "revenue" and "churn" to specific SQL queries. Semantic layers brought discipline to analytics, but they were static, manually maintained, and brittle. When a team member left or a product line launched, the definitions went stale. As a16z has documented, agents attempting to answer "What was revenue growth last quarter?" routinely failed because the semantic layer they depended on had not been updated in months.

The second was the **knowledge graph**: structured representations of what an organisation knows, encoded as nodes and relationships, designed to help humans and early AI systems navigate complex information landscapes. Knowledge graphs brought relational structure to enterprise data, including ontologies, entity resolution, and canonical facts. They remain powerful tools for search, recommendation, and retrospective analysis.

The third, now emerging, is the **context graph**: a live, transactional representation of what is happening right now, enriched with the semantic meaning of why it matters. Where the semantic layer captures definitions and the knowledge graph captures facts, the context graph captures state: the agent's current goals, its observations, the relationships between the entities it is reasoning about, and the situational relevance of everything in its environment.

Each generation subsumes the previous one rather than replacing it. A well-designed context graph anchors itself to the canonical facts of a knowledge graph and the governed definitions of a semantic layer. The architectural challenge is building a foundation that can hold all three (static definitions, stable facts, and dynamic state) in a single, unified, transactional system.

This paper describes that transition, the infrastructure it requires, and why a unified multi-model foundation is the only architecture capable of supporting it at production scale.

The 2026 enterprise reality
Hitting the wall

The promise of autonomous AI agents captured enterprise attention throughout 2024 and into 2025. Nearly every organisation with a data team attempted to deploy some form of agentic workflow: a customer support agent, a data analyst, an internal knowledge assistant. Most of those efforts failed.

From pilot to production: where agents are failing

The failure mode was consistent. Agents struggled to answer basic questions correctly. "What was revenue growth last quarter?" produced wrong answers, not because the model could not reason, but because the agent lacked the context to understand how revenue was actually defined in that organisation, which tables were authoritative, and how the definition had changed since a team member left the previous year.

MIT's State of AI in Business research[1] confirmed what practitioners were already experiencing: the majority of AI agent deployments failed due to brittle workflows, lack of contextual learning, and misalignment with day-to-day operations. The problem was not the model. It was the data.

Enterprise data is still deeply fragmented. Even organisations that have invested in data platforms and semantic layers find that those layers go stale, are incomplete, and were never designed to be queried by agents operating at millisecond speed. Context that exists as tribal knowledge in Slack threads, stale YAML files, or undocumented table conventions is invisible to an agent. The context wall is real, and most enterprises have already run into it.

The cost and governance reckoning

A second reality is now pressing on enterprise AI teams: the cost of the current approach. Running a production agentic system on a "frankenstack" of four specialist databases means paying for four sets of infrastructure, managing four data synchronisation pipelines, enforcing access controls across four systems, and debugging failures that span all of them.

Procurement and security teams, initially tolerant of experimental architectures, are now asking harder questions. Can you audit what data an agent used to reach a decision? Can you prove that a user's permissions were correctly enforced across every retrieval operation? Can you justify the infrastructure cost relative to the business value delivered?

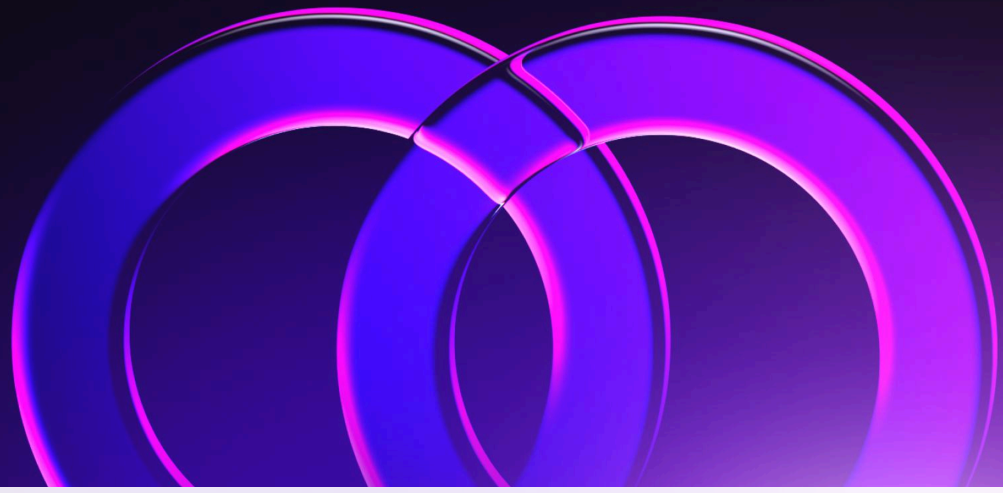
These are not engineering questions. They are governance questions, and the current fragmented approach cannot answer them cleanly.

Why architecture is the bottleneck, not the model

The instinct in 2024 was to wait for better models. Models have improved. The context problem has not. Longer context windows do not solve the fundamental issue: an agent that cannot access reliable, structured, real-time context about its environment will hallucinate, contradict itself, and fail to coordinate with other agents regardless of how capable its underlying model is.

The bottleneck is architectural. The data infrastructure that enterprises built for human analysts is fundamentally unsuited for autonomous agents. Fixing this requires not a new model, but a new foundation.

[1] MIT Nanda The GenAI Divide - State of AI in Business 2025



The architecture of agency
Why AI needs a context layer

The data landscape has changed dramatically over the last decade. Monolithic data warehouses have largely given way to more distributed architectures: data platforms, lakehouse designs, and data mesh patterns built on systems like Databricks, Snowflake, and S3-backed lakes. These architectures solved real problems around scale, cost, and analytical flexibility.

Databricks recently introduced the "Lakebase" architecture, separating compute from storage and placing data in open formats on object storage, with a serverless Postgres layer on top. This is a genuine innovation for developer experience, cost, and analytical workloads. But none of these architectures were designed for what AI agents actually need.

The shift from human-driven analytics to autonomous agency introduces a fundamentally different set of requirements. Humans query data to find insights. Agents query data to decide what to do next. That distinction changes everything about how a data architecture must behave.

The read-think-write loop

An AI agent operates through a continuous cycle: perceive the environment, reason over memory, commit an action or observation back to storage. This loop runs in milliseconds, and it never stops.

Modern data platforms, regardless of how sophisticated, are optimised for analytical throughput. They are designed to answer questions about the past at scale. Fast ingest makes data queryable quickly; it does not provide the ability to lock a record, mutate it, and guarantee that the next agent in the loop reads the updated value within the same millisecond window. That requires read-after-write transactional consistency, a property that streaming ingest architectures are not designed to deliver. The distinction is not any individual latency figure; it is design intent.

The state problem

Agents are not stateless. They track goals, monitor sub-tasks, and maintain a working memory of their progress. When an agent is coordinating a complex workflow, it must be able to lock a resource, update a belief, and trigger a downstream action, all within a single consistent operation.

This requires ACID compliance and granular transactional control. Analytical platforms are not built for this. They are built to tell you what happened. Agents need infrastructure that shapes what happens next.

The missing layer: context

What is missing from every existing data architecture - whether a traditional warehouse, a modern lakehouse, or a distributed data mesh - is a dedicated context layer. A context layer is an active, low-latency, transactional substrate that holds the live state of an agent's understanding: what it knows right now, what it is currently doing, and how that relates to everything else in the system.

This layer must be capable of being written to at high frequency, read with consistent low latency, and structured in a way that reflects relationships, not just records. It must also push updates to agents rather than waiting to be polled. The context layer is where autonomous AI actually lives. Without it, agents are reasoning in the dark. This is precisely the question that a16z posed in March 2026, asking where the context layer will live and whether it will become its own standalone product. SurrealDB's answer was that it must live at the database layer, not above it, because only at the database layer can context be kept consistent, governed, secured, and transactionally unified with the canonical knowledge that grounds it.

How context gets built

Context enters the system through multiple channels: automated pipelines that ingest canonical data from existing warehouses and semantic layers, agent-generated observations written back during the read-think-write loop, human-curated business definitions and rules contributed by domain experts, and self-updating flows where agent corrections are incorporated back into the context graph.

A multi-model foundation supports all of these ingestion patterns natively. Canonical facts arrive as structured documents. Relationships between entities are expressed as graph edges. Semantic embeddings are stored alongside the data they describe. Because all three models exist in the same transactional system, an update to a business definition, its relationships, and its embedding can be committed atomically, eliminating the synchronisation gaps that plague fragmented architectures.

The context layer is not a static artefact. It is a living corpus, continuously refined by both humans and agents, that must be as easy to update as it is to query.

The context layer
Why multi-model wins

The transition from **Large Language Models** (LLMs) to **Autonomous Agents** represents a shift from "stateless inference" to "stateful reasoning." For an agent to be effective, it requires more than a temporary window of text; it requires a persistent, high-fidelity memory. In many current architectures, this memory is fragmented across a "frankenstack" of specialised databases - a vector store for similarity, a graph database for relationships, and a relational or document store for metadata.

This fragmentation imposes a significant **"Cognitive Tax"** on the agent and a **"Semantic Tax"** on the developer.

The fragmented memory tax

When an agent's memory is split across multiple systems, the "unification" of that data happens at the application layer. This creates several architectural bottlenecks:

The latency of reconstruction

Every time an agent needs to "remember" something, it must make multiple asynchronous calls to different APIs, wait for the results, and then use application logic to join that data together. In an autonomous loop, these milliseconds compound into seconds of "thinking" time.

Semantic drift

When a document is updated in a NoSQL store, but its corresponding vector embedding in a separate store isn't refreshed, or its relationship in a graph database remains stale, the agent begins to reason over contradictory information.

Loss of dimensionality

A standalone vector search can tell an agent that "Product A is similar to Product B," but it cannot easily explain why (the relationship) or show the structured constraints (the price, the stock, the owner) without a complex multi-stage join.

Why memory middleware is not enough

A category of products has emerged to address agent memory at the application layer. Tools such as Mem0 provide a managed memory abstraction: extract facts from conversations, store them, retrieve them later. They make it easy to add persistence to an LLM application with a few lines of code, and they have found real traction, particularly for personalisation and conversational continuity.

However, application-layer memory middleware has a structural ceiling. These systems are typically built on top of the same fragmented infrastructure: a vector store for semantic retrieval, optionally a graph layer for relationship tracking, and a relational store for metadata. The abstraction hides the complexity from the developer, but it does not eliminate it. When context changes in one underlying system, synchronisation to others is not guaranteed. The semantic drift problem persists underneath the abstraction.

More fundamentally, memory middleware treats context as something to be retrieved and injected into a prompt. It does not treat context as a live, transactional substrate that the agent reasons within. For autonomous agents co-ordinating complex workflows with enterprise governance requirements, the foundation must be deeper.

The multi-model advantage: atomic context

A native multi-model foundation - where the database engine treats documents, graphs, and vectors as first-class citizens of the same storage layer - solves the memory problem by providing **Atomic Context.**

In this architecture, a single "memory" is not a collection of pointers across different systems; it is a unified record.

Feature	The fragmented approach (status quo)	The multi-model approach (SurrealDB)
DATA RETRIEVAL	Multiple round-trips to Vector, Graph, and Doc stores.	A single query traversing relationships and vectors.
DATA INTEGRITY	Manual synchronisation (risking semantic drift).	ACID-compliant updates across all models.
AGENT LOGIC	Complex "glue code" to merge disparate data types.	"Context-native" queries (e.g. via SurrealQL).
SCALING	Sharding and managing three different infrastructures.	Horizontal scaling via distributed deployment.

From "pointers" to "rich edges"

In a traditional graph database, an edge is often a simple line between two nodes. In a multi-model semantic foundation, edges are documents. An agent can traverse a relationship and find, embedded directly within that link, the vector embedding of the last interaction, the JSON metadata of the permissioning logic, and the relational history of the connection. This allows the agent to fetch a "complete thought" in one operation.

By eliminating the boundary between the "what" (document), the "how it relates" (graph), and the "what it's like" (vector), we provide agents with a memory layer that mirrors human cognition: interconnected, multidimensional, and instantly accessible.

The missing layer
Context graph support

To understand why traditional Knowledge Graphs (KGs) are failing modern AI agents, we must distinguish between Canonical Knowledge and Dynamic Context.

Most enterprises have spent the last decade building Knowledge Graphs as "dictionaries of truth." They are excellent at defining what a company knows, but they are remarkably poor at helping an agent decide what to do now.

Knowledge vs. experience: the static and the fluid

A **knowledge graph** is the map; a **context graph** is the journey.

The knowledge graph (the map)

Contains stable, ontological facts. *"Product X is a subscription service," "User Y lives in New York," "London is a city."* These are high-trust, low-velocity data points.

The context graph (the journey)

Contains ephemeral, situational relevance. "The agent is currently troubleshooting Product X for User Y, who is frustrated because of a recent billing error." For an AI agent, the "truth" of the Knowledge Graph is useless without the "relevance" of the Context Graph. If an agent only has access to a static KG, it suffers from a form of digital amnesia. It knows who the user is, but it forgets why they are talking.

Why native graph support matters for agency

However, agents do not just need to find similar text; they need to understand influence, proximity, and impact. Modern vector databases such as Pinecone, Chroma, and Weaviate now support metadata filtering, hybrid search and namespace scoping.These are genuine improvements. However, they remain fundamentally flat: they can tell an agent that two things are semantically similar, but they cannot model the structured relationships between entities, enforce transactional consistency across updates, or traverse impact paths through a connected graph. Similarity is not context.

Consider a concrete example. An agent monitoring a production system receives an alert about a failing API endpoint. With a vector database, the agent can search for documentation semantically similar to the error message. Useful, but shallow. With a graph-capable multi-model database, the agent can traverse from the failing endpoint to the upstream service that depends on it, to the team that owns that service, to the on-call engineer currently assigned, and simultaneously retrieve the vector-embedded runbook most relevant to this specific failure mode, all in a single query. The structural traversal and the semantic search are not two separate operations stitched together in application code; they are one atomic operation. No vector database, however sophisticated its metadata filtering, can perform this kind of connected reasoning.

A graph-native multi-model foundation allows an agent to perform "impact traversals" in real time:

Relationship over similarity

A vector search might find a manual about "Server Errors." A graph-native query finds the *specific* server that is down, the *upstream* services it impacts, and the *on-call engineer* currently assigned to it.

The impact radius

When an event occurs, an agent can "walk the graph" to see how far the ripples go. This is a structural calculation that flat data structures cannot perform efficiently.

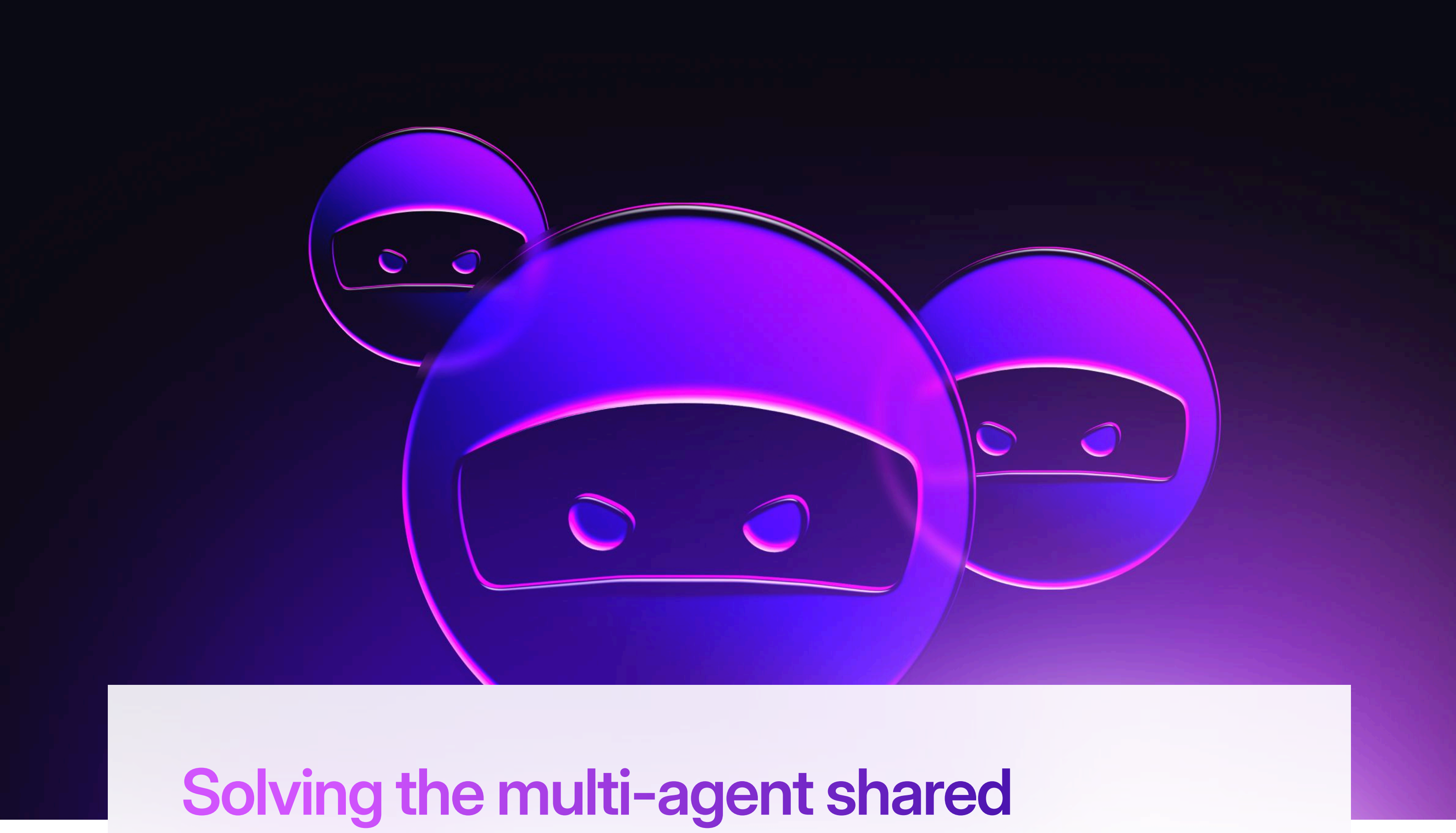
Filtered reasoning

By using the graph as a native layer, agents can prune their search space. Instead of searching "all documents," the agent follows the edges: "Search only the documents related to the specific version of the software this user is running."

The real-time contextual loop

In an agentic system, the context graph is in a state of constant flux. Every observation the agent makes is a new node; every action is a new edge. If the underlying database cannot handle these high-frequency updates without cache invalidation or performance degradation, the agent’s "short-term memory" becomes a bottleneck.

The missing layer in Agentic AI isn't just "more data" - it is a high-velocity context graph that lives on top of a stable knowledge graph, unified within a single, multi-model foundation.



Solving the multi-agent shared state problem

As AI moves from single-task assistants to Multi-Agent Systems (MAS), the architectural challenge shifts from individual memory to shared state. When multiple agents - each with specific roles like "Researcher," "Coder," or "Reviewer" - work together, they cannot operate in silos. They require a "Blackboard," a common area where the current state of a task is visible, verifiable, and actionable by all participants.

The "blackboard" pattern vs. message passing

In many early multi-agent frameworks, agents communicate via direct message passing. While simple, this creates a "telephone game" effect: context is lost, versions diverge, and the "source of truth" becomes a moving target.

A **multi-model database** acts as the central blackboard. Instead of agents telling each other what they found, they **commit** their findings to the shared context graph.

Concurrency control

When two agents attempt to update the same sub-task, the database provides the ACID guarantees necessary to prevent "hallucinated conflicts."

Global visibility

A "supervisor" agent can query the database to see the real-time progress of all sub-agents without needing to poll them individually.

Why AI agents need a database, not a data platform

There is a common misconception that LLMs can simply "query the platform" for context. While analytical platforms are excellent for historical analysis, they are poorly suited for agentic state management.

The latency gap

Analytical platforms are optimised for massive, high-latency scans. Agents operate in "loops" - they need to read a state, make a decision, and write a result in milliseconds. A warehouse "commit" can take seconds or minutes to be visible to other agents.

Transactional integrity

Agents are often performing actions that have real-world consequences (e.g., booking a flight, moving a file, updating a budget). These require transactional locking. If an agent is modifying a "Task" node in a Graph, no other agent should be able to move it until the first operation is complete.

The "live" requirement

In a multi-agent environment, agents shouldn't have to keep asking "Is it done yet?" A modern database supports live queries (or Change Data Capture). When the "Coder" agent finishes a script and writes it to the database, the "Reviewer" agent is instantly notified by the database itself.

Managing the "agentic race condition"

Without a robust database layer, multi-agent systems fall victim to the "Agentic Race Condition" - where Agent A acts on information that Agent B is currently changing. By using a native multi-model database, the graph maintains the relationships between tasks, the document stores the content of the work.

ACID transactions with snapshot isolation ensure each agent reasons over a consistent view of the database, and committed writes from one agent are immediately visible to subsequent transactions from other agents. There is no synchronisation gap across which a race condition can form. The result is a system that does not just "talk" more, but "knows" more, collectively.

The competitive landscape
What else is being built

The market has recognised the context problem. In early 2026, a16z published research² noting that enterprises are “hitting the wall” and that a new category of "context layer" solution is emerging. Several architectures are competing to answer the question of where this context layer should live. Understanding the distinctions matters for any organisation making infrastructure decisions.

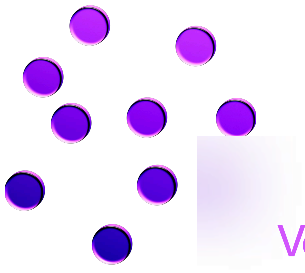
Memory middleware: Mem0 and the abstraction layer approach

Mem0 is the most widely deployed product in the adjacent "AI memory" space, with over 80,000 developers on its cloud platform. Its approach is to provide a managed memory abstraction layer over existing storage infrastructure: extract facts from conversations, store them, retrieve them when relevant. It integrates with OpenAI, LangChain, CrewAI, and other frameworks, and was selected as the memory provider for AWS's Strands Agents SDK.

The architectural limitation is that memory middleware sits above the storage layer rather than within it. The semantic drift problem is not eliminated; it is managed. More fundamentally, treating memory as a retrieval layer is a different paradigm from treating memory as a live operational substrate that an agent reasons within.

Document databases: MongoDB Atlas

MongoDB has added vector search and graph traversal capabilities to its document-first platform. The key distinction is that MongoDB's multi-model capabilities are additive rather than architecturally unified. Vector search runs on a separate `mongot` process that is not part of the same ACID transaction as document writes. For agentic workloads where a single agent action may simultaneously update a relationship, write an observation document, and update a vector embedding, it creates the same consistency risks as a fully fragmented stack.



Vector databases: Pinecone and Chroma

Both are significantly more capable in 2026 than when RAG architectures first emerged. Both support metadata filtering, hybrid search, sparse-dense retrieval, and namespace-based isolation. Yet no vector database, however sophisticated, can provide what agents fundamentally need: structured relationship traversal, transactional consistency across model types, and the ability to model impact, causality, and proximity rather than just semantic similarity. These are architectural properties, not feature gaps that an additional index type can close.

Data platforms: Databricks Lakebase and Snowflake

Databricks Lakebase represents a genuinely new database architecture: compute separated from storage, data stored in open formats on object storage, serverless Postgres with instant branching and elastic scaling. Snowflake is pursuing a similar direction. Their data gravity (the concentration of enterprise data already on these platforms) is a meaningful advantage that should not be understated. Any new infrastructure must integrate with these existing investments, not demand that enterprises abandon them.

The distinction is design intent. These platforms are not optimised for the high-frequency transactional read-think-write loop of an autonomous agent, the rich edge model that allows an agent to traverse a relationship and find an embedded vector in the same operation, or the record-level permission model that enforces consistent access control across graph, vector, and document operations in a single query. The context layer and the analytical platform are complementary infrastructure, not competitors.

Why the foundation matters more than the layer

SurrealDB occupies a distinct position. The context layer does not live above the database as a middleware abstraction, nor does it live inside an analytical platform as a bolt-on feature. It lives in the database itself, as a native property of a storage model that treats graph, vector, and document as first-class citizens of the same transactional system. This is not a feature decision. It is an architectural one, and it determines what is possible at production scale.

[2] https://a16z.com/your-data-agents-need-context/

Towards an Enterprise Semantic Foundation

As organisations move from experimental AI "sandboxes" to production-grade agentic workflows, they encounter a final, formidable barrier: semantic drift. In a fragmented data landscape, different departments, applications, and AI agents often operate on different definitions of the same entity. An "active customer" to a marketing agent might mean something entirely different to a billing agent.

The Enterprise Semantic Foundation (ESF) is the architectural response to this chaos. It is not a single database or a rigid schema; it is a unified, multi-model substrate that serves as the common language for the entire organisation.

Bridging the gap: stability meets adaptability

Anchoring Canonical Meaning

The Foundation utilises the **Knowledge Graph** to house high-integrity, governed facts - the "Gold Standard" data that has passed through traditional ETL and quality checks.

Activating real-time meaning

Simultaneously, the **Context Graph** captures the high-velocity, "lived experience" of AI agents. Because these exist in the same multi-model system, the context is always anchored to the canonical truth. There is no "translation layer" where meaning can be lost or distorted.

Solving the security and governance paradox

In a traditional "frankenstack" (vector store + graph store + document store), enforcing security is an operational nightmare. If a user's permissions change, those changes must be propagated and synchronised across three or four different systems simultaneously. If the synchronisation lags, an AI agent might inadvertently "remember" or "reason" over sensitive data it is no longer authorised to see.

An Enterprise Semantic Foundation solves this through record-level and field-level permissions native to the database:

Unified access control

Permissions are defined once at the data layer. Whether an agent is performing a vector similarity search, a graph traversal, or a document lookup, the database enforces the same security constraints.

Explainable AI

Because the relationships (edges) are first-class citizens, the path an agent took to reach a conclusion is traceable. You can query the graph to see exactly which "contextual edge" led to a specific decision, providing a level of auditability that flat vector stores simply cannot offer.

The end of semantic drift

By moving semantics from the application code (where it is hidden and fragmented) into the database (where it is transparent and shared), organisations create a "self-describing" data environment. The database doesn't just store bytes; it stores intent. When a new agent is deployed, it doesn't need to be "trained" on how to interpret the data; it simply queries the Enterprise Semantic Foundation to understand the relationships, constraints, and context inherent in the system.

Integration, not replacement

A production context layer cannot exist in isolation. Enterprise data lives in Snowflake, Databricks, S3, and dozens of operational systems. The context layer must sit alongside these investments, not demand that they be replaced.

SurrealDB is designed to complement existing data platforms. Canonical data from warehouses and lakehouses flows into the context graph through standard ingestion pipelines. Agents query the context layer for real-time state and relationships, while analytical workloads remain on the platforms best suited for them. The result is a clear separation of concerns: the data platform answers "what happened," and the context layer answers "what should happen next."

This is not a rip-and-replace proposition. It is an architectural addition: a transactional, multi-model layer purpose-built for the operational requirements that analytical platforms were never designed to meet.

Developer experience and time-to-value

Enterprise adoption depends not only on architectural soundness but on practical accessibility. SurrealDB is designed to reduce the distance between concept and deployment:

Enterprise adoption depends not only on architectural soundness but on practical accessibility. SurrealDB is designed to reduce the distance between concept and deployment:

Single query language

SurrealQL handles document operations, graph traversals, vector search, and access control in one language, eliminating the need to learn and integrate multiple query syntaxes.

Flexible schema

Tables can operate in schemaless mode for rapid prototyping, then transition to schemafull mode as the data model stabilises, without migration overhead.

Embedded to distributed

The same SurrealDB binary runs as an embedded library, a single-node server, or a distributed cluster backed by TiKV, allowing teams to start small and scale without re-architecture.

Native SDKs and protocol support

SDKs for Rust, JavaScript, Python, Go, Java, and .NET, along with HTTP and WebSocket APIs, allow integration with any existing agent framework.

The goal is to make the context layer as easy to adopt as it is powerful to operate.

Conclusion
The architecture of the agentic era

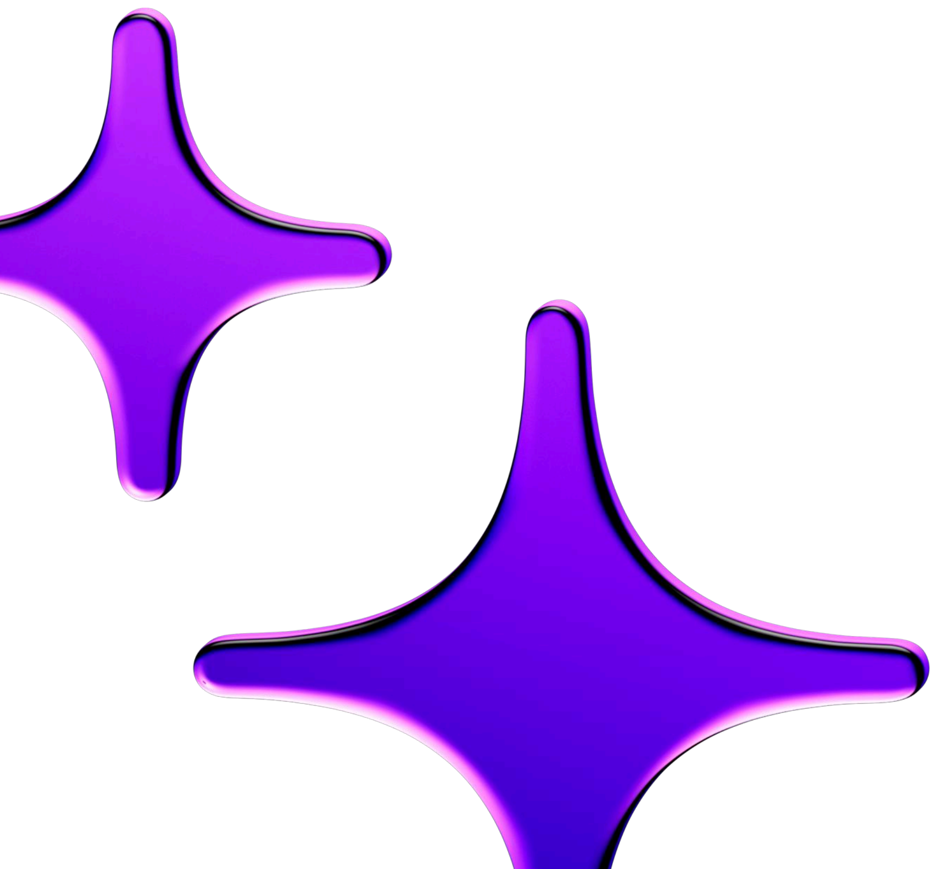
The first generation of graph databases solved the problem of interconnected data for human analysts. The next generation must solve the problem of interconnected reasoning for AI agents.

The leap from simple RAG to true autonomous agency requires a move away from static data warehouses and fragmented "specialty" stores and analytical platforms that were never designed for real-time agentic state. It demands a multi-model **Enterprise Semantic Foundation** - a system where the graph provides the structure, the document provides the substance, and the vector provides the intuition.

The market is beginning to understand this. Specialist databases are addressing parts of the problem. Memory middleware is abstracting over the fragmentation. Data platforms are building context features on top of existing products. Each of these efforts is moving the industry forward. But none of them solve the problem at the foundation level, where graph, vector, and document exist as a single unified transactional model, governed by a single permission system, queryable in a single language, and consistent within a single transaction.

SurrealDB was built specifically for this transition. By providing a native multi-model environment that scales horizontally and handles real-time updates without fragility, it allows enterprises to stop building "glue code" and start building the future of AI memory.

The context layer will live in the database.
The question is which database is ready for it.



Technical appendix

Implementing semantic context with SurrealDB

To illustrate how a multi-model foundation simplifies agentic workflows, we can look at how SurrealQL collapses complex operations into single, atomic queries.

A. The "Rich Edge" (Graph + Document)

Instead of just a pointer, an edge in SurrealDB can store the full context of an interaction, including the vector embedding of the sentiment.

```
-- Relate an agent to a user with a 'memory' edge containing metadata and vectors
RELATE agent:researcher->interacts_with->user:customer123
SET
    time = time::now(),
    topic = "Billing Dispute",
    sentiment_score = 0.2,
    -- Store the embedding for semantic retrieval
    context_embedding = [0.12, 0.45, -0.31, ...],
    summary = "User frustrated regarding double-billing on invoice #55";
```

B. The unified search (Vector + Graph)

An agent can find "Similar Memories" (Vector) but immediately filter them by "Established"

```
-- Required: define an HNSW index on the embedding field first
DEFINE INDEX ctx_emb_idx ON interacts_with FIELDS context_embedding HNSW DIMENSION 384 DIST COSINE;

-- Find interactions similar to current context, but only for a specific product
-- Uses HNSW approximate search: k=2 results, ef_search=40
SELECT * FROM interacts_with
WHERE context_embedding <|2,40|> [0.12, 0.45, ...]
  AND out.purchased_product = product:enterprise_saas
  AND time > time::now() - 30d;
```

C. Live Queries for multi-agent sync

Instead of polling a warehouse, agents subscribe to changes in the "Shared State."

```
-- A 'Supervisor' agent listens for any sub-task being marked as 'Completed'
LIVE SELECT * FROM task WHERE status = 'Completed' AND project = project:ai_migration;
```