A MAGAZINE BY MONGODB

02

Database Digest

ISSUE 02

The Unified Intelligence Layer: Powering the Agentic Era

Published since 2025
by MongoDB,
1633 Broadway, 38th Floor
New York, NY 10019, USA

Contributing Editors:
Boris Bialek, Sebastian Rojas Arbulu, Daniel Jamir

Brand Creative:
Andrew Considine, Hans Bennewitz

Acknowledgements

Industry Solutions: Taylor Hedgecock, Gabriela Preiss, Genevieve Broadhead, Peyman Parsi, Humza Akhtar, Francesc Mateu, Jeff Needham, Benjamin Lorenz, Thorsten Walther, Rama Gabbita, Ken Wiebke, Diego Albano, Ainhoa Múgica, Raphael Schor

Developer Relations: Apoorva Joshi, Franck Pachot, Mikiko Bazeley

Education: Raghu Viswanathan, Rita Rodrigues, Thalita Carrico, Emily Pope

Product Marketing: Jad Jarouche

Global Corporate Marketing: Joanne Johnson, Laila McDonald, Kevin O'Rouke

Solutions Architect: Diana Annie Jenosh, Ashwin Gangadhar

International Marketing: May Petry, Penta Stanley, Cathy Gallego, Ilana Matro

Global Demand Center: Steve Jurczak, Jack Yallop, Brandon Lee

Letter from the Editor

Boris Bialek
Vice President,
Industry Solutions
at MongoDB

In 35 years in this business, I have never seen anything quite like the present moment. Usually a few industries make big moves while others watch, wondering if they need to act. Not this time. Banks, automotive, insurance, manufacturing, and more are all moving at once, and most of them, as one customer put it to me, are betting on the whole racetrack.

That has led to some strange architecture decisions. Earlier this year, I sat with a client whose proof-of-concept for a single, narrow-agent workflow involved 14 different vendors stitched together. He was rather proud of it. I asked him, gently, how he planned to put it into production. He looked at me as if the question hadn't occurred to him, and said, "We haven't really got to that part yet." I showed him a competitor who had and was already live in production after eight weeks. He stopped talking.

That small private moment is, to my mind, the defining moment of agentic AI in 2026: a smart team realizing they have built a beautiful demo that can't be turned into a business.

When our president and chief executive officer, CJ Desai, stood up at MongoDB.local London in May 2026 and said, "The hardest part of running agents in production isn't the model. It's the data layer underneath it." Nobody in the room needed further explanation.

He had reason to be blunt. Gartner expects 40% of agentic AI projects to be canceled by 2027. Deloitte finds just 11% of enterprises running agentic systems in production, meaning 89% are still stuck short of it. The reason, almost without exception, is not the model. It is everything beneath it: Security bolted on at the end, audit trails nobody planned for, real-time data that lives in four systems at once and arrives at the agent half a step too late.

Consider the alternative. The largest pharmaceutical distributor in the United States moves 1.2 billion containers per year, and behind each of those containers is a person: a patient, waiting for the right drug in the right dose on the right day. The Drug Supply Chain Security Act does not care how clever your model is. It cares whether you can trace any container, end to end, in real time, every time. A few years ago McKesson made a quiet, unglamorous, architectural choice: to retire a monolithic legacy stack and run on a unified data platform instead. They scaled their operations 300-fold. They are still trusted with patient safety. That trust was not won by an agent. It was won by what sits underneath it.

We make the case for needing that solid base throughout the rest of the issue. Section 01 measures what fragmentation actually costs, and what McKesson built instead. Section 02 turns to what replaces fragmentation: the unified intelligence layer, shown across the industries where it pays back first. Section 03 asks how to make the agents on top of your foundation actually accurate, with LG U+ offering a glimpse of what that looks like in production. In section 04, we reach the climax: the shift from systems of record to systems of action.

I'll close with a moment that has stayed with me. On the day his team went live, the CTO of a very large retailer told me the project we had built together was obsolete. I was, for a moment, heartbroken. Then he explained. "Boris, it's obsolete because it's live," he said. "But the data was MongoDB. That is my foundation. I can replace the LLM and the model at any time. I can replace the framework on top. The data is mine."

What we've learned from making these big moves is that the architectural choices made now will outlast the AI tools they were chosen for. That, in the end, is why the data layer is the decision worth getting right.

Table of Contents

SECTION ONE

When AI Outruns the Stack

The hidden cost of fragmented architecture.

p. 6

SECTION TWO

The Unified Intelligence Layer

A single platform for the full retrieval lifecycle.

p. 15

SECTION THREE

Accuracy at Scale

Grounding AI agents in real-time operational truth.

p. 23

SECTION FOUR

Building the AI Era

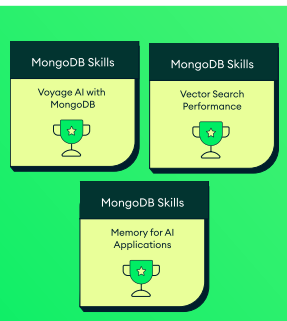
From systems of record to systems of action.

p. 31

FEATURED BOOK

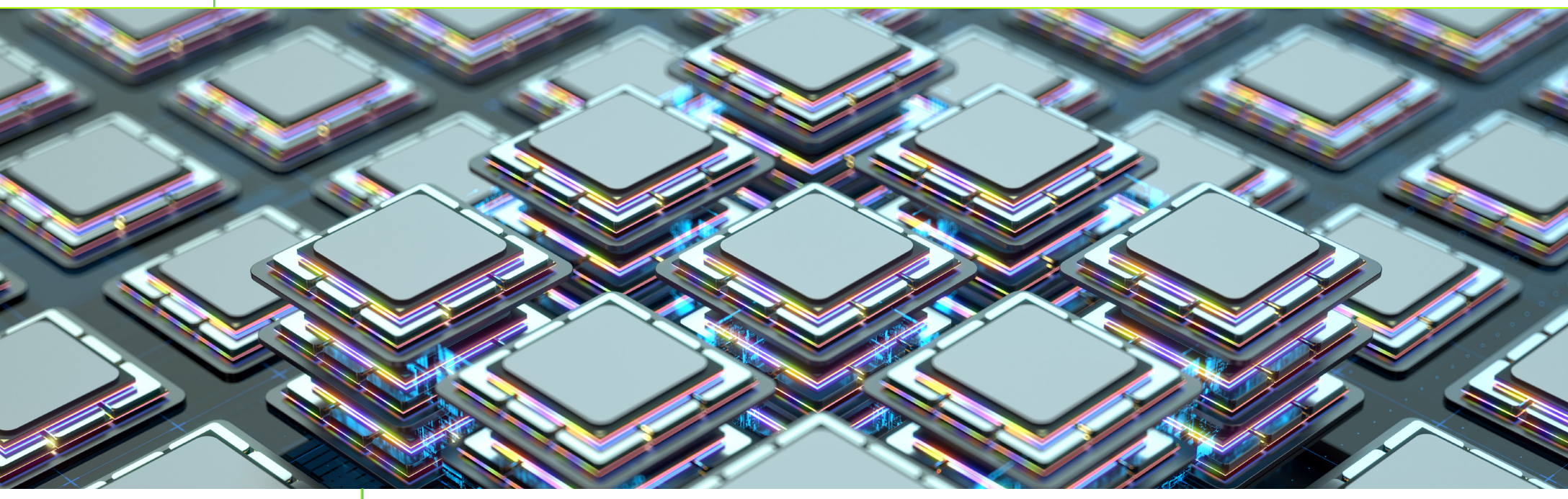
Architectures for the Intelligent AI-Ready Enterprise

Practical frameworks for building AI-ready architectures that deliver lasting business value.



LEARNING & CERTIFICATION

Three new MongoDB AI Skill Badges: Voyage AI with MongoDB, Vector Search Performance, and Memory for AI Applications.



SECTION ONE

When AI outruns the stack

That gap has a name now. Call it the ***architectural drag***, the cumulative weight a fragmented stack puts on every team trying to ship AI on top of it. New research from International Data Corporation (IDC), commissioned by MongoDB and conducted across 1,400 organizations in eight Asia-Pacific markets, puts a number on it. Forty-three percent of organizations say their existing architecture is a major obstacle to building new applications without extensive modernization. The systems are too rigid, too costly, too slow. And the cost of doing nothing compounds: IDC predicts that organizations that fail to address technical debt will face a 50% higher AI failure rate by 2027.

The hidden cost of fragmented architecture

Two enterprises start the same AI project on the same day.

By the end of the quarter, one of them has shipped a working agent in production. It's grounded in their company's operational data. It has memory across sessions. It passed the security review. The cost is predictable.

Eighteen months later, the second one remains in pilot. The agent works in the demo environment. It hallucinates in staging. The vector store and the operational store have started to drift–and nobody is sure who owns that problem. Procurement is asking whether the cloud bill is supposed to look like this. Legal is asking how to audit something no one can fully describe.

Same talent. Same models. Same budget. The only variable is the architecture they walked in with.

That failure rate isn't a model-quality problem. It's what happens when agents meet a stack that was assembled to retrieve, not to act. A chatbot reads. An agent reads, decides, writes, transacts, and changes state–and then writes some more, because every decision an agent makes leaves a trail. Decision logs. Intermediate states. Audit records. Agent memory that has to survive the next session. The drag is what happens when a stack assembled to answer questions is asked to do all of that at once.

That hits in four places at the same time. The vector store can't write, but the agent has to change state. The operational store and the vector store can't share a transaction, but the agent's action has to be atomic or it half-completes: the

customer is charged, the order isn't booked. The sync between them lags, and the agent decides on yesterday's truth. And when a regulator asks what the agent did, the audit trail covers four of the six steps because no one system saw the whole sequence.

The gap between the two enterprises isn't talent or budget. It's that one of them modernized the data layer while building the agent. The other tried to do it after, and lost a year to integration work that should have happened first.

It doesn't show up as a line item. It shows up as a slower roadmap, more painful security reviews, and the AI projects that quietly disappear from the next quarterly review.

> "The business wants to move fast, but the systems underneath say no. Something that should take two weeks takes six months."

Thorsten Walther *Managing Director, CXO Advisory Asia at MongoDB*

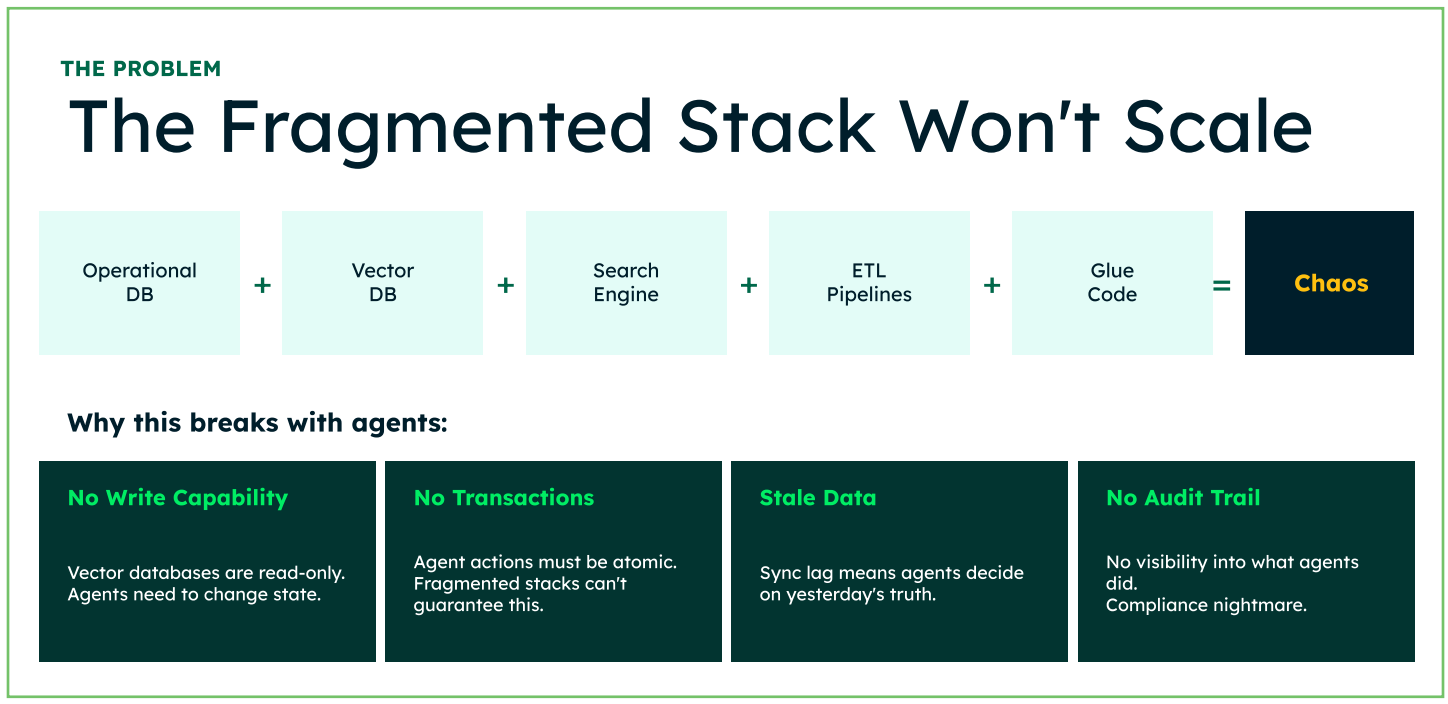


Figure 1.
The fragmented stack and why it breaks with agents

READ THE IDC RESEARCH

The rest of this section works through the problem in two parts.

First, the diagnosis. Why split architectures–separate vector and operational stores stitched together with extract, transform, and load (ETL) pipelines–impose a cost that only shows up at scale. Why the same mismatch shows up at the data model level whenever document workloads are forced into relational engines. And an honest decision framework for choosing between Postgres and MongoDB, from an architect who's spent 30 years on both sides of that debate.

Then an example of one company that addressed it. McKesson moves 1.2 billion drug containers a year and decided early that the data layer was the part of the stack worth getting right. What they built, and what changed because they built it, is the case study that closes the issue.

ARCHITECTURE

Strategic database architecture for AI: unified vs. split

Every architecture review for an agentic AI project ends up at the same question: can the stack we already have support what we're about to build? It's tempting to answer it by benchmarking vector databases. The deeper question is whether your architecture can answer questions about data and meaning in the same query, with the same guarantees.

That's what an agent actually needs. It's the part of the stack most teams settle last, after the rest of the system has already taken shape. The operational database is treated as a given. According to Harvard Business Review Analytic Services, only 15% of organizations consider their data foundation ready for agentic AI. The result is a split architecture that looks reasonable on a whiteboard and starts to fall apart in production.

The two patterns are worth comparing honestly. In a unified architecture, a single platform handles operational data and vector search together. In a split architecture, a dedicated vector store (Pinecone, Weaviate, or a search engine such as Elasticsearch) sits alongside the operational database, with an ETL pipeline keeping the two in sync. A common pairing, shown in Figure 2, is MongoDB as the system of record for product documents and Elasticsearch as the vector search engine for the embeddings. Both work in a demo. The differences only show up when you scale.

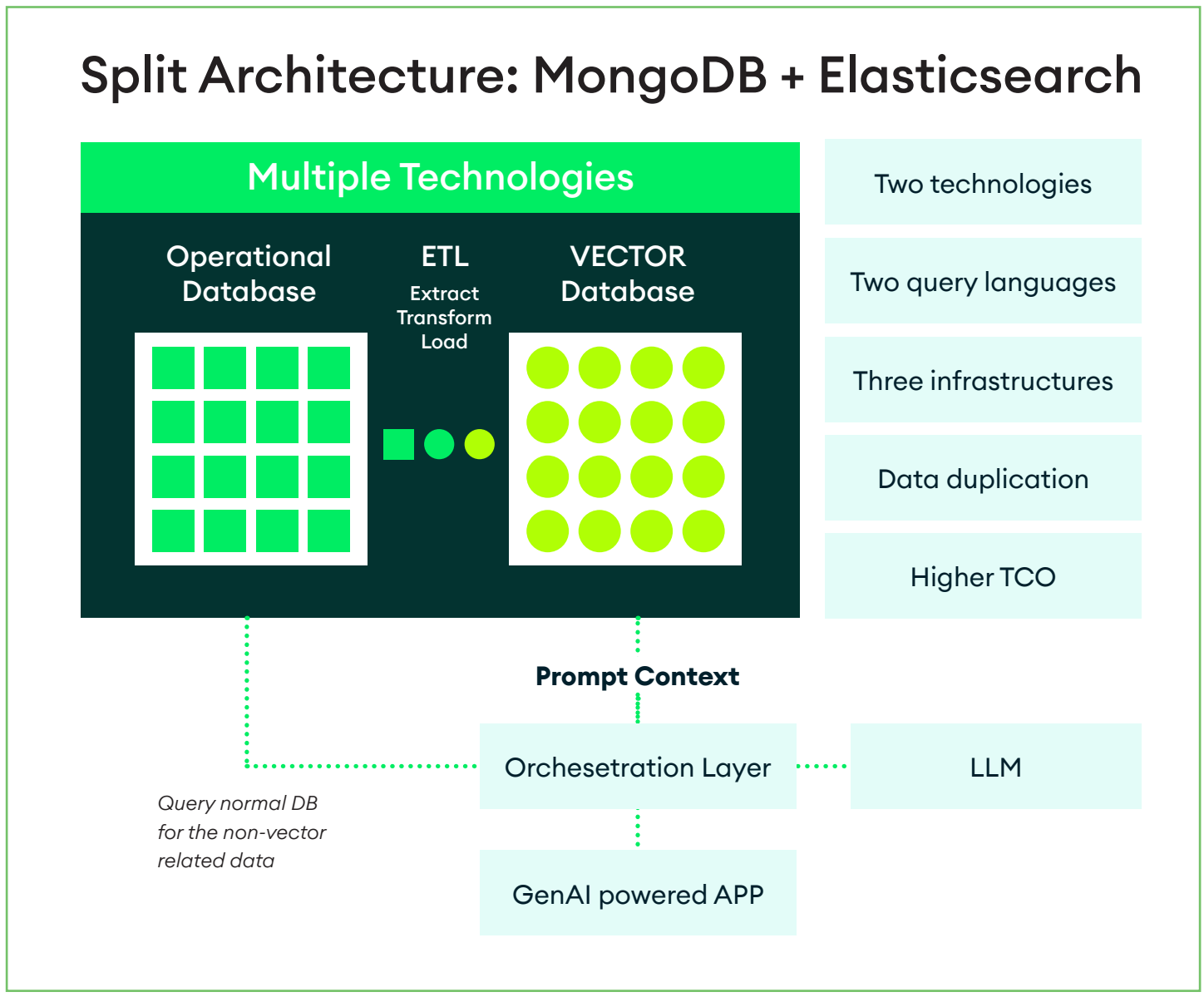


Figure 2.
Split architecture: MongoDB operational database + Elasticsearch vector store

The real failure modes of the split approach are predictable. Synchronization complexity grows with every new field. When a document is deleted in the operational store, the corresponding vector embedding can linger as a ghost document, returning results that no longer exist. Atomic transactions aren't an option because two systems can't share a transaction boundary.

The operational costs compound just as quietly. Developer velocity suffers because most of the engineering effort goes into reconciliation code rather than features. Separate backup strategies, separate failover procedures, separate monitoring stacks, and separate on-call rotations. And the bill arrives every month: two database technologies, two query languages, three pieces of infrastructure to secure and pay for, duplicated data, and a higher total cost of ownership.

The unified case isn't abstract. When the document and its embedding live together, an update is one atomic write, a delete removes both, and the same access controls protect both. Platforms such as MongoDB Atlas that take the unified approach use dedicated search nodes to isolate vector workloads from the operational tier so they can scale independently. Unification doesn't mean giving up the performance headroom that vector workloads demand. Total cost of ownership reflects that simplicity: one platform to pay for, one platform to secure, one platform to operate.

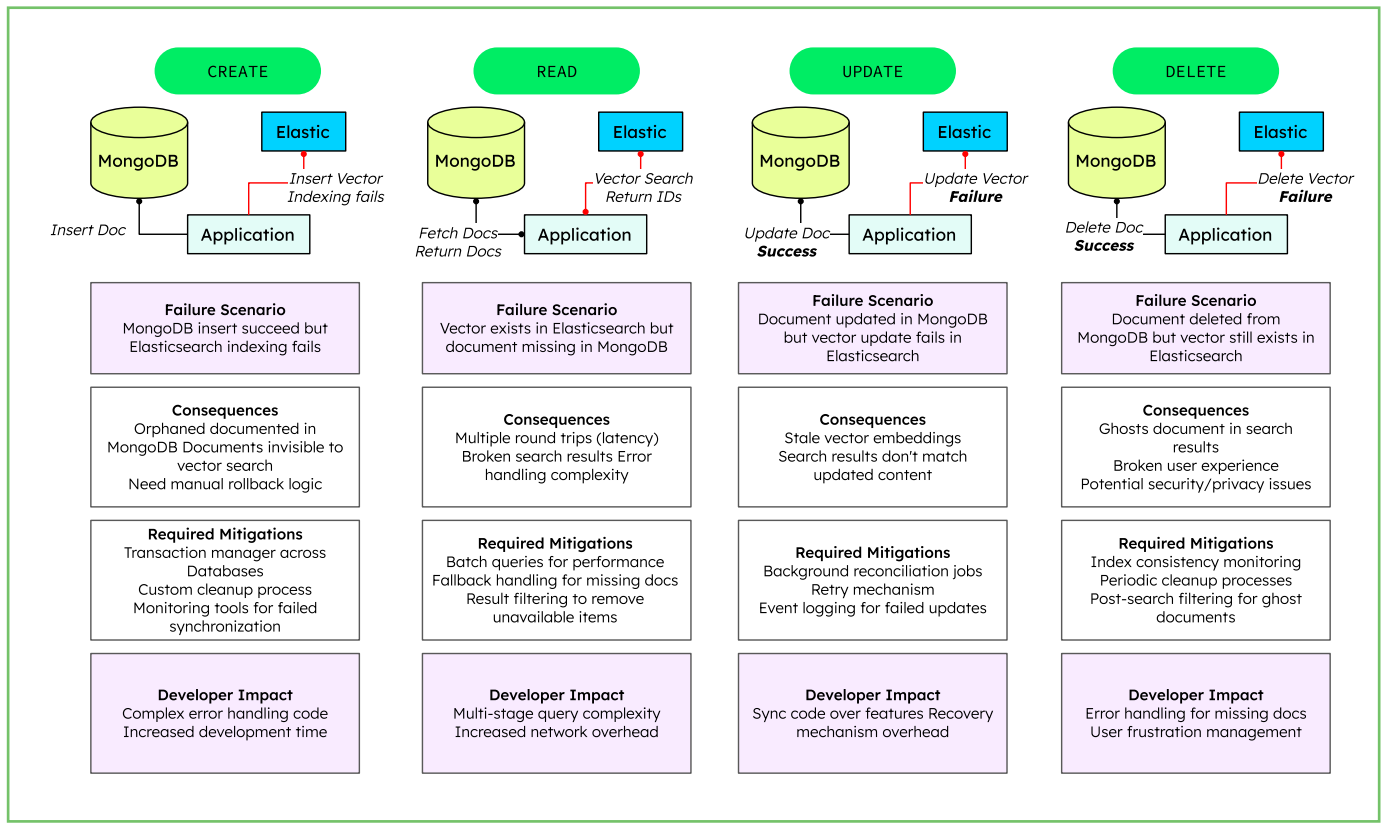


Figure 3.
Split Architecture: The hidden cost.

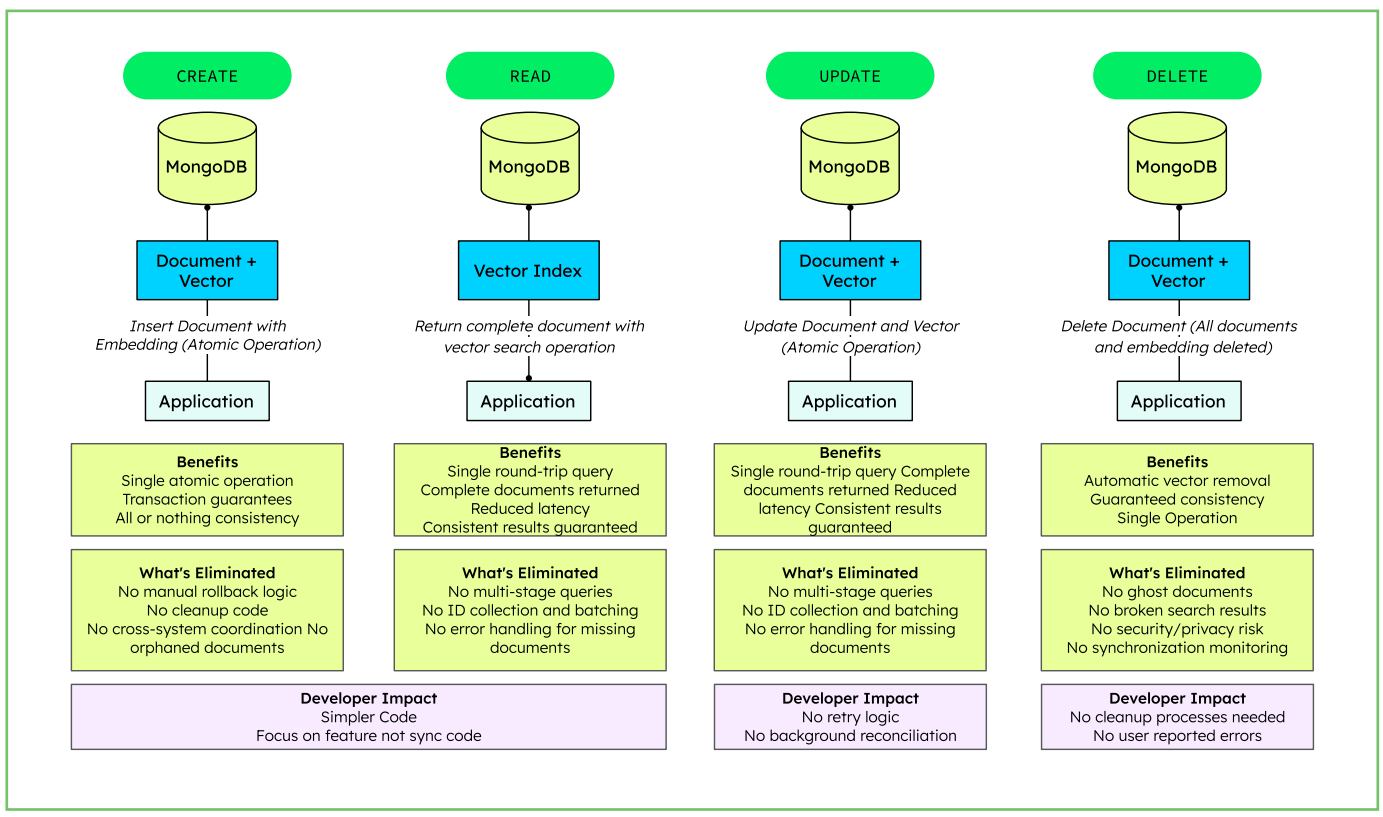


Figure 4.
CRUD operations in a unified architecture: MongoDB Atlas with vector search.

READ THE ARTICLE

MongoDB vs. SQL: Beyond the myths

The modern web stack speaks one language: JSON (JavaScript Object Notation), the light-weight text format browsers, APIs, and mobile apps use to exchange data. The browser sends JSON. The API processes JavaScript objects. The mobile app expects JSON. Message queues, Web-Sockets, microservices: all JSON. Pick a database whose data model matches the rest of the stack, and an entire category of work disappears. Pick one that doesn't, and your team spends its career translating between formats. That's the case Tim Carter Clausen, who writes as The Decipherist and has run MongoDB in production for a decade, makes in a piece worth reading in full.

Every few months a familiar blog post goes viral: "Why we moved back to Postgres." The comment threads fill with the same talking points. *MongoDB doesn't scale. There are no joins. The transactions are weak.* Clausen's response isn't to rehearse the usual both-databases-have-their-strengths fence-sitting. It's to address each criticism in turn, with 10 years of production data behind him.

Two themes will resonate with technical leaders weighing a stack decision in 2026.

The first is the JSON pipeline. A typical SQL request makes eight format changes on a single round trip:

1. The client sends JSON.
2. The API parses it into a JavaScript object.
3. An object-relational mapping (ORM) tool decomposes that object into rows scattered across normalized tables.
4. The database does its work.
5. The rows are reassembled.
6. They're mapped back into an object.
7. The object is serialized back to JSON.
8. The response is sent.

MongoDB's equivalent is four transformations, and Clausen argues that calling it four overstates it because the shape of the data never actually changes. Each format change in the SQL path is CPU, memory, latency, and a place a bug can hide. None of it makes the product better. All of it has to keep working anyway.

The second is schema evolution. Renaming a field, adding a nested object, or restructuring a document all happen live in MongoDB with zero downtime. The same operation against a large relational table can lock writes for minutes, sometimes hours. For teams shipping weekly, the difference isn't theoretical. It's the difference between adding a field this afternoon and scheduling a maintenance window next quarter.

Clausen's diagnosis is direct. The team that blame MongoDB for failures are usually the ones who built it like SQL: collections modeled as tables, Mongoose used as an ORM, **find()** calls scat-tered across files, nothing denormalized, nothing indexed. Then a 3,000-word post about how the database is the problem. "It's not," he writes. "They are."

None of this is an argument that relational engines are obsolete. They remain the right answer for many workloads, and large enterprises will run them for decades. The point is narrower. When the workload is JSON-shaped, document-oriented, and iterating weekly, the assumptions underneath relational design stop being neutral. Which raises a different question: do modern relational engines actually close that gap, or only paper over it? Postgres with native JSON support is the leading example.

READ THE ARTICLE

Postgres or MongoDB? An architect's decision tree

There's an obvious objection: modern relational engines now support JSON columns, vector indexes, and array fields. Surely a Postgres deployment with JSONB and pgvector can carry an enterprise into the AI era without forcing a data-layer migration, right? The question is fair, and worth answering on technical merit rather than tribal allegiance. Franck Pachot, a 30-year veteran of Oracle, Postgres, YugabyteDB, and MongoDB, as well as an AWS Data Hero and Oracle Certified Master, takes up exactly that question in a talk worth watching in full. His framework isn't partisan; he recommends Postgres for some workloads even from a MongoDB role. That balance is what makes it worth taking seriously.

Two technical observations shift the comparison from preference to architecture. The first is what happens to a 10 MB document on each engine. Insert it on MongoDB and a Linux trace shows write calls of 10 MB each, because the document stays together as one B-tree leaf. Insert it on Postgres and the same trace shows 8 KB writes, because Postgres splits the document into 8 KB blocks using TOAST. The second is what "a single table" actually means. A Postgres table holding a JSON column isn't one table behind the scenes. It's two tables and two indexes: the main table with its primary key index, and a TOAST chunk table with its own index. Fetching one large document is, mechanically, a nested-loop join across both.

Physical data locality

MongoDB clusters data physically

PostgreSQL splits it to multiple tables, multiple indexes

```
create table "MongoDB" (
 _id UUID primary key
         default uuidv7(),
 data JSONB
);
CREATE TABLE

select amname, count(*)
 from pg_class c join pg_am a on c.relam=a.oid
 where c.oid >= '"MongoDB"'::regclass
 group by amname;
```

```
1 select c.oid, relkind, amname,relname from pg_class c
2  join pg_am a on c.relam=a.oid
3  where c.oid >= '"MongoDB"'::regclass
4  order by c.oid;
```

oid	relkind	amname	relname	
16384	r	heap	MongoDB	table for tuples
16389	t	heap	pg_toast_16384	table for toasts
16390	i	btree	pg_toast_16384_index	index for toasts
16391	i	btree	MongoDB_pkey	index for tuples

Figure 5.
The physical reality of a Postgres table holding a JSON column. One CREATE TABLE statement, four objects under the hood: a tuples table, a TOAST chunks table, and an index for each.

CONTINUED ON NEXT PAGE

The same divergence shows up at the indexing layer. Consider a query familiar to anyone running an operational system: Return the last 10 orders for a given product in a given country. In the document model, a single compound index serves this directly, including the fields nested inside arrays. In a relational system holding the same data as JSON, the equivalent typically requires a GIN index for the array contents, a separate B-tree index for the scalar fields, and a sort the planner can't avoid. The plan reads more rows than necessary, and it scales unfavorably as the dataset grows. None of this is visible from the application layer. All of it is visible on the cloud bill.

The structural insight beneath both points is data locality. In the document model, the logical model and the physical model are the same. The shape the application writes is the shape the database stores, and the shape the database stores is the shape the retrieval layer returns to an AI agent. That equivalence isn't a slogan. It's what allows the data, the index, and the retrieval path to live as one, without the synchronization pipelines, double-write logic, and reconciliation jobs that fragmented architectures otherwise need to approximate the same outcome.

Pachot's decision framework lands in the direction of the technical observations. A central database serving many applications, where the use cases aren't all known yet, is a relational job, and Postgres is the right relational answer. A single application built from a domain model and persisted as the application reasons about it is a document job, and that's where MongoDB earns its place. Both engines are honest answers to different questions. The question worth asking next is which one describes the work being built today. Is it microservices with bounded contexts, schemas that iterate weekly, or domain objects persisted the way the application thinks about them?

Pachot's closing line is worth carrying into any database conversation in 2026: "A database is good and fast if you use it correctly, and the most important thing is to choose one you know, or that you want to learn."

The responsibility goes back where it belongs: on the architect, not on the engine. The next section shows what that responsibility looks like when the workload is patient safety at the scale of 1.2 billion containers a year.

WATCH THE TALK

MCKESSON

McKesson meets compliance at scale with MongoDB

Architecture arguments are abstract until the largest drug distributor in the United States stakes patient safety on them.

When the U.S. Drug Supply Chain Security Act took full effect, McKesson faced a non-negotiable mandate: Trace more than 1.2 billion serial numbers annually across its supply chain, with every scan authenticating a container back to the manufacturer, in real time. Their existing setup, SAP and Postgres, struggled with the flexibility and scale the regulation required.

The framing from McKesson's principal IT architect, Brian Schmidt, was characteristically operational: The data accompanying each container of medication is as critical as the medication itself. Medication arriving without verifiable, deliverable data can't be released to the patient who needs it.

McKesson chose MongoDB as the foundation for two new systems. The ***Central Data Repository*** manages serial number data for the 350,000 customers McKesson serves daily. The ***Distributed Serial Repository*** is a distributed application handling verification calls across McKesson's network of distribution centers. The document model was a natural fit for hierarchical supply chain data, where a single shipment can contain dozens of products, each with its own provenance, lot, expiration, and chain-of-custody history. Modeling that in flat tables would have meant a join graph that became its own latency problem. Modeling it as documents meant the data shape mirrored the business object.

Schmidt's own description of what go-live felt like is the line that earns its place as the section's quote: "There was no blip on the map." For a top-10 Fortune 500 company operating under a federal compliance deadline and supplying medication to a third of the country, no blip on the map isn't a minor technical achievement. It's the difference between a successful regulatory go-live and a public failure with patient-safety consequences.

The takeaway here isn't that healthcare is uniquely hard (though it is). It's that compliance at scale and architectural agility aren't opposing forces. McKesson modernized for compliance, not AI. The foundation the team built is what they now use to explore generative AI and the next class of use cases. The architectural choice that made compliance achievable at scale is the same one that makes the next workload achievable at all.

> The scale that we achieved with MongoDB is overwhelming. It's a moment we should all be incredibly proud of.

Upendra Kulkarni, *Principal Product Manager, McKesson*

READ THE STORY

WATCH THE KEYNOTE

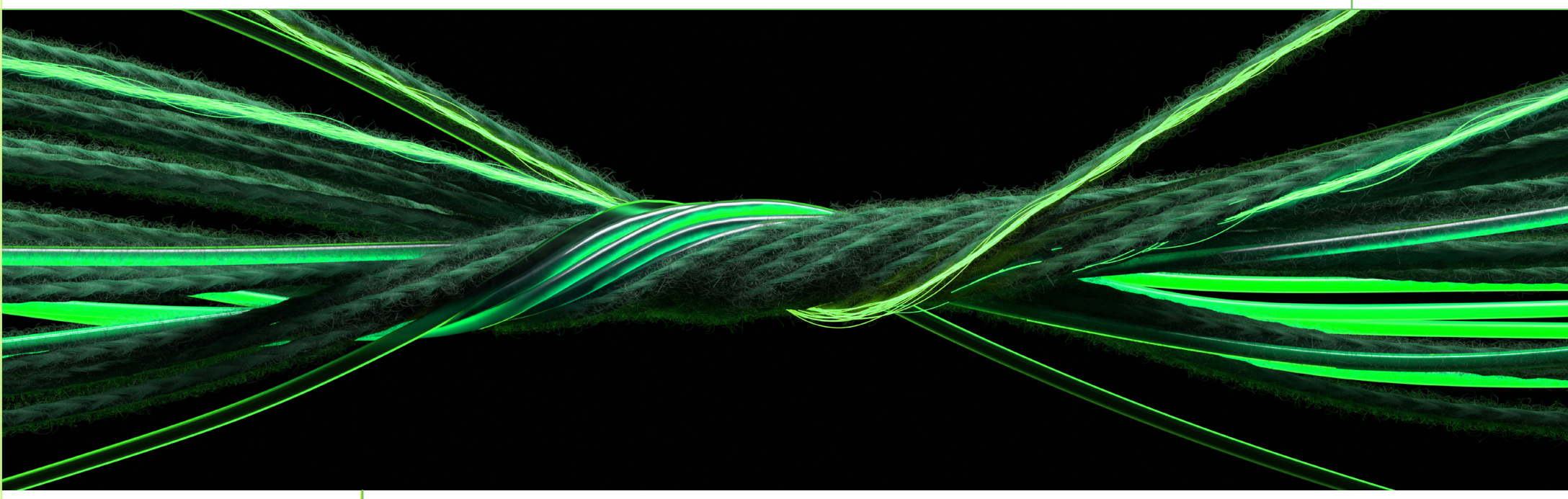
From architectural drag to the unified intelligence layer

Section 01 was the diagnosis. The architectural drag is real, IDC's research validates it, and McKesson shows what paying it down looks like at scale. A diagnosis without a treatment plan, though, is just an audit.

Section 02 reveals the treatment plan. Once you accept that stitching together separate stores and retrofitting document workloads into relational tables impose the same cost in different currencies, the questions become practical:

1. What does a unified intelligence layer actually look like?
2. Which workloads does it have to absorb?
3. What does it feel like to operate one when your business depends on agents that act in production every day?

Over the next 10 pages, we'll answer all three, shown across the industries where the unified pattern pays back first: manufacturing, retail, financial services, and automotive.



SECTION TWO

The Unified Intelligence Layer

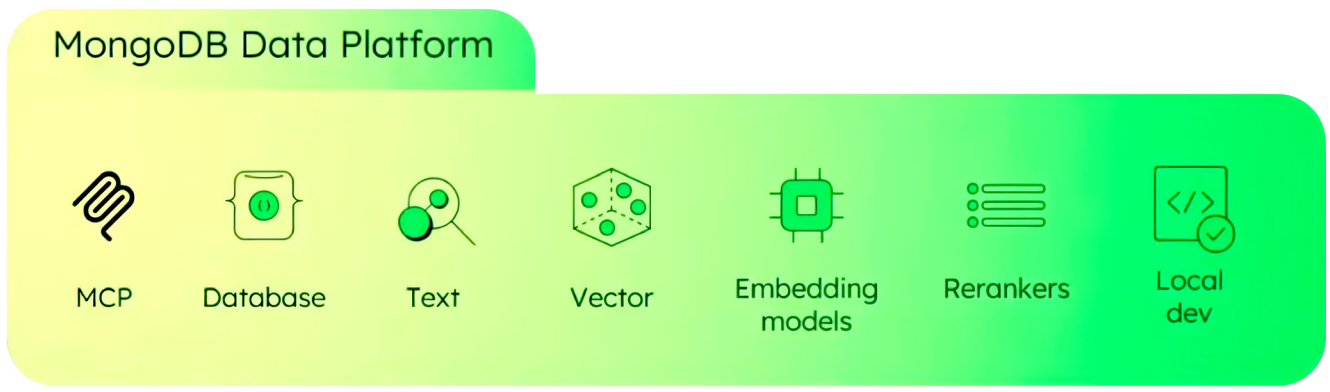
A single platform for the full retrieval lifecycle.

A unified intelligence layer isn't an idea. It's a list of capabilities that have to live in the same place. This section takes them apart one at a time, showing why each one matters when an agent has to act on what it reads.

The architectural pattern this volume is built around has a name: a ***Converged Datastore for agentic AI***. The entities a business operates on, the embeddings its AI agents reason over, and the state those agents accumulate across interactions all live in the same place: under a single API, a single query language, and a single security model. Operational data, vector search, full-text search, embedding generation, agent memory, and stream processing stop being six integration problems and become one platform.

Models will keep improving. Frameworks will keep churning. The data layer is the part that has to outlast both. Architect it well and model swaps and framework migrations become refactors instead of rebuilds. Architect it poorly and every new wave of AI tooling forces you to replatform.

Section 01 made that case at the architectural level: split versus unified, with the architectural drag falling on whoever picked split. The same contrast plays out at the AI workload itself, capability by capability.



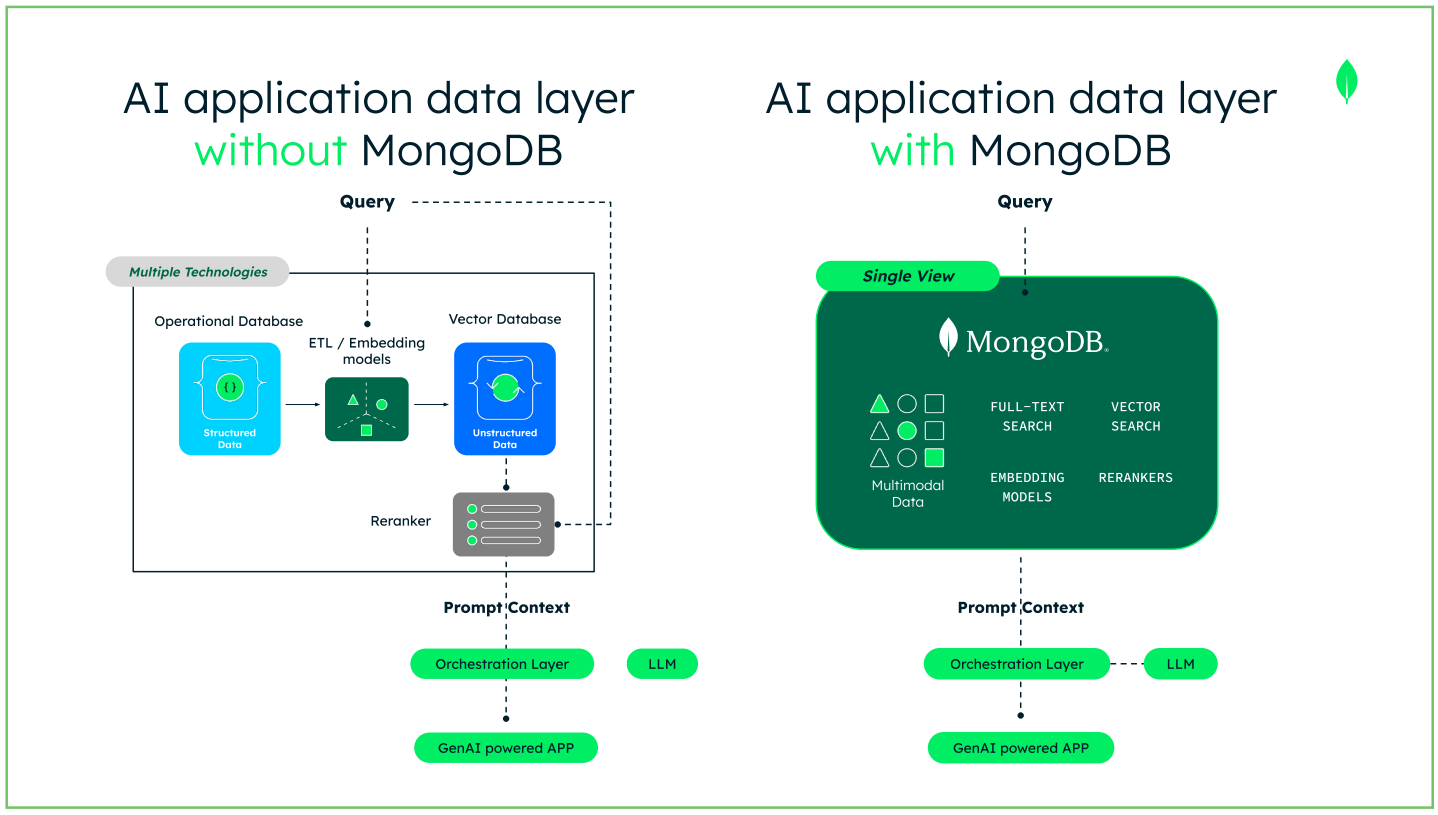


Figure 6.
The AI application data layer, before and after consolidation. On the left, multiple technologies stitched together with ETL pipelines: operational database, embedding models, vector database, reranker. On the right, the same capabilities native to a single store.

What follows is a sweep across four industries. AI itself is domain-agnostic: a weld robot reporting torque, a fraud investigator searching transcripts, a technician matching a corroded connector to a fault tree, an agent retrieving a customer's order history. Different vocabularies, different users, same foundation underneath. So this section moves through industries deliberately, showing what changes about the workload and what stays the same.

We start with manufacturing, where the Unified Namespace pattern already exists on most factory floors and needs only memory to become agent-ready. We move to retail, where unified commerce solves a decade-old data-silo problem and incidentally makes the AI agent that handles support queries possible. Then financial services, where one platform powering search, semantic similarity, graph reasoning, and machine learning side by side is the only honest answer to financial crime at machine speed. We close with automotive, where after-sales diagnostics is one of the few places a unified intelligence layer pays back inside a single product cycle.

The thread through all four is the same: consolidation pays back in places that look unrelated until you look at the architecture diagram.

Section 01 ended on a question: What does paying the architectural drag down actually buy you? Section 02 answers that question.

READ THE CONVERGED DATASTORE BLOG

WATCH REDMONK INTERVIEW

Manufacturing's data layer needs memory for agentic AI

The Unified Namespace pattern exists on most modern factory floors, whether or not anyone calls it that: a single semantic hierarchy where every controller, sensor, and system of record publishes live production data, and every subscriber reads from the same place. It's the architectural answer to a generation of point-to-point integrations that no one wants to maintain. But the namespace alone won't carry manufacturers into the agentic era. A namespace is transient. An agent needs memory.

Programmable logic controllers (PLCs), the small computers that run individual machines–along with sensors and supervisory control systems – publish into the namespace; the manufacturing execution system (MES), enterprise resource planning (ERP), and quality systems subscribe to it. That's enough to run a modern factory. It's not enough to run an agent on top of one. An agent that responds to a tolerance drift, schedules unplanned maintenance, or coordinates a line changeover doesn't just need the current state of the world. It needs prior incidents, last week's quality data, the operator notes from the last shift, and the embeddings of the maintenance manuals it consulted last time. The namespace publishes the present. Memory holds the past.

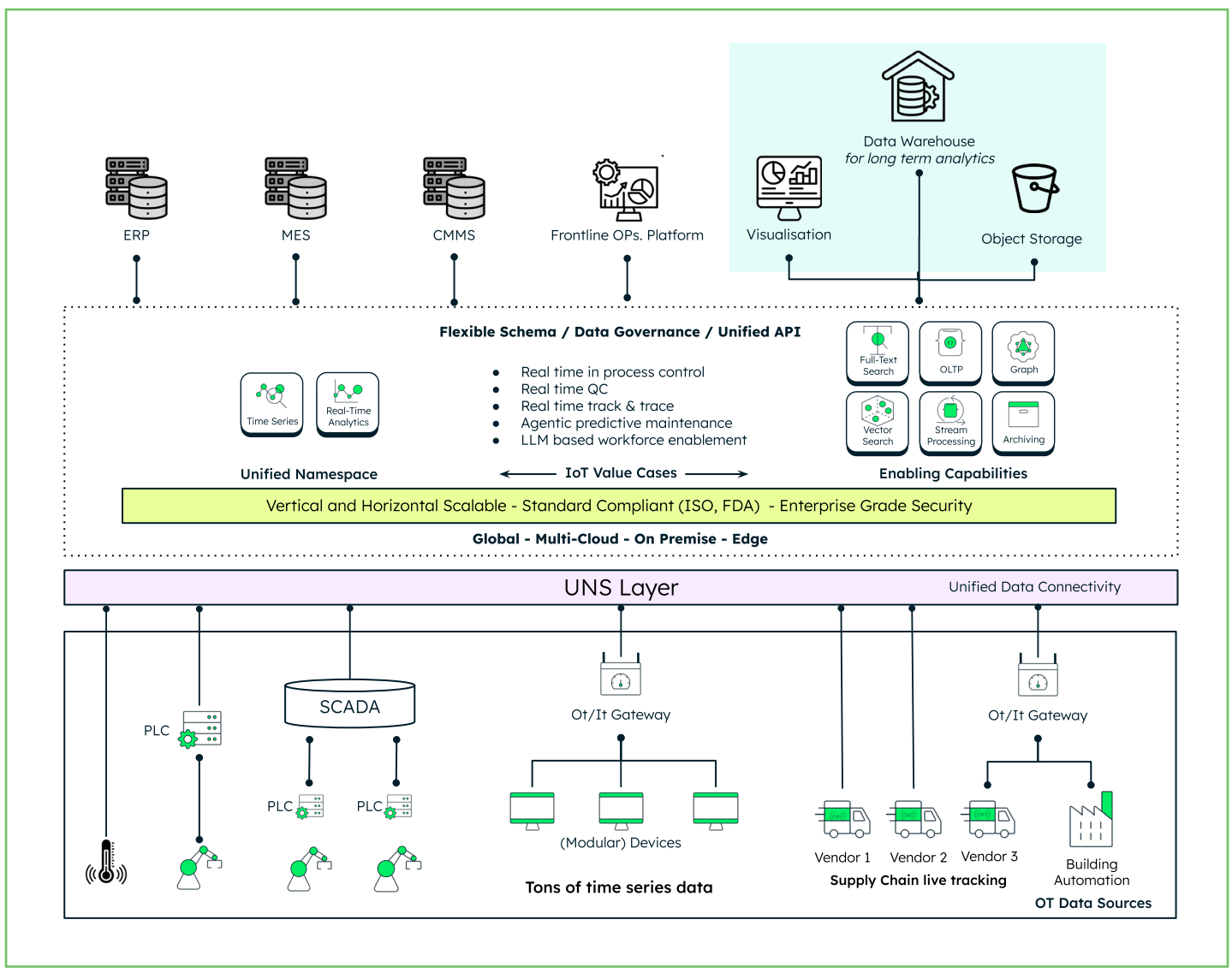


Figure 7.
A unified architecture linking OT data through the Unified Namespace to a scalable, real-time industrial data platform.

CONTINUED ON NEXT PAGE

This is where MongoDB extends the namespace into a memory-bearing data layer. The same documents that hold operational telemetry also hold embeddings, conversation history, and the audit trail of agent decisions. A weld robot's torque reading sits next to the work order, the part number, the operator, and the eventual quality outcome. So when the welding line shows torque drift across three shifts, the agent can pull every prior weld with similar drift signatures, match them against maintenance records and operator notes, and open a maintenance work order before the next failed inspection. It names the suspected component, the technician with the right certification, and a window in the production schedule when the line can come down.

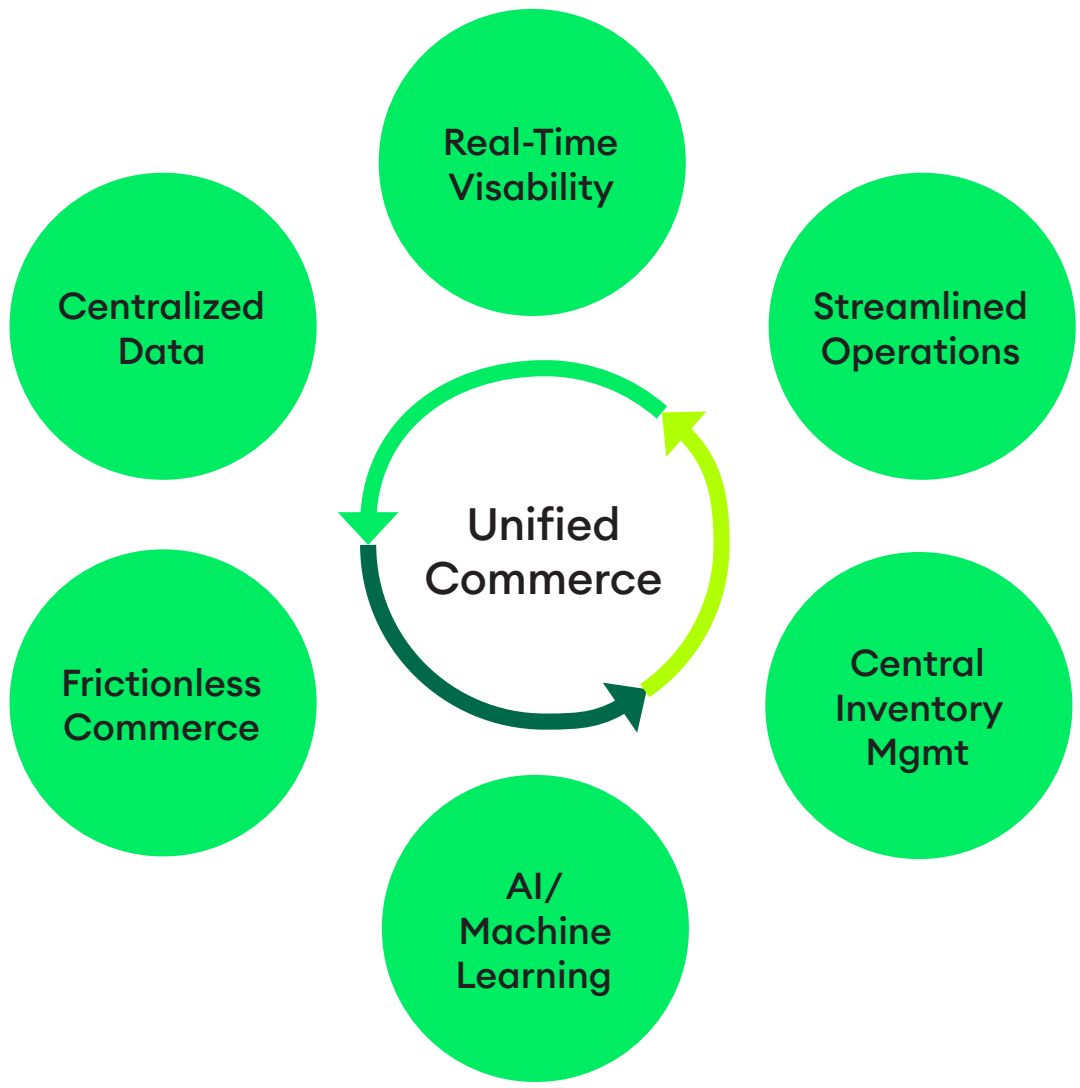
The capabilities that make this possible come from the same engine. Time series collections handle the high-cardinality sensor streams. Vector search handles semantic retrieval over manuals and incident reports. Stream Processing handles real-time enrichment as data lands. Together these capabilities turn the namespace from a feed into a foundation. The next AI workload the factory wants to add lands on it, not on a new one.

DOWNLOAD THE WHITE PAPER

Unified commerce with smart search

Retail teams have spent the better part of the last decade trying to deliver a single experience across web, mobile, physical store, and call center. The data was always there. It just lived in several different systems. The result was a customer who could see one set of inventory online and a different one in store, recommendations that contradicted each other, or a search bar that returned different results in the app than on the desktop.

Unified commerce fixes the surface challenges by integrating the data underneath. The MongoDB data platform handles product catalog, inventory, pricing, order history, and personalization, while a smart search layer combines lexical precision, semantic meaning, and geospatial signals: the same query that ranks products by relevance can also surface the nearest store with stock.



EXPLORE THE SOLUTION

What becomes possible once the data is unified is more interesting than what gets solved. Hybrid search returns results that respect both keyword precision and semantic meaning. Personalization draws on cross-channel history without an overnight ETL. And the same data layer that powers the storefront powers an AI agent that handles support queries, because there's only one source of truth to ground both. The pattern that pays back for the storefront pays back at machine speed too, for workloads such as financial crime detection.

Building a financial crime mitigation platform with MongoDB

Financial crime detection has always lived at the intersection of two data problems. The first is structured: transactions, account hierarchies, Know Your Customer (KYC) fields, and sanction lists. The second is unstructured: case notes, transcripts, documents, and the long tail of context that explains why a transaction may be suspicious. For decades, banks have tried to solve both halves in separate systems and reconcile them at the alert level. Investigators have paid the cost, most visibly in the 90% of legacy-system alerts that turn out to be false positives, overwhelming compliance teams while real threats slip past.

The stakes have outgrown the architecture. Global financial institutions lose more than $2 trillion to financial crime each year, a figure projected to nearly triple to $6 trillion by 2030, and AI-assisted adversaries already operate at machine speed. The fix is one platform powering search, semantic similarity, graph reasoning, rules, and machine learning side by side. Appropriate access controls are the trustworthy default for any platform handling this much sensitive data. Hybrid search lets investigators ask both *“who else moved money on this device in the last 90 days?”* and *“who else used language similar to this case in their support transcripts?”* from the same query interface, against the same source of truth.

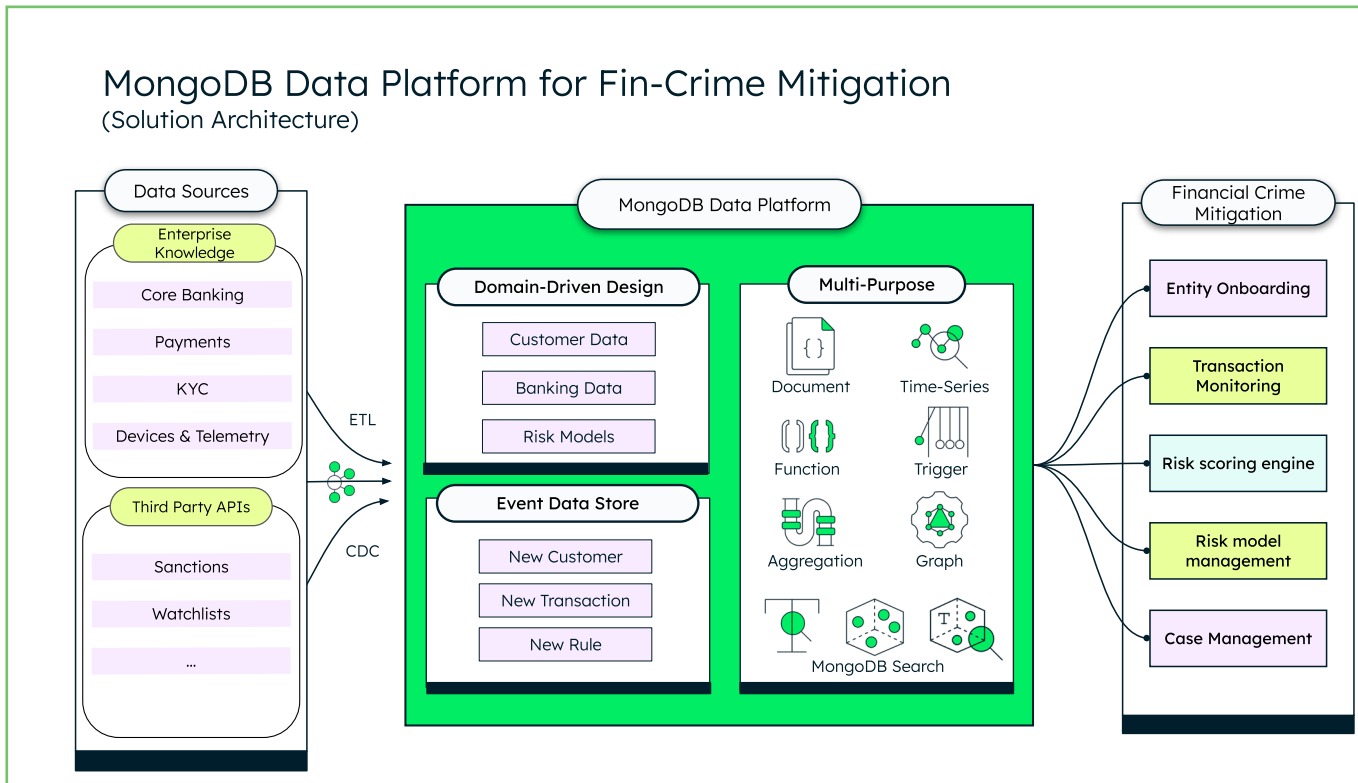


Figure 8.
High-level architecture of a unified anti-financial crime platform with MongoDB.

Fragmentation doesn’t make a financial crime platform more secure. It makes it slower to detect, harder to audit, and more expensive to operate, a cost the next piece traces through automotive diagnostics and the assembly line.

READ THE BLOG

Inside an intelligent automotive after-sales diagnostics app

The diagnostic conversation that used to happen between a service advisor, a technician, and a binder of repair manuals is increasingly happening between a service-bay tablet and an AI agent that has read every manual and bulletin issued for that platform. The architecture that makes this work is more interesting than the frontend suggests. The stakes are larger. Technicians can spend up to 30% of their time searching for information rather than repairing vehicles, and “No Fault Found” (NFF) events, where a working part is replaced because the diagnostic was inconclusive, account for nearly 30% of warranty costs.

The pattern starts with the manuals. Service documentation and technical bulletins from the original equipment manufacturer (OEM, the carmaker itself) are processed through optical character recognition (OCR), chunked to preserve procedural integrity, and embedded with both text and multimodal models, so a technician can describe a symptom in words or snap a photo of a corroded connector. The data layer is unified inside MongoDB Atlas: text embeddings, multimodal image embeddings, and the relationship graph between symptoms, systems, and root causes all live in the same collection. The agent uses vector search to find the relevant manual section, traverses the fault tree with $graphLookup (MongoDB’s operator for recursively querying connected data) to suggest the next logical step, and applies a Voyage AI reranker that pushes safety warnings and verified fixes to the top. Mechanics think in systems, not keywords: a dead battery might be caused by a trunk latch with parasitic draw, and a unified graph-and-vector backend is what lets the agent suggest checking the latch when the technician searches for the battery.

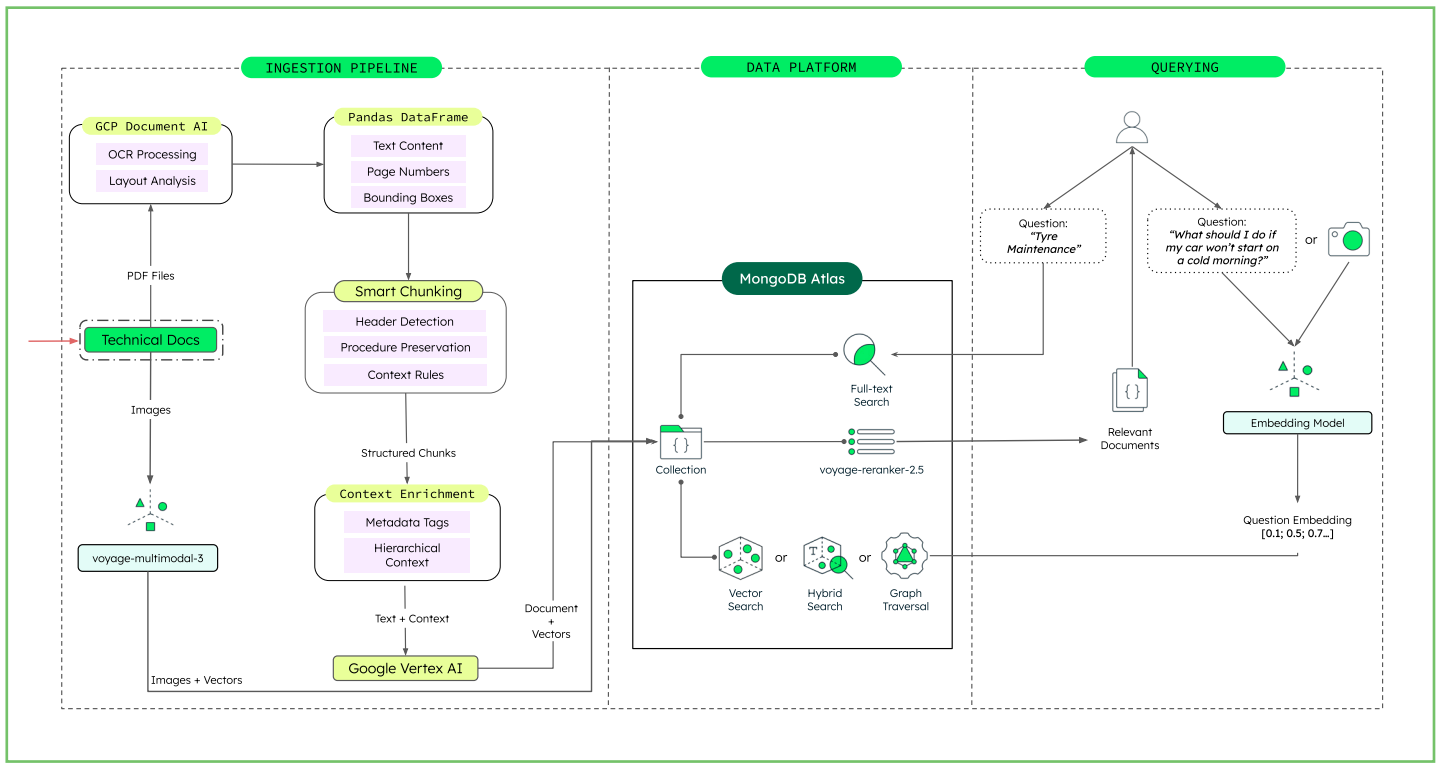


Figure 9.
Unified vector, graph, and multimodal architecture for automotive diagnostics on MongoDB Atlas. The ingestion pipeline processes manuals through OCR and embeddings; queries can be text or images.

BUILD THE APP

For automotive teams, this is one of the few places where a unified intelligence layer pays back inside a single product cycle: lower diagnostic time, fewer escalations, faster first-time-right repairs, and meaningful reductions in NFF warranty cost. The architectural change is the prerequisite, not the consequence.

From unified intelligence layer to accuracy at scale

Section 02 was the architecture. The Converged Datastore is real, four industries proved it absorbs different workloads under one roof, and the thread through manufacturing, retail, financial services, and automotive is the same architectural argument in different vocabularies. An architecture without trust, though, is just a diagram.

Section 03 is about the discipline that earns the trust. A unified layer can hold operational data, vectors, search, and agent memory in one place. But if the retrievals it serves to an agent are stale, incomplete, or imprecise, the agent hallucinates and the architecture's elegance becomes the customer's problem. Once you accept that hallucinations are a retrieval problem rather than a model problem, the questions become practical:

What does it take to ground every agent answer in real-time operational truth?

How do you keep embeddings, retrieval, and reranking fast and cheap without giving up accuracy?

What does the data layer have to deliver before a production agent is safe to ship?

Over the next eight pages, we answer all three: Voyage 4 raising the accuracy ceiling, the engineering decisions that make retrieval a pick-three problem rather than a pick-two, and LG U+'s call center, which already runs all of this in production at 3.5 million calls a month.

SECTION THREE

Accuracy at Scale

Grounding AI agents in real-time operational truth.

Section 02 made the case for a unified data layer. This section is about what that layer has to deliver before an agent built on top of it can be trusted in production.

Hallucinations aren't just a model problem. They are a retrieval problem. When a language model invents a number, an entity, or a policy that doesn't exist, it's almost always because the retrieval that fed the prompt was inaccurate, stale, or incomplete. There's a name for the distance between a system that returns an answer and a system whose answers can be acted on: the ***Trust Gap***. This gap is not abstract. An agent reasoning over yesterday's snapshot of customer data, a vector that lags the record it describes, or a retrieval whose grounding has quietly drifted because the embedding model was swapped and the data wasn't reprocessed: none of these produces a visibly wrong answer. It produces a confident one. A hallucination in a single demo is a curiosity. The same hallucination applied across thousands of customer interactions a minute is an incident.

MongoDB has spent the last two MongoDB.local events shipping the building blocks for closing that gap. At MongoDB.local San Francisco in January 2026, the message was ship faster: the Voyage 4 series raised the retrieval-accuracy ceiling, the Embedding and Reranking API brought high-fidelity retrieval models natively into the data layer, and Automated Voyage AI Embeddings began generating the moment data is written so agents always have fresh context. At MongoDB.local London in May 2026, the message tightened to run agents in production at enterprise scale: MongoDB 8.3 went generally available with up to 45% more reads and 35% more writes over 8.0, and the LangGraph.js Long-Term Memory Store went generally available too, giving JavaScript and TypeScript teams the same persistent memory Python developers have had.

CONTINUED ON NEXT PAGE

October 2024

Available Today

8.0 → 8.3

35% More Write Throughput

45% More Read Throughput

15% More Transaction Throughput

Two events, one argument. Accuracy is an architectural property. Call it Contextual Integrity: the discipline of keeping operational data, embeddings, search indexes, and agent state consistent automatically rather than reconciled after the fact. When those four things live in the same record served by the same query path, there is no synchronization step for accuracy to leak through. The retrieval an agent receives is grounded in the operational truth of the same instant it was issued. The trade-offs that remain (quantization, dimensionality, embedding choice) are tuned against a foundation whose accuracy isn't eroding underneath them.

.LOCAL SAN FRANCISCO RECAP

.LOCAL LONDON RECAP

The Voyage 4 series is now available

Voyage 4 is a family, not a model. The series ships in four sizes: *voyage-4-large*, *voyage-4*, *voyage-4-lite*, and the open-weights *voyage-4-nano*. The headline is retrieval accuracy. Voyage AI's embedding models lead the Retrieval Embedding Benchmark (RTEB), outperforming comparable models from Gemini, Cohere, and OpenAI on the kinds of tasks that show up in enterprise RAG. The RTEB leaderboard remains the closest the industry has to an objective measure of how well an agent will find what it needs.

The architectural innovation is the shared embedding space, an industry first. Earlier generations required identical models on both sides of a query, which meant teams locked themselves into a single accuracy and cost profile when they built the index. Voyage 4 models are interoperable: all four produce compatible embeddings. The recommended production pattern, asymmetric retrieval, follows from this. Embed your document corpus once with voyage-4-large for maximum accuracy, a one-time cost. Embed queries continuously with voyage-4-lite or voyage-4-nano to keep per-query latency and cost low. The big-model accuracy lives in the index; the small-model economics live at serving time. The section opener named "the embedding model was swapped and the data wasn't reprocessed" as one of the failure modes that produces the Trust Gap. The shared embedding space removes the precondition for that failure: nothing has to be reprocessed, because the new model speaks the same coordinate language as the previous one.

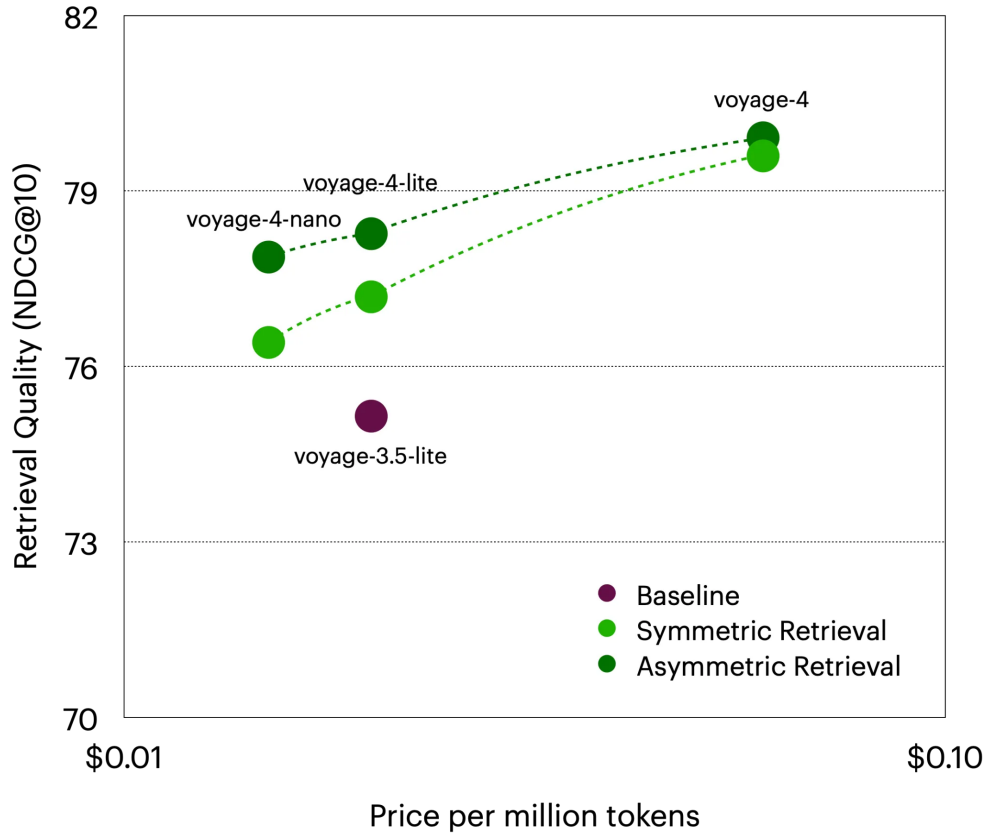


Figure 10.
Asymmetric retrieval lifts quality at the same cost.

The voyage-4-large model itself uses a mixture-of-experts (MoE) architecture that delivers state-of-the-art accuracy at roughly 40% lower serving cost than comparable dense models. It's the first production-grade embedding model to use MoE. For high-volume agentic workloads where a single agent might issue dozens of retrievals per task, that cost ratio compounds quickly. But the model is only the ceiling. Reaching it in production is an engineering problem.

READ THE ANNOUNCEMENT

EARN THE SKILL BADGE

Building fast, cheap, and accurate retrieval

The triangle of fast, cheap, and accurate is supposed to be a pick-two problem. Retrieval pipelines are where it actually becomes a pick-three, because the wrong choices early lock in cost and latency that no amount of model upgrade can rescue. Honestly framed, production retrieval has to be solved on three axes simultaneously: accuracy (the right context, grounded in the operational truth of the moment the query was issued), latency (a correct answer that arrives after the user has moved on is, for operational purposes, a wrong one), and cost (a pipeline that runs on every query and reruns multiple times per agentic interaction can't scale economically if each call is expensive).

Voyage 4 raises the accuracy ceiling. The engineering work to reach it in production covers three optimizations, each targeting a different bottleneck. ***Dimensionality reduction*** uses Matryoshka Representation Learning to truncate embeddings while preserving meaning: 50% less storage and memory, 40% faster queries, 98% of baseline accuracy retained. ***Binary quantization*** compresses high-precision vectors to one bit per dimension: 96% less memory, 60% faster queries, 99% of accuracy retained. ***Asymmetric retrieval*** pairs a larger model for documents with a smaller model for queries: 50% lower per-query embedding cost with voyage-4, up to 83% lower with voyage-4-lite. Concrete numbers, not vibes.

Optimization	Mean NDCG@10	P50 latency (ms)	P95 latency (ms)
None (baseline)	0.65	1.772	2.11
Dimensionality reduction	0.643	1.049	1.265
Binary quantization	0.646	0.718	0.841
Asymmetric retrieval	0.64	1.722	2.006

Figure 11.
Retrieval optimization techniques evaluated on the NFCorpus dataset. All three approaches preserve 98% to 99% of baseline accuracy while delivering 40% to 60% latency improvements.

READ THE BLOG

The cost-scaling story is the one worth carrying back to the architecture review. At 10,000 queries a day, asymmetric retrieval with voyage-4-lite saves roughly $11,000 a year against the baseline. At 1 million queries a day, the same configuration saves over $1 million annually. And the bigger that number gets, the more the data layer matters, because every optimization has to compose cleanly without forcing the team to replatform underneath them.

The through-line is that accuracy at scale is a stack of choices, not a single optimization. Get the embedding model right and you bound your accuracy ceiling. Get the dimensions right and you bound your storage and memory. Get the quantization right and you bound your latency. Get the query-side embedding strategy right and you bound your per-query cost. Get the data layer right and these decisions compose instead of compete.

AI-powered medical report generator

Clinical documentation is the part of medicine that consumes the most time and produces the most burnout. In specialty units like oncology, patients arrive with a long trail of scanned reports, PDFs, pathology narratives, imaging summaries, and lab results scattered across disconnected systems. Before every multidisciplinary team meeting, coordinators and residents spend hours reconstructing the story: extracting facts, resolving contradictions, normalizing terminology, and building a coherent timeline. The data problem underneath the documentation crisis is what makes the AI promise hard to keep.

The reference architecture for an AI-powered medical report generator fuses structured Electronic Health Record (EHR) data, unstructured clinical notes, imaging metadata, and the relevant medical literature into a single retrieval pipeline. MongoDB acts as the unified data layer, holding patient records, embeddings of clinical content, and the audit trail required for clinical governance.

The reference is opinionated about accuracy. Every generation step is grounded in retrieved sources. Every retrieved source is attributable. Every clinician edit feeds back into the system as a signal. The point isn't to remove the clinician from the loop. The point is to make the clinician's work the high-judgment work, not the transcription.

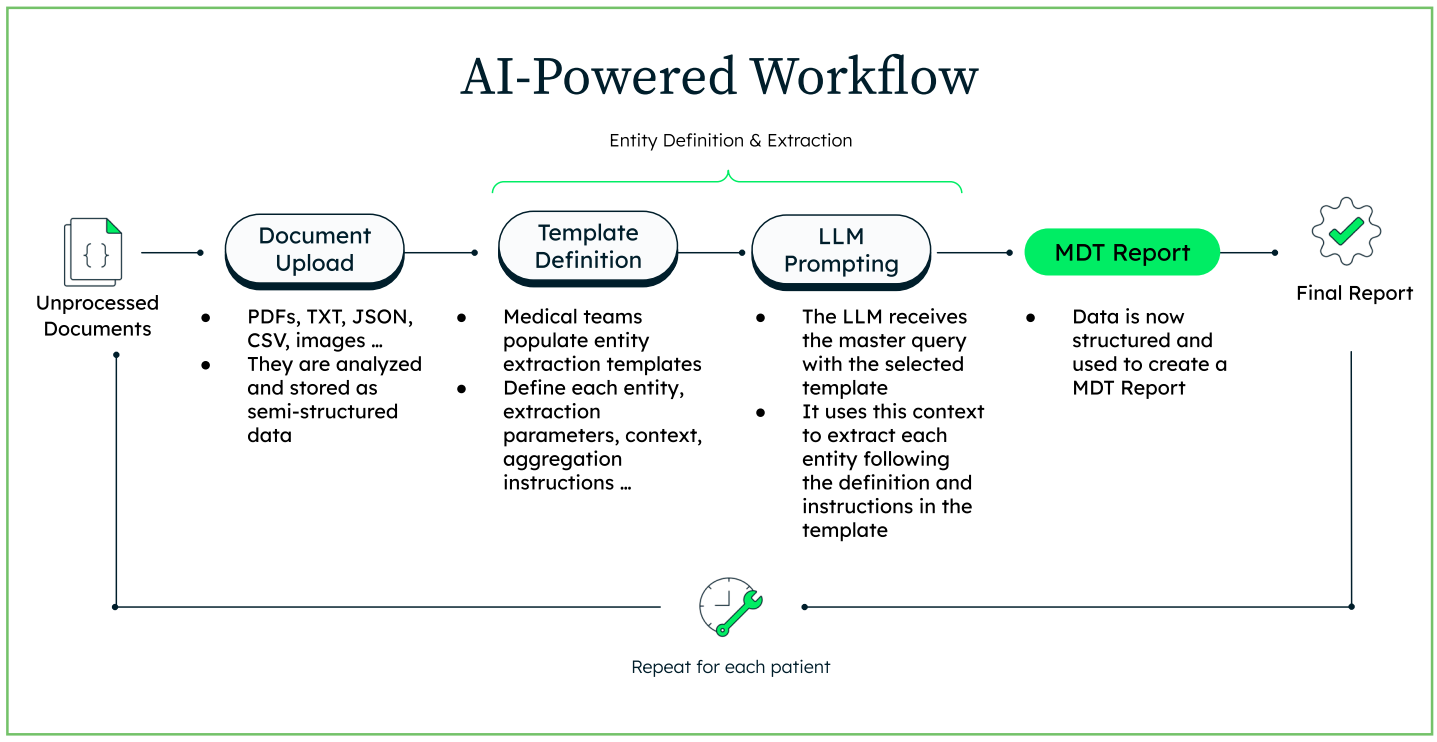


Figure 12.
AI Workflow Diagram

EXPLORE THE SOLUTION

The Builder's Journey: building AI apps with MongoDB

For engineers tired of conceptual AI content who want to see something get built end to end, the Builder's Journey is the MongoDB.local San Francisco talk to watch. It builds a developer productivity app on stage in two evolutions: V1 ships an AI assistant for brainstorming projects with semantic search over past work; V2 adds image ingestion, multimodal search across text and images, and "memories": extracted preferences and implementation details that proactively personalize the next project. The architectural change between V1 and V2 is a single field set from 1 to 2. Same three collections. Same queries. Same cluster.

What makes the talk useful for technical leaders, even those who don't write code day to day, is the implicit architecture argument, and one explicit one. On the implicit side: every step reuses the same data layer. The same Atlas cluster powers operational data, vector search, agent memory, and the retrieval pipeline. There's no "and now we add a vector database" interlude. On the explicit side: the talk pauses on filtered vector search and walks through why MongoDB filters by user ID first and then runs the vector search, while Postgres runs vector search first and filters after, which means relevant results can fall out of the candidate set before they're ever evaluated. Small architectural choice. Real accuracy consequence at scale.

It's also a useful onboarding tool. Pointing a new team member at the talk gets them productive faster than any architecture diagram, because they see the choices being made in context, and the trade-offs being avoided. The talk teaches the pattern. The next story shows it carrying real production load.

WATCH THE TALK

LG U+ scales AI customer service with MongoDB Atlas

Architecture arguments hit a different ceiling when the workload is a call center taking 3.5 million calls a month.

LG U+, the Korean telecom that runs one of the country's largest customer service centers, set out to build Agent Assist: an AI tool that gives 4,000 human agents real-time transcription, automatic categorization of customer inquiries, and intelligent contextual summaries during live calls. The relational database underneath could not accommodate the vector data the AI features generated, the tests the team needed to run on those features during consultations, and the operational queries the service required, all in the same path.

LG U+ built Agent Assist from scratch on MongoDB Atlas and adopted Atlas Vector Search early in development. By migrating operational data off Postgres into the same store as the embeddings, the team eliminated the sync layer between vector and operational systems entirely. The document model accommodated the varied shapes of consultation data, and $vectorSearch ran semantic similarity directly against the same collections the agents were already querying for transactional answers.

The service launched four months after the project started. Processing time per call dropped 7%. Resource efficiency improved 30%. Sub-second latency held under load on $vectorSearch queries at scale, and auto-scaling absorbed the morning peak when call volumes spike. Initially deployed for 1,200 agents, the service now handles over a million user queries per week and is scaling toward 2,300 agents.

> "Managing both vector and operational data in MongoDB opened a new world to our team."
>
> **Minkyu Ha,** *Senior Software Engineering Manager, LG U+*

The decision that closes the story carries more weight than the percentages. Beyond Agent Assist, LG U+ is extending the same architecture into its callbots and chatbots, expanding the service into the B2B market, and working with MongoDB on a broader data-layer modernization that includes migrating substantial on-premises applications onto Atlas. The architectural choice made for one application has become the foundation for the company's AI portfolio.

That is what Contextual Integrity looks like at the scale of a national telecom. Vector embeddings and operational data sit in the same record. The retrieval an AI surfaces to a human agent on a customer call is grounded in the operational truth of the same instant the call is occurring. The Trust Gap this section opened with isn't a process to manage. It's a failure mode the architecture has been designed out of.

READ THE STORY

CONTINUED ON NEXT PAGE

What LG U+ shows is the shortlist of what the data layer has to deliver before a production agent is safe to ship.

- The AI's search index and the business's live data have to live in the same place, so the agent isn't answering from one version of the truth while acting on another.
- Semantic queries have to return in under a second, even under real production load. Anything slower and the agent is too late to be useful.
- The platform has to scale itself when call volumes spike, without the team rebuilding anything to absorb the peak.
- And the data model has to be flexible enough to hold whatever shape the conversation takes, without a schema migration every time the business learns something new. None of this is a model-quality problem.

All of it is architecture, decided before the first prompt is written.

From accuracy at scale to systems of action

Section 03 was about grounding. Voyage 4 raised the accuracy ceiling, and the engineering techniques behind it (dimensionality reduction, quantization, and asymmetric retrieval) made that ceiling reachable in production. LG U+ then showed what grounded retrieval looks like at production scale, with 3.5 million calls a month running on a unified data layer. A grounded agent that retrieves from operational truth, in the same instant the question is asked, is something more than a sophisticated chatbot. It's the agent that's ready to act.

Once accuracy at scale is solved, the question shifts again. What can an enterprise actually build that it couldn't build before? Section 04 is the answer.

SECTION FOUR

Building the AI Era

From systems of record to systems of action.

There's a difference between an agent that summarizes a claim and one that pays it. Between an agent that recommends a payment route and one that executes the wire. Between an agent that drafts a lending decision and one that funds the loan. The moment an agent transacts, the requirements multiply. Memory has to be durable across sessions. State has to survive crashes. Permissions have to be auditable. Accuracy was the easier half. Action is the harder one.

The industry has a name for the shift: ***systems of record versus systems of action***. A system of record holds, retrieves, and surfaces information. A system of action changes the state of the world it operates in. It commits transactions, triggers workflows, books appointments, updates ledgers, and makes payments. Section 03 perfected the first half. This section is about what changes when accuracy is no longer the only thing that can fail.

The stakes have a number. Gartner predicts 40% of enterprise applications will embed AI agents by the end of 2026. And 40% of agentic AI projects will be canceled by the end of 2027. The symmetry isn't accidental. The cancellations aren't a model-quality problem. Enterprise agent projects stall at three predictable points: at security review, where there's no industry-standard set of controls for autonomous systems; at cost ceilings, where token spend outpaces business value; and at the integration boundary, where the data layer can't keep up with what the agent needs to know.

Production agent systems have more in common with production Machine Learning systems than most teams realize. The model gets the attention. The infrastructure around the model decides whether anything actually works: memory, state, orchestration, observability, evals, security, and the data layer that feeds all of it. MongoDB's Designing an Agentic Platform series develops this architecture in depth. Here is its shape.

CONTINUED ON NEXT PAGE

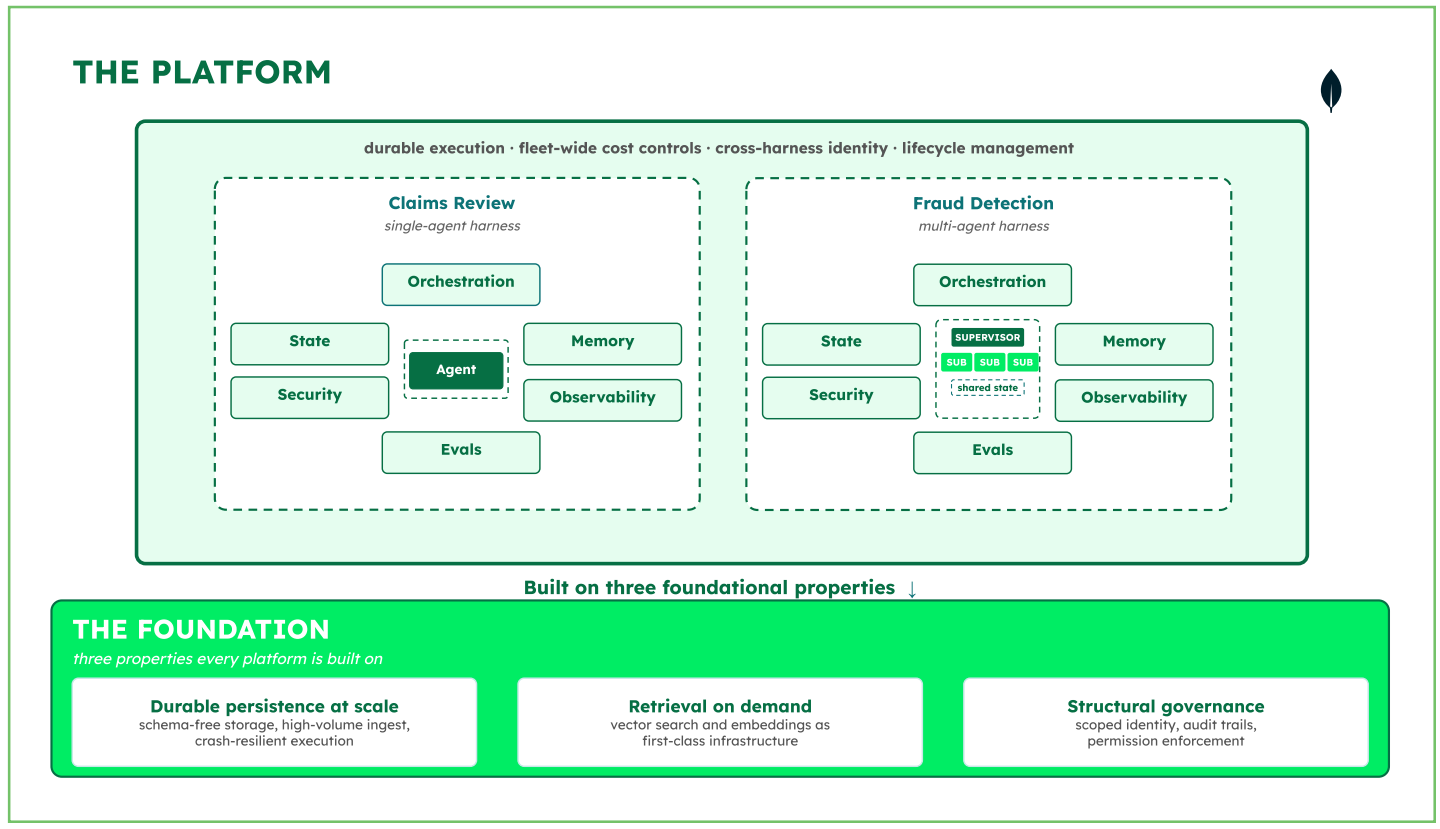


Figure 13.
A platform enables the execution of many harnesses and agents across a diverse set of use cases, domains, and frameworks.

Each box in the figure is a harness, the runtime that drives one agent or a coordinated set of agents through its decision loop. The six engineering disciplines inside it (orchestration, state, memory, security, observability, evals) are where most production work actually happens. The platform sits above them, running many harnesses across many teams over time.

Three foundational properties hold the platform up. Durable persistence at scale, for the schema-free data that harnesses generate. Retrieval on demand, through embeddings and vector search, so context is available the instant an agent needs it. And structural governance, with scoped identity, audit trails, and permission enforcement provided once at the platform rather than bolted onto each harness. A platform missing any of the three produces harnesses that appear to work in controlled conditions and fail under production load.

READ THE PLATFORM SERIES

Agents should be invisible: business objects matter more than AI memory

There's a subtle architectural mistake at the heart of most early agent platforms. Teams treat the agent's memory as the central abstraction, with everything else organized around it: the customer, the policy, the claim, and the transaction. It works in a demo because the demo only needs one agent and one workflow. It breaks in production because real businesses have many agents, many workflows, and one underlying truth: the business object.

Agents should be invisible. The customer doesn't interact with an agent. In insurance, the customer interacts with their policy, their claim, and their account. In retail, with their order. In healthcare, with their case. The system happens to use agents to act on those objects. Organize the architecture around business objects rather than agent memory, and three things follow. New agents plug into existing objects without re-architecting the data layer. Multiple agents can coordinate around the same object without stepping on each other. The audit trail naturally aggregates at the object level, where regulators and operators want it.

The data layer this requires has a name: the ***intelligence data layer***, where the business objects, the embeddings, the agent memories, the workflow state, and the audit trail all sit in a single coherent place. MongoDB's document model is a natural fit. Business objects are inherently nested. Access controls can be enforced at the field level. And the same query interface answers operational, semantic, and historical questions. This is the Converged Datastore pattern applied to the case where it pays back fastest.

The practical takeaway is to design the object model first. Agents come and go. The objects do not. Get the object model right, and every subsequent agent compounds on the same foundation. Get it wrong, and every agent is its own re-architecture.

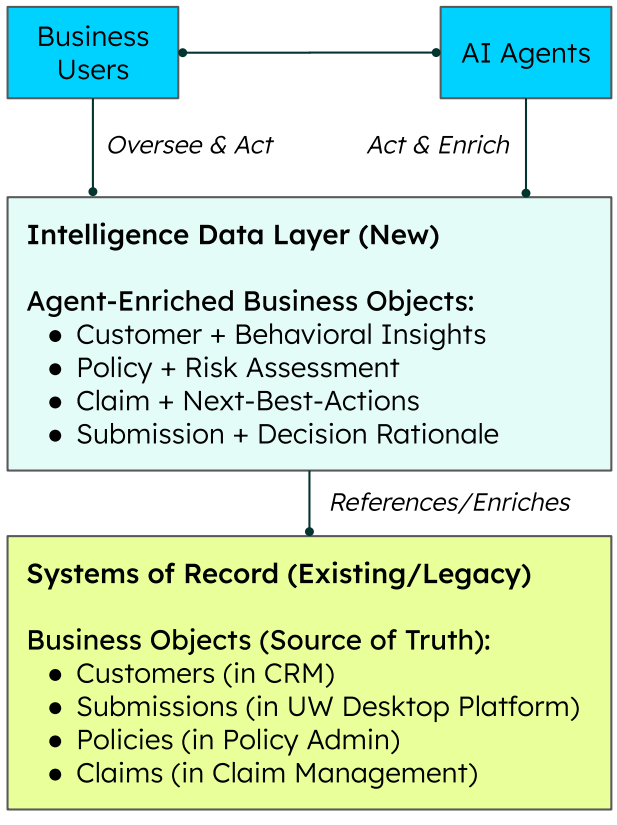


Figure 14.
The System of Action: Enabling More Intelligent Business User Oversight & Action

READ PART 1

READ PART 2

Build AI agents worth keeping: the Canvas Framework

Most agent projects fail in the same way. The team starts with a framework choice, picks a model, builds something interesting, and then realizes too late that no one needed the thing they built. McKinsey research is consistent on this. Nearly 80% of companies report using generative AI, but fewer than 10% of use cases ever make it past pilot. The pattern is so common that MIT researchers have a name for it: the gen AI divide.

The Canvas Framework is one direct response to this pattern. Mikiko Bazeley, a Staff Developer Advocate at MongoDB, modeled it after the business model canvas. The structure forces teams to answer the right questions in the right order: product first, then agent, then data, then model. The premise is that agent design sits between product and data, determining what knowledge the agent needs and which models can deliver it. Reverse the order, as most teams do, and you end up with technology in search of a problem.

The proof-of-concept (POC) canvas is eight focused squares aimed at validating quickly, organized in four phases: product, agent, data, and model integration. The production canvas adds 11 squares across five phases: business case and scale, robust agent architecture, unified data infrastructure, model operations, and the hardening that POCs skip and enterprises demand. Security review, cost ceilings, latency budgets, observability, governance, evaluation pipelines, and fine-tuning strategy. The framework forces teams to plan all of it before scale, not after.

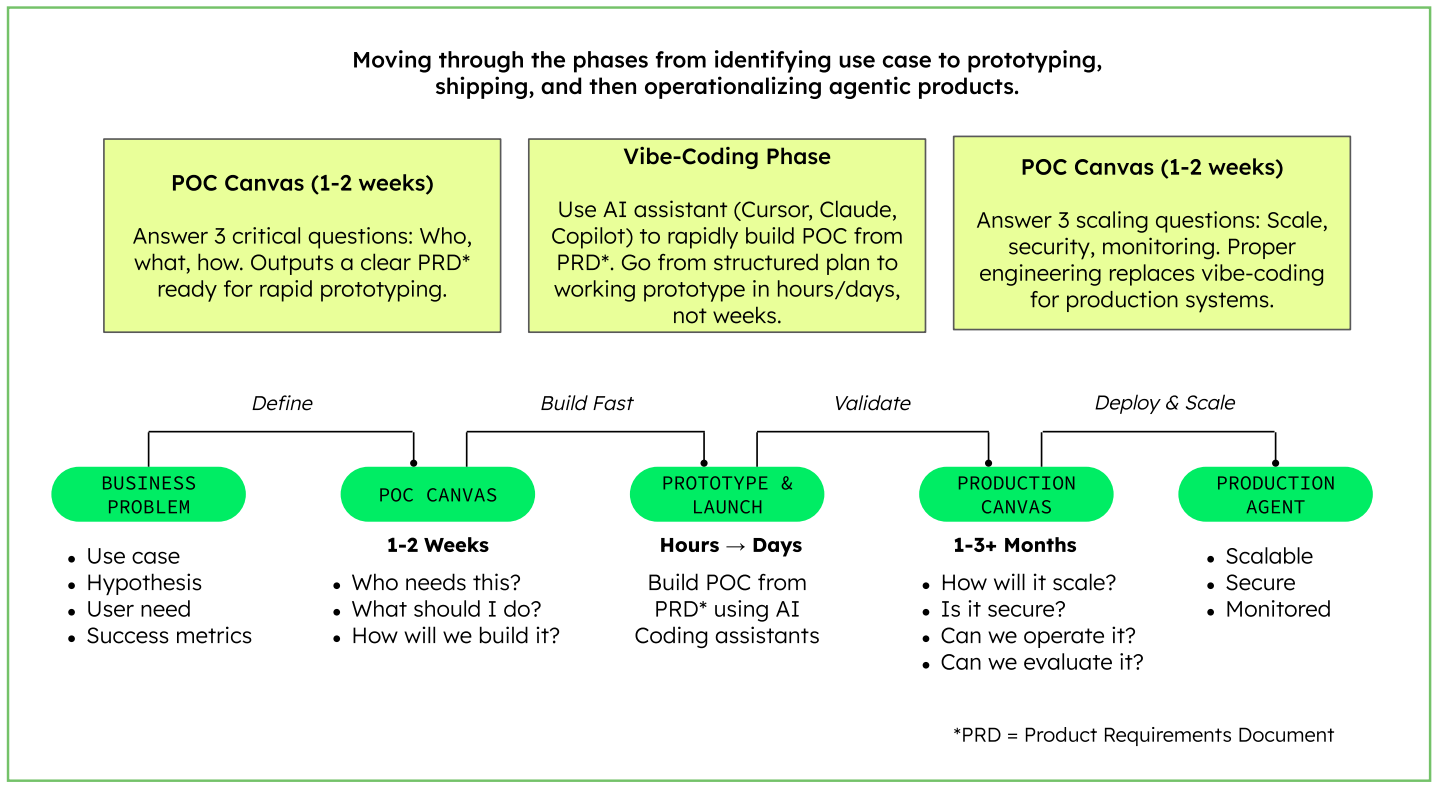


Figure 15.
The Agentic AI Canvas Framework. A structured five-phase approach moving from business problem definition through POC, prototype, production canvas, and production agent deployment.

READ THE BLOG

What the framework gets right, and what most agent guidance misses, is that the data layer choice is foundational. Teams that try to defer it end up rearchitecting halfway through. Teams that pick a unified platform from day one spend their time on prompt engineering and orchestration rather than infrastructure plumbing. The Canvas treats this as a structural decision, not a tooling preference.

The next pieces show systems of action in production, across financial services, payments, supply chain, and lending.

Unlocking financial services document intelligence with agentic AI

Financial services run on documents. Loan files, KYC packets, mandate letters, derivative confirmations, and compliance attestations. The work of extracting structure from those documents has historically been low-leverage manual labor. Agentic AI changes the economics, but only if the data layer can support multi-agent ingestion, extraction, and reconciliation at scale.

The reference architecture for document intelligence uses MongoDB as the unified data foundation for a supervisor multi-agent system. A supervisor agent oversees four specialized child agents that work collaboratively: a Scanner identifies relevant documents, an Evaluator scores them for relevance to the use case, an Extractor pulls content and prepares it for chunking, and a Processor generates embeddings for downstream tasks. A second-tier Document Assistant agent then handles agentic RAG, deciding when and how to retrieve based on the user query. Voyage AI's voyage-context-3 model produces contextualized chunk embeddings that capture both local and global document context, delivering significantly higher retrieval accuracy at lower cost.

The agent memory is the part that quietly does most of the work. Each agent remembers prior interactions, learns from corrections, and builds context over time. That memory has to live somewhere durable, with access controls and audit trails that satisfy financial services governance. MongoDB's role is to be that place: the document, the embeddings, the agent memory, and the audit trail in one coherent layer.

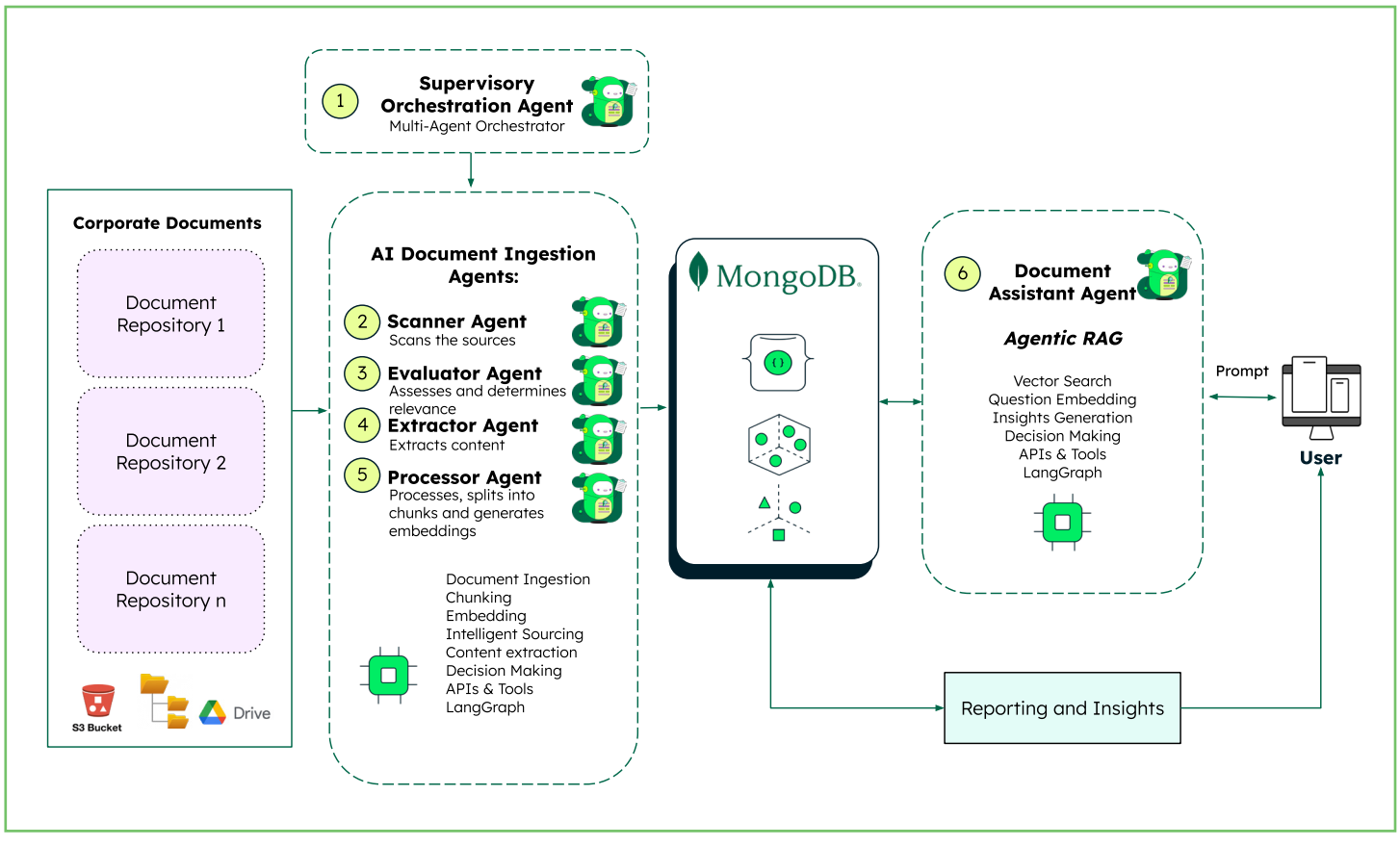


Figure 16.
Intelligent document processing architecture with MongoDB

READ THE ARTICLE

Unlocking agentic power to modernize global payment systems

Payment systems are the part of the financial stack where modernization is hardest and the cost of getting it wrong is highest. The rails that move money were each built for a different era: SWIFT speaks ISO 20022 XML, ACH speaks fixed-width files, card networks speak ISO 8583, and crypto speaks JSON. Most banks wire them together with point-to-point integrations and patch the gaps with human exception handling. Failed payments alone drain the global economy of over $100 billion every year. The result is slow time to market, high cost of ownership, and persistent compliance risk.

The fix isn't to replace the rails. It's to land everything they emit into one canonical payment model: business data such as instructions and parties, operational and AI data such as processing jobs and vector embeddings, and configuration data such as routes and transformation rules. On top of that model, an automation engine with AI/ML, rules, and a built-in audit trail can finally do useful work: routing, reconciliation, fraud triage, exception resolution, and customer communication.

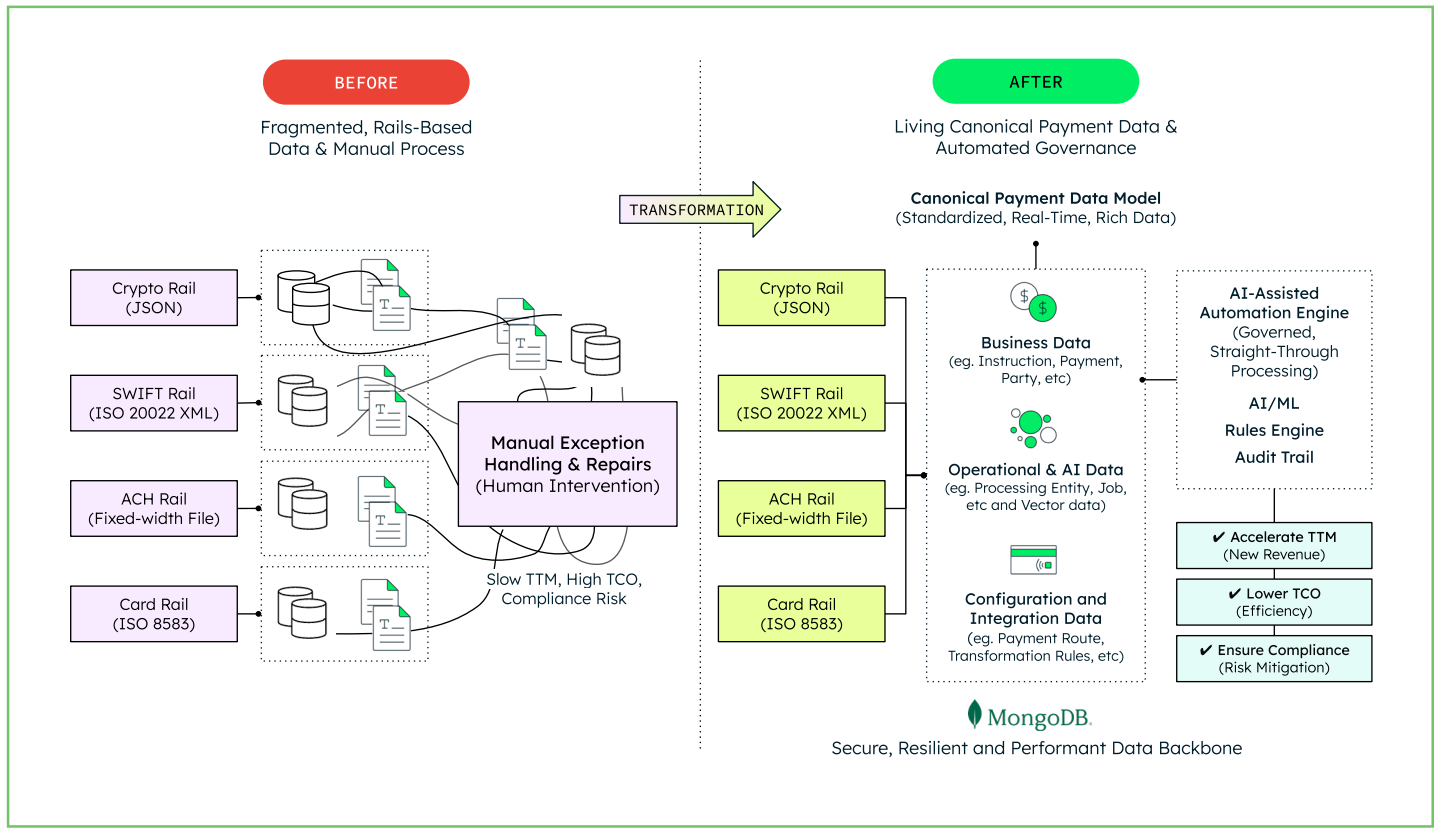


Figure 17.
From legacy rail sprawl to a canonical, agentic payment platform.

The result is faster time to market, lower cost of ownership, and compliance treated as a property of the data rather than a quarterly fire drill. What changes is what the bank can ship.

READ THE BLOG

Multi-agent systems for supply chain disruption management

Supply chain disruption is the canonical multi-agent problem. A weather event in one port, a labor action in another, a quality issue at a tier-two supplier, and a demand spike in a key market all interact. No single human team can hold the full picture in real time. No single agent can either. The cost of getting it wrong is concrete: supply chain disruptions cost businesses an estimated $184 billion annually.

In practice, the work splits into specialist roles. Disruption analysis watches for incidents as they unfold. Supply chain planning models the downstream impact and proposes mitigations. Risk analysis scores the exposure and surfaces what needs escalation. LangGraph (an orchestration framework for multi-agent workflows) routes between them, but the agents' shared understanding of the world lives in the data layer beneath: a single store where shipments, weather, carriers, fleet hubs, and embedded incident reports all sit together.

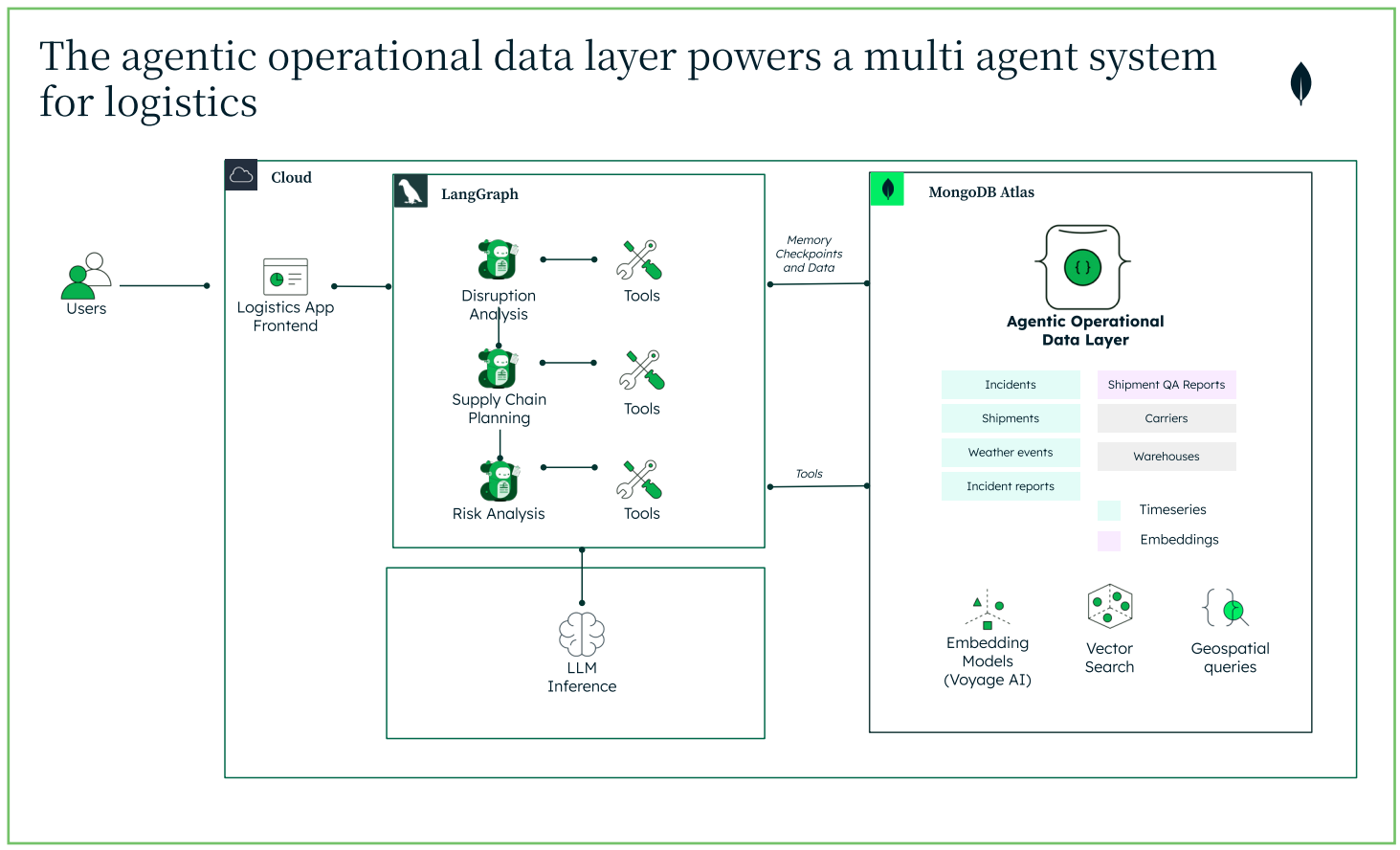


Figure 18.
Solution architecture diagram. Three specialist agents (Disruption Analysis, Supply Chain Planning, Risk Analysis) are orchestrated through LangGraph, with MongoDB Atlas serving as the agentic operational data layer for shipments, weather events, carriers, warehouses, vector embeddings, and geospatial queries.

What MongoDB Atlas adds is more than storage. Weather events stream in as time series. Shipments and warehouses carry coordinates that geospatial queries can reason over. Historical incident reports get embedded with Voyage AI so vector search can surface what the agents need from prior disruptions. And the memory checkpoints LangGraph writes between agent steps live in the same store, which is what makes the postmortem trail accumulate as a structured record rather than a Slack thread.

EXPLORE THE SOLUTION

The modern end-to-end digital lending journey, powered by MongoDB and agentic AI

Lending is the canonical financial workflow that has been promised end-to-end automation for two decades and has mostly delivered partial automation across many silos. The reasons are familiar. The data lives in a dozen systems, the compliance steps require human judgment at multiple stages, and the customer experience is shaped by whichever system happens to own the worst handoff.

Agentic AI changes the bottleneck. A modern lending journey can be designed as a coordinated set of agents handling the full lifecycle from origination to renewal. ***A loan application agent*** acts as the intelligent front door, guiding borrowers, validating and enriching data in real time, and extracting insights from documents to produce a decision-ready loan profile. An ***underwriting agent*** combines business rules, machine learning models, and alternative data to assess creditworthiness continuously, routing exceptions and high-risk cases to a human-in-the-loop reviewer for final sign-off. A ***renewal agent*** monitors the post-funding lifecycle, predicting refinancing, hardship, and churn risk and triggering personalized offers or early intervention. Each agent is bounded, each is observable, and each shares a single view of the application as a business object. The customer sees a coherent experience. The lender sees an audit trail.

The architecture matters because lending is high-judgment work. The data layer determines where automation is safe, where human review is required, and ultimately where the difference lies between a journey that takes minutes and one that takes weeks. The pattern generalizes beyond lending to any document-heavy financial workflow that has to balance speed with regulatory control.

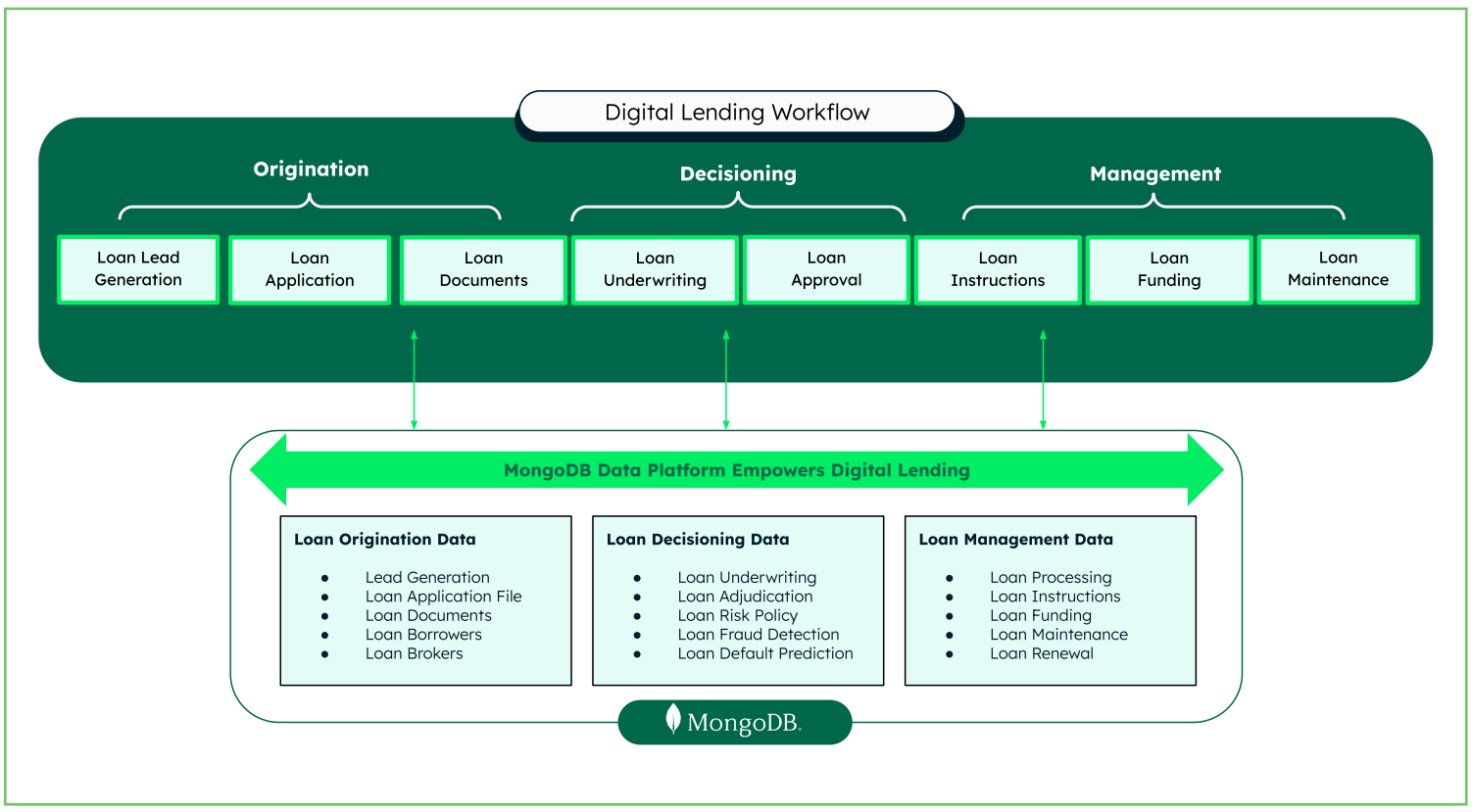


Figure 19.
Agentic AI Workflow–Digital Lending

READ THE ARTICLE

The portability test

The use cases in this section have shown what systems of action look like in the real world: payments that reconcile, supply chains that respond, lending journeys that move from intake to decision, and agents that do more than retrieve information. But once those systems begin to matter, another question arrives. Not just what will the agent do, but where will the business need it to run?

Enterprise agentic systems rarely stay confined to one neat environment. They may span clouds, regions, model platforms, developer tools, operational data, and security boundaries. Some workloads move for latency. Some for data residency. Some because a particular AI service, model, or compute option is available somewhere else. Some because the business itself changes: a new market, a new acquisition, a new regulatory requirement, and a new platform standard.

That makes portability part of the architecture, not an afterthought. If the operational store, vector search, agent memory, and security model have to move separately, the stack fragments again. If they can move together, the system of action can follow the business.

Your data platform moves with you

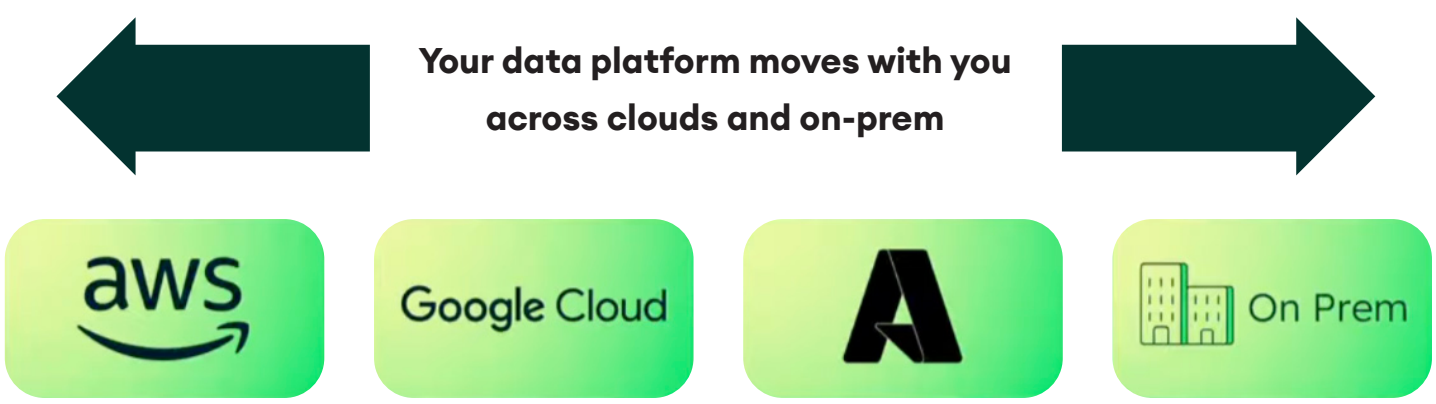
The first decade of cloud was about picking one. The second decade is about running across several. Acquisitions add a cloud the team didn't choose. Regional regulators force workloads to stay in-country on whichever provider has the local zone. The best AI tooling for one workload sits on a different cloud than the best AI tooling for the next. And capacity itself has become a constraint: teams go to their hyperscaler and are told no GPUs are available in the region they need, or no capacity at all.

A pattern is already showing up in production. A frontier model company runs out of GPU capacity on AWS, moves part of the workload to Google Cloud for TPUs, and ends up running its data layer distributed across both clouds. The decision was forced by physics, not preference. The architecture absorbed it because the data layer was already cloud-agnostic.

A single MongoDB Atlas cluster can span AWS, Azure, and Google Cloud at once. The same engine, including full-text and vector search, now runs in MongoDB Enterprise Advanced for on-premises and air-gapped deployments where data residency or sovereignty requirements demand it.

This protects against the most expensive class of cloud mistake: building an AI architecture that works on one cloud and falls apart when the business adds, replaces, or splits across another. 2025 closed with three hyperscaler outages worth naming. Google Cloud in June. AWS US-EAST-1 on October 20. Azure Front Door nine days later. Each took customers offline who had bet on a single provider. Each was survivable for customers whose data layer spanned more than one.

READ THE ARTICLE



MongoDB as the Mandate Ledger for agentic commerce: A2A, AP2, and UCP

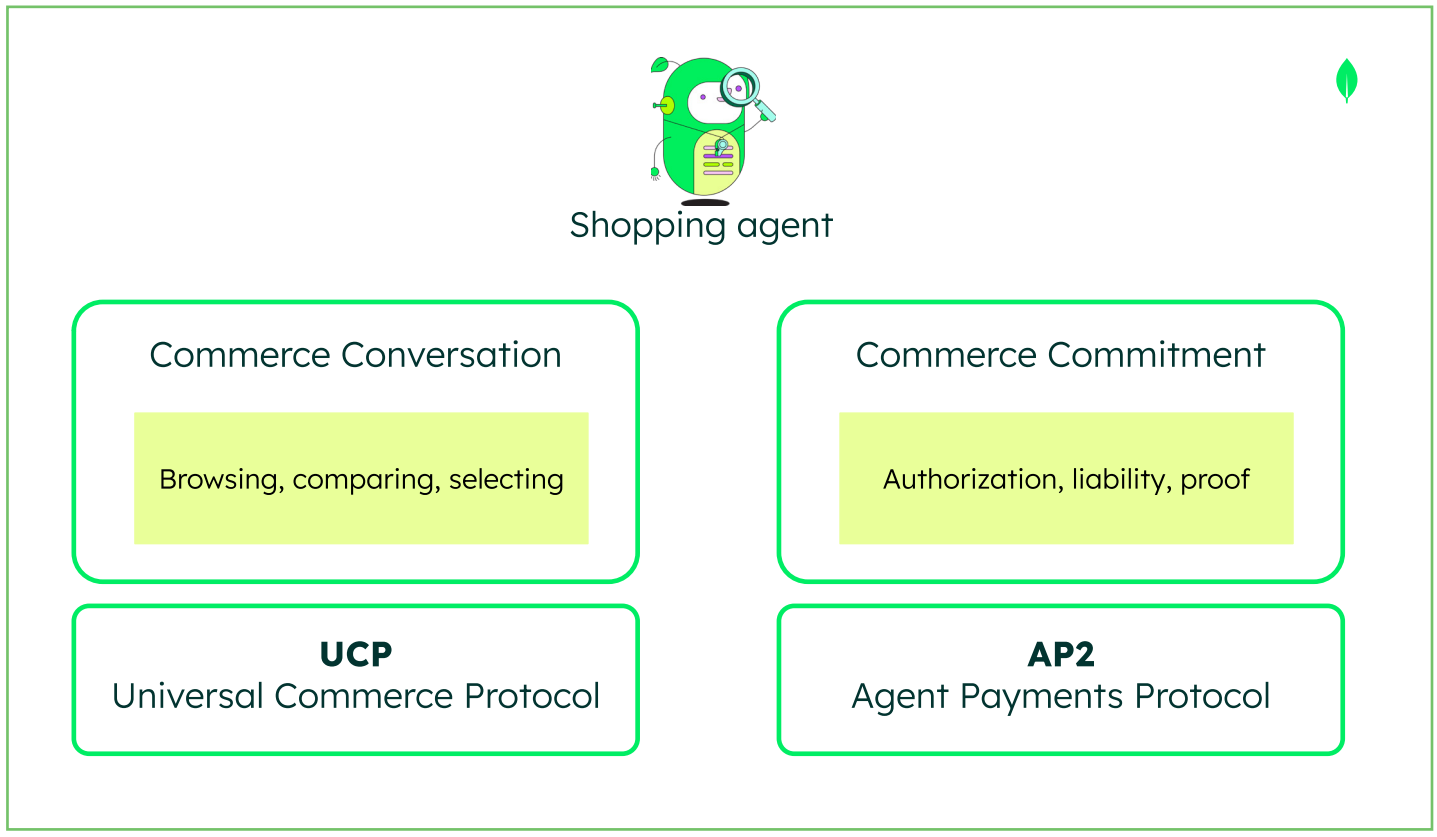
This is the most forward-looking item in the issue, which is why it closes the section. Agentic commerce is the next inflection point for retail. The customer increasingly delegates the purchase decision to an agent. The agent shops on the customer's behalf, evaluates options, completes checkout, and follows up on fulfillment. Three emerging protocols are shaping how that interaction works: Agent-to-Agent communication (A2A), Agent Payments Protocol (AP2), and Universal Commerce Protocol (UCP).

Each of those protocols requires a ledger that captures the customer's authorization, the agent's actions, and the merchant's responses with sufficient detail to support reconciliation, dispute resolution, and audit. MongoDB serves as the ***Mandate Ledger*** for that emerging ecosystem. A mandate is the durable, tamper-proof record of an agentic transaction, and AP2 defines three types: intent mandates (what the agent is allowed to do), cart mandates (what is being purchased), and payment mandates (how and when funds may be transferred). The ledger that holds them has to be queryable across structured and semantic dimensions, append-only at the application layer, and able to absorb new mandate types as the protocols evolve. That is exactly what a unified document store with vector search delivers natively.

For retail teams, the takeaway is that being agent-friendly isn't just an integration problem. It's an architectural opportunity. The brands likeliest to support agent transactions in 2027 are the ones whose ledgers are coherent today.

The arc closes there. We started with the architectural drag that fragmented architectures impose on every project. We end with the architecture that doesn't just absorb the agentic era, but underwrites it.

Figure 20.
UCP conversation vs. AP2 commitment



READ THE ARTICLE

What Will Outlast the Agent

At the beginning of this issue, two teams started the same AI project on the same day. By the end of the quarter, one had shipped. The other had a beautiful demo and no path to production. It's a familiar scene now: the agent works, the slides impress, and the model behaves just long enough for the room to believe. Then the real questions arrive. Who owns the data? Can the answer be audited? What happens when the customer is waiting, the regulator is watching, and the system has to act?

Across these pages, the answer has been the same. McKesson didn't protect patient safety with a clever prompt. LG U+ didn't improve live customer service by keeping intelligence and operations apart. The financial institutions, manufacturers, retailers, and builders in this issue aren't waiting for the perfect model. They are doing something quieter and more durable: They are building the layer that lets intelligence become reliable enough to use.

Not the spectacle of the agent, but the foundation beneath it. The memory that persists. The permissions that follow the action. The operational truth that arrives in time. The audit trail that proves what happened after the fact.

Models will be replaced. Frameworks will be renamed. Protocols will mature.

The winning architectures will be the ones that absorb all of that change without asking the business to start again.

The agentic era will not be powered by agents alone. It will be powered by the data layer that makes them safe to trust.

→ The closing pages of this issue include an architecture book and three Skill Badges for readers ready to start.

Architectures for the Intelligent AI-Ready Enterprise

Practical frameworks for building AI-ready architectures that deliver lasting business value.

About this resource

Hand this to the architect who finishes the issue and asks, fairly, where the rest of it lives.

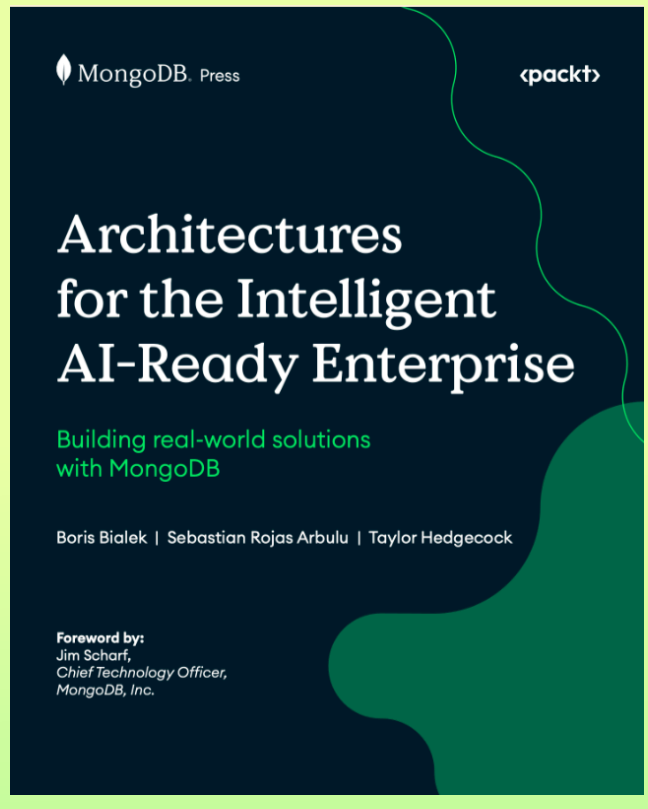
The magazine makes the case for a unified intelligence layer in narrative form. The book is the working architecture behind it: more than 500 pages of patterns, case studies, and chapters, an issue this size has no room for.

25+ AI use cases across 6+ industries. 15+ enterprise case studies, including Novo Nordisk and Base39. The partner implementations with Iguazio by McKinsey & Company, Cognigy, RegData, Fireworks AI, Dataworkz, and Encore that this issue could only excerpt.

Foundational chapters on what a System of Action actually requires, the modernization path that takes legacy systems to AI-ready without starting over, the semantic data protection that lets AI operate in regulated industries, and the agentic and multi-agent patterns that turn an answer into an action.

With a foreword by Jim Scharf, Chief Technology Officer, MongoDB.

Build for what outlasts the model.



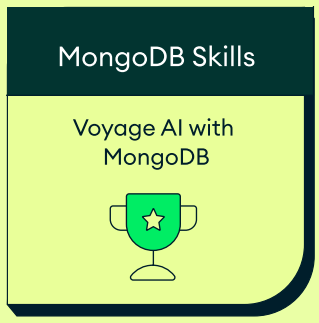
GET THE RESOURCE

READ FREE SAMPLE

Skill Badges

Hands-on credentials that take readers from concept to working implementation. Each badge pairs a focused module with a practical lab, earned in under an hour, useful for the rest of an AI project.

EARN THE BADGE



Voyage AI with MongoDB

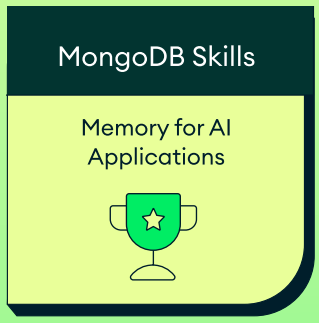
Build AI applications using Voyage AI embeddings stored and searched natively in MongoDB Atlas. Covers embedding models, ingestion, and semantic search fundamentals. The fastest path from “I have heard about embeddings” to “I have shipped a working semantic search.”

EARN THE BADGE

Vector Search Performance

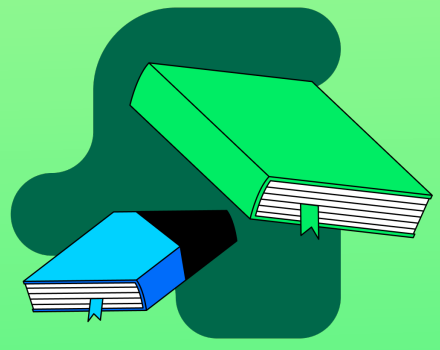
Use search nodes, quantization techniques, and views to scale vector workloads from prototype to production on MongoDB Atlas. The badge that pays for itself the first time you avoid a performance regression in production.

EARN THE BADGE



Memory for AI Applications

Engineer durable, retrievable memory for AI agents. Short-term context, long-term recall, and the patterns that keep multi-agent systems coherent over time. If your agents are forgetful, this is the module to start with.



ADVANCE YOUR CAREER

MongoDB Education helps learners build practical MongoDB skills through free, flexible training, role-based learning paths, skill badges, and certifications. With self-paced content, hands-on labs, and expert instruction, MongoDB University makes it easy to learn, validate skills, and keep growing.

Additional Resources

Where to go next, depending on what you're trying to do.

MongoDB Atlas Architecture Center

Best practices and guidance for designing and operating MongoDB Atlas at enterprise scale. Structured around the Atlas Well-Architected Framework: security, operational efficiency, performance, reliability, and cost optimization. The reference architects and platform engineers reach for when they need a cohesive blueprint.

VISIT THE CENTER

Customer Success Stories

Beyond McKesson, LG U+, and the others featured in this issue, MongoDB publishes new customer success stories every month. The full library is searchable by industry, use case, and outcome. A useful starting point when you're building a business case internally and need a reference customer in your sector.

BROWSE CUSTOMER STORIES

AI Learning Hub

A curated path of tutorials, reference implementations, and skill badges for developers building AI applications on MongoDB. Organized by skill level, from "I just want to ship a RAG prototype" to "I'm designing a multi-agent platform for production."

START LEARNING

Want to learn more?

Contact us.

MongoDB