O'REILLY®
Report

Vector Databases for Enterprise AI

Semantic Retrieval Systems for RAG, Search, and AI Applications

Emma McGrattan

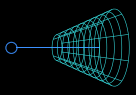
ACTIAN™
a division of HCLSoftware

Actian VectorAI DB

The vector database built for local AI and beyond

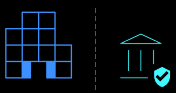
No cloud dependency
Run vector search locally, including fully offline environments.

Low-latency, semantic retrieval of context
Eliminate network round trips. Fast, predictable performance across environments.



Build once, deploy anywhere
Same APIs from laptop to edge and on-prem production. Developer-first from day one.

Private embeddings, full control
Keep your data, embeddings, and retrieval pipeline inside your infrastructure.



Enterprise-ready by design
Secure, reliable, and compliant with GDPR, HIPAA, and residency requirements.

Works with your existing AI stack
Integrate with leading frameworks, models, and infrastructure platforms.

Build AI systems that perform in real-world environments—not just the cloud.

Get started at actian.com/vectorai-db

Vector Databases for Enterprise AI

Semantic Retrieval Systems for RAG, Search, and AI Applications

Emma McGrattan

O'REILLY®

Vector Databases for Enterprise AI
by Emma McGrattan

Published by O'Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Aaron Black
Development Editor: Gary O'Brien
Production Editor: Kristen Brown
Copyeditor: nSight, Inc.
Cover Designer: Susan Brown
Cover Illustrator: Susan Brown
Interior Designer: David Futato
Interior Illustrator: Kate Dullea

May 2026: First Edition

Revision History for the First Edition
2026-04-27: First Release

This work is part of a collaboration between O'Reilly and Actian. See our statement of editorial independence.

979-8-341-67360-1

[LSI]

Table of Contents

Preface

Enterprise data platforms are shaped by who and what consumes data. For most of their history, that consumer was human. Analysts wrote SQL queries, applications executed deterministic transactions, and dashboards reflected predefined metrics. The systems we built, including databases, search engines, and pipelines, were optimized for precision, predictability, and structure. Those assumptions held for decades, and they still matter today.

The adoption of large language models (LLMs) and AI-driven applications introduces a different kind of consumer. Instead of asking precise questions, these systems retrieve information probabilistically and reason over relevance rather than correctness. Techniques such as retrieval-augmented generation (RAG) and semantic search depend on similarity-based retrieval across both structured and unstructured data. This shift does not make traditional databases obsolete, but it does expose clear limits to how they support semantic access to information.

Vector databases are an architectural response to this shift. By storing and retrieving embeddings (numerical representations that capture the meaning of text, images, or other data), vector databases enable AI systems to find relevant data without relying solely on schema or exact matches. In practice, however, many organizations encounter vector databases through isolated experiments or developer tooling. These efforts are often disconnected from enterprise data platforms, governance practices, and operational expectations. As a result, teams struggle to move from promising prototypes to systems that can be trusted in production.

Who This Report Is For

This report is written for data engineers, architects, and technical leaders who are navigating that transition. It focuses on when vector databases are appropriate, how they differ from existing data systems in ways that matter architecturally, and how they can be integrated with relational data, metadata, and governance frameworks already in place. The goal is not to catalog tools but to help readers understand the trade-offs that shape real-world deployments, including performance, cost, accuracy, and operational complexity.

Scope and Perspective of This Report

Throughout the report, vector databases are treated as a complementary capability within enterprise architectures rather than as a replacement for existing platforms. The discussion covers embedding pipelines, similarity search, integration patterns, and the governance considerations that arise when AI systems become primary consumers of enterprise data.

Vector databases reflect a deeper shift in how systems access and reason over data. Understanding that shift, and its architectural implications, is now part of the core skill set for those designing and operating enterprise data platforms. This report aims to provide the clarity and practical guidance needed to approach semantic retrieval with confidence and rigor.

Why This Shift Matters Now

From my experience building and operating database and data management platforms, the most important architectural shifts are rarely about adopting a single new technology. They are about recognizing when long-standing assumptions no longer hold and then adjusting foundations accordingly. Vector databases are one such shift. They reflect a change in how systems consume data, not just in how data is stored or queried. Approached thoughtfully, they can become a durable part of the enterprise data platform. Approached casually, they risk becoming another disconnected experiment. My hope is that this report helps readers make those distinctions early and design systems that will hold up as AI moves from exploration into everyday operation.

O'Reilly Online Learning

For more than 40 years, O'Reilly Media has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

Acknowledgments

This report was made possible through the collaboration and support of many individuals. Special thanks to my colleagues at Actian who contributed their insights, challenged assumptions, and provided real-world perspectives that shaped the content.

I would like to particularly thank Steffen Kläbe, whose thoughtful reviews, detailed feedback, and deep technical perspective significantly strengthened the clarity and rigor of this report. His willingness to challenge assumptions and refine key sections helped ensure the material reflects both practical experience and technical accuracy.

I am truly grateful to Anthony Corbeaux and Jennifer Jackson for their sponsorship and guidance, and to the extended CTO Office and marketing teams for helping refine the vision for this work.

CHAPTER 1
Why Vector Databases Matter Now

Vector databases are often introduced to enterprises as yet another component in an already crowded data landscape. The term appears in vendor roadmaps, AI reference architectures, and internal experiments built by enthusiastic teams. What is less clear, especially for decision makers, is why a new kind of database is needed at all and what problem it really solves beyond "better search." Before discussing mechanics, it is useful to step back and look at how data access is changing as AI systems become primary consumers rather than occasional clients.

This chapter focuses on that shift in access patterns. It traces the path from traditional keyword-based search and relational queries to semantic retrieval, explains where existing systems begin to strain, and introduces vector databases as an architectural response rather than a point solution. It then examines the practical questions that follow for enterprise teams: what changes when retrieval is based on meaning rather than exact matches; how model choice starts to influence system behavior; and whether vector capabilities should live in standalone systems or be integrated into existing platforms. The goal is to give readers a clear, shared understanding of why vector databases matter now, so that subsequent chapters on mechanics, pipelines, and governance have a concrete context.

From Keyword Search to Semantic Retrieval

Enterprise data systems have always reflected the dominant ways in which data is accessed. For years, those access patterns were stable. Applications relied on deterministic transactions. Analysts queried structured data using well-defined schemas. Search systems focused on keywords, rankings, and relevance tuned for human interpretation. These systems worked well because the questions being asked were explicit and the answers were expected to be exact.

AI-driven applications introduce a different access pattern. Large language models (LLMs), retrieval-augmented generation (RAG) pipelines, and agent-based systems do not look for exact matches. They look for information that is relevant in context. They operate across structured and unstructured data. They tolerate ambiguity and prioritize usefulness over precision. This change in consumption exposes limitations in traditional databases and search technologies that were designed under different assumptions. Figure 1-1 illustrates this shift from deterministic keyword-based access to context-aware semantic retrieval.

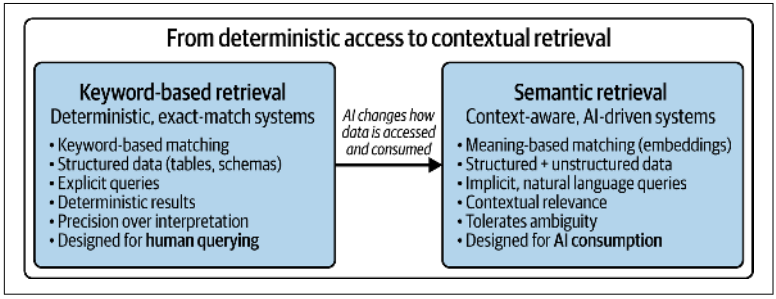


Figure 1-1. The shift from keyword-based to semantic retrieval moves from exact matches to context-aware relevance across structured and unstructured data.

The Limits of Traditional Databases and Search

Relational databases excel at enforcing structure, consistency, and correctness. They are optimized for well-defined schemas and queries where the meaning of each column and value is explicit. Keyword-based search systems are optimized for text retrieval where relevance is inferred from term frequency, ranking

algorithms, and heuristics tuned for human users. Both approaches are highly effective within their intended domains.

Semantic retrieval breaks those assumptions by changing both how questions are asked and how relevance is determined. Instead of matching exact terms or predefined fields, semantic retrieval focuses on meaning. Data is retrieved based on conceptual similarity between a query and available information, even when the same words are not used. When an AI system requests information related to a concept rather than a specific term or seeks supporting context rather than a single correct result, exact matching becomes a constraint rather than a strength.

Keyword-based search struggles to capture meaning across different word choices, rephrasing, and implied intent, particularly when relevant information is expressed in a different language than the query. Relational queries depend on predefined schemas and explicit joins, which are often impossible or impractical when working with large volumes of unstructured or semistructured data. These constraints become especially apparent in RAG workflows, where the relevance, diversity, and coherence of retrieved context directly influence the accuracy and usefulness of generated outputs.

The issue is not that existing systems are inadequate. It is that they were never designed to answer questions framed in terms of meaning rather than structure.

Embeddings and the Shift to Similarity-Based Retrieval

Vector databases address this gap by introducing a fundamentally different retrieval model. Rather than operating on exact values, predefined fields, or keyword matches, vector databases store data as embeddings, which are numerical representations designed to capture semantic meaning. These embeddings place related concepts closer together in a high-dimensional space, allowing systems to retrieve information based on similarity of meaning rather than exact correspondence. Similarity is determined using distance functions that quantify how close two embeddings are in the vector space, which replaces the notion of equality or keyword ranking with a measure of semantic proximity. Retrieval, in this model, is

about identifying the most relevant information rather than the most exact match.

This shift has significant architectural implications. Similarity-based retrieval is inherently probabilistic, meaning that results are ranked by how close they are to a query in semantic space, not by whether they are strictly correct or incorrect. Query precision varies based on thresholds, indexing strategies, and model behavior. Performance characteristics differ from those of traditional queries as systems are optimized for approximate matching across large, high-dimensional datasets. Storage and indexing strategies are therefore specialized to balance retrieval quality, latency, and resource consumption.

For enterprise teams, this often marks the first time retrieval behavior is shaped as much by model choice as by data modeling. Decisions about which embedding model to use, how embeddings are generated, and how a given embedding model is evaluated directly influence retrieval quality and system behavior. As a result, model selection becomes an architectural concern rather than an implementation detail, and understanding this dependency is essential for building reliable semantic retrieval systems. Readers interested in deeper exploration of embedding model evaluation can consult the article "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks" by Nils Reimers and Iryna Gurevych (2019), the Massive Text Embedding Benchmark (MTEB), and recent retrieval-focused embedding research such as OpenAI's text embedding models and related literature.

Standalone Vector Databases and/or Integrated Platforms?

As vector capabilities mature, enterprises face an architectural question that is difficult to avoid. Should vector search be deployed as a standalone system, or should vector capabilities be integrated directly into existing data platforms?

Standalone vector databases offer clear advantages. They allow independent scaling, specialized indexing, and rapid iteration on embedding and retrieval strategies. Integrated platforms reduce operational complexity and simplify access to relational data and metadata. In practice, many organizations adopt a hybrid approach,

using standalone vector systems while working toward tighter integration with the organization's core data platforms.

This report does not advocate a single correct answer because there is none. There are only architectural choices, each with benefits and challenges. Instead, it treats this choice as a context-dependent decision shaped by workload characteristics, governance requirements, and operational maturity.

Decision Checklist: Standalone Vector Database or Integrated Vector Capabilities

This checklist is intended as a set of signals, not strict rules. The items on each side are not mutually exclusive, and most organizations will see themselves in several points on both lists. If multiple items on one side resonate strongly with your current or target state, that is a good indicator of where to lean, but it should inform the decision rather than decide it outright.

Consider a standalone vector database when:

- ☐ Semantic retrieval workloads require independent scaling from transactional or analytical systems.
- ☐ Latency and recall requirements are tightly coupled to approximate nearest neighbor (ANN) indexing and vector-specific storage.
- ☐ The organization expects rapid iteration on embedding models and retrieval strategies.
- ☐ Vector workloads are AI-first and not secondary to relational queries.

Consider integrated vector capabilities when:

- ☐ Strong transactional consistency between relational and vector data is required.
- ☐ Metadata and relational joins are central to retrieval logic.
- ☐ Operational simplicity outweighs fine-grained performance tuning.
- ☐ Vector workloads are incremental extensions of existing data platforms.

In practice, many teams will satisfy criteria on both sides and still adopt a hybrid approach, using standalone vector systems for specific workloads while moving toward deeper integration as the platform matures.

Vector Databases as Foundational Infrastructure

The most costly mistakes happen when this decision is made implicitly rather than deliberately. One risk in early vector database adoption is treating them as experimental add-ons. When deployed in isolation, without integration into enterprise data platforms, governance models, and operational processes, they tend to remain prototypes. The real value of vector databases emerges when they are treated as infrastructure that supports production workloads and shared data access patterns.

This requires thinking beyond retrieval speed or benchmark performance. It requires understanding how embeddings are created and maintained, how vector data relates back to source systems, how access is governed, and how retrieval quality is monitored over time. These concerns are familiar to anyone who has operated enterprise data systems, but these concerns surface in new ways when similarity-based retrieval becomes central.

The rest of this report builds on this foundation. It explains how vector databases work, how they are integrated into enterprise architectures, and how organizations can adopt these vector databases responsibly. The goal is not to replace existing systems but to extend enterprise data platforms to support a new and increasingly important mode of data access.

The hardest part is not adopting vector databases; it is recognizing that the questions AI systems ask of data no longer fit the assumptions the enterprise platforms were built on, namely explicit queries, predefined schemas, deterministic joins, and relevance based on exact matches. Semantic retrieval replaces those assumptions with meaning, probability, and contextual relevance, which is why new architectural foundations are required.

Lessons I've Learned

From my experience, the hardest part of adopting new data infrastructure is not learning how the technology works but recognizing when long-standing design assumptions no longer apply. Vector databases are often introduced as tools for AI teams, yet their real impact is architectural. They change how relevance is defined, how retrieval is evaluated, and how data systems are expected to behave under uncertainty. Teams that treat this shift deliberately can extend their existing platforms in durable ways. Teams that treat it casually risk repeating the same pattern of disconnected systems that data engineering has spent years trying to undo.

CHAPTER 2

How Vector Retrieval Behaves at Query Time

A vector database returns results based on similarity rather than exact matches. If you search for "laptop overheating," the system retrieves documents about thermal throttling, cooling performance, or temperature management, even if those exact words never appear in the query. This happens because the system compares representations of meaning rather than literal text.

This introduces a fundamentally different retrieval model than traditional databases. In SQL, you define conditions, and the system returns the rows that meet those conditions. The logic is explicit and deterministic. In a vector database, you are describing what you are looking for, and the system returns the closest semantic matches ranked by relevance. The outcome depends on how meaning is encoded, how similarity is measured, and how the system is tuned for recall and latency.

For enterprise workloads, this difference becomes visible in practice. The database is not just retrieving data; it is shaping which information is surfaced to applications, analysts, and AI systems. Two queries that appear similar can produce slightly different rankings. Small changes in embeddings, thresholds, or filters can shift what is retrieved and how it is ordered.

Understanding how vector databases work, therefore, starts with observable retrieval behavior and then moves to the mechanisms that produce it.

What Good Retrieval Looks Like

In a well-functioning system, semantic retrieval feels intuitive. A query such as "customer churn risk" should surface documents about retention analysis, cancellation trends, and predictive churn models, even if the original documents never use the exact phrase "churn risk." The results may use different terminology but still align with the intent of the query.

This is especially valuable in enterprise environments where terminology varies across teams. A finance team might refer to "revenue leakage," while a product team discusses "subscription drop-off." A vector database can recognize the semantic relationship and surface relevant material across both vocabularies.

Good retrieval is not about exact wording. It is about consistently surfacing contextually relevant information across heterogeneous data.

What Degraded Retrieval Looks Like

When retrieval degrades, the system does not fail loudly. Instead, it becomes subtly misaligned. A query for "quarterly revenue decline" might return documents about cost optimization or general business performance because the embedding model clusters broad business concepts too closely. The results are not random, but they are less precise than expected. Other failure patterns include missing obvious matches due to overly strict similarity thresholds or unstable rankings where small phrasing changes reorder the top results. For example, "database latency spike" and "sudden query slowdown" may retrieve overlapping but differently ranked documents, even though they refer to the same operational issue.

These behaviors are often blamed on the database itself. In practice, the root cause may lie in how embeddings were generated, how data was segmented, or how similarity thresholds were configured. This is why observability and tuning are core parts of operating vector systems, not optional enhancements.

From Query to Embedding

Before a vector database can compare a query to stored data, the query must be converted into an embedding. As discussed in Chapter 1, an embedding is a numerical vector that represents the meaning of the text rather than the literal words. Figure 2-1 illustrates this process, showing how both stored content and incoming queries are transformed into embeddings so they can be compared within the same semantic space.

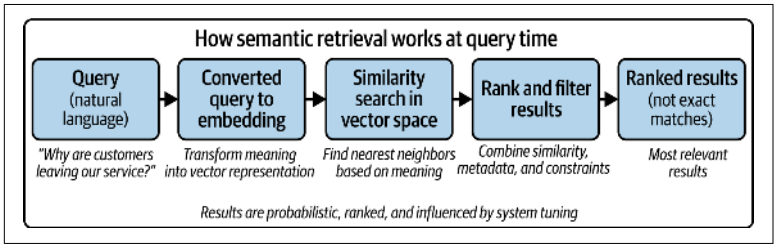


Figure 2-1. At query time, semantic retrieval converts queries into embeddings and retrieves results based on similarity, returning ranked outputs based on contextual relevance.

A simple way to visualize this is with a toy example. Suppose a system represents meaning in three dimensions:

- "laptop overheating" → [0.82, –0.14, 0.67]
- "thermal throttling issue" → [0.79, –0.10, 0.71]

The specific values do not matter. What matters is that these vectors are close to each other in space, which signals semantic similarity despite different wording.

In production systems, embeddings are not three-dimensional but hundreds or thousands of dimensions. Dimensionality refers to the number of numerical values in each vector. A 768-dimensional embedding is simply a list of 768 numbers that together encode meaning. Higher-dimensional embeddings can capture more nuance across language, domain concepts, and context, but they also increase storage footprint, memory usage, and compute cost during retrieval.

This transformation is applied consistently to both stored data and incoming queries. Documents, records, or chunks of text are embedded during ingestion and stored as vectors. At query time, the

system embeds the query using the same model or a compatible one, then compares the resulting vector against the stored vectors.

Three things determine whether retrieval works reliably. First, the embedding model defines the semantic space in which all comparisons occur. If the model does not understand domain-specific terminology, retrieval will appear inconsistent regardless of how the database is tuned. In this context, tuning refers to adjustments such as similarity thresholds, index parameters, or retrieval depth that control how many results are returned and how aggressively the system searches for neighbors in the vector space. Even when these settings are optimized, a poorly aligned embedding model will still produce weak retrieval. Second, consistency between ingestion and query embeddings is critical. Changing embedding models or model versions can shift the geometry of the vector space and degrade retrieval quality even when the underlying data has not changed. Third, the chunking strategy has a direct impact on retrieval performance. Embedding models operate within fixed input limits, which means large documents, images, audio, or other artifacts must be segmented into chunks before embedding. If chunks are too small, they lose semantic coherence. If they are too large, they dilute meaning and produce less precise embeddings. Effective chunking balances semantic completeness with embedding constraints, ensuring that each chunk represents a distinct, retrievable concept rather than a blurred aggregation of topics.

How Similarity Is Calculated

Once the query is converted into an embedding, the database compares it to stored vectors using a similarity metric. Common approaches include cosine similarity, dot product, and Euclidean distance. Cosine similarity is often preferred when the goal is to compare semantic direction regardless of vector magnitude, dot product is commonly used with normalized embeddings and optimized hardware implementations, and Euclidean distance measures absolute distance in the vector space and is sometimes used in systems that rely on geometric proximity rather than directional similarity. These metrics measure how close two vectors are in semantic space, which determines how results are ranked.

At a small scale, a system could compare the query vector to every stored vector. At enterprise scale, this is not practical. Collections

may contain millions or billions of embeddings, and exhaustive comparison would introduce unacceptable latency and cost.

Instead, vector databases use approximate nearest neighbor (ANN) search to find the nearest semantic neighbors quickly without evaluating every possible comparison. ANN algorithms trade a small amount of precision for dramatically improved speed and scalability. Approximate does not mean random. The system is still identifying the closest vectors in the embedding space, but it does so without checking every vector. In practice, this means the database may occasionally miss a lower-ranked relevant result while still correctly identifying the most relevant matches at the top of the list.

These similarity scores are then used to produce a ranked list of results, which is how the system determines what is surfaced first.

Indexing as a Mental Model

Rather than scanning every vector, the database relies on specialized indexes that organize vectors so that nearby points in semantic space can be located efficiently. A useful mental model is to navigate a map rather than check every possible location. The system follows paths through the index to reach regions where similar vectors are likely to be found.

Common indexing structures include hierarchical graph-based approaches such as hierarchical navigable small world (HNSW) and partition-based methods such as inverted file index (IVF). While the underlying algorithms differ, their goal is the same: to quickly identify promising regions of the vector space so the system can locate the nearest semantic neighbors without scanning every vector.

From a user perspective, this shows up as a balance between speed and retrieval accuracy. Tuning index parameters creates a direct trade-off: tighter search improves recall but increases latency, while looser search speeds up queries but may miss some relevant results. Most of this complexity should be managed by the platform, but teams still need a basic mental model to understand why results may vary under different performance settings.

Ranking, Thresholds, and Result Interpretation

Similarity calculations do not produce a single answer. They produce a ranked list of results ordered by semantic closeness. The system must then decide how many results to return and what threshold qualifies as "similar enough" for the application.

Because similarity search produces ranked results rather than exact matches, interpretation becomes part of system design. The system is identifying the closest matches in vector space, not returning a single deterministic answer. Small changes in thresholds, embeddings, or filters can shift rankings or surface different but still relevant items.

For retrieval-augmented generation (RAG) and AI-driven applications, these decisions directly affect output quality. Returning loosely related context can dilute answers, while overly strict thresholds may omit critical supporting information.

Hybrid Retrieval and Filtering in Practice

In enterprise systems, semantic similarity alone is rarely sufficient. Queries must also respect structured constraints such as access permissions, time ranges, regulatory boundaries, and business context.

In practice, hybrid retrieval combines vector similarity with structured filtering over metadata and relational fields. For example, a query such as "customer churn drivers" may first retrieve semantically similar documents and then apply filters like `country = "Spain"`, `year = 2025`, or `region IN ("Munich", "Amsterdam")`. Time-based filtering is simply one form of structured filtering applied to temporal attributes.

This pattern is critical in enterprise environments where relevance is not defined by meaning alone. A document may be semantically similar yet operationally irrelevant if it falls outside the correct jurisdiction, time window, customer segment, or business unit.

Observability and Tuning

When retrieval behavior appears inconsistent, the underlying cause is rarely obvious without visibility into system signals. Useful observability includes similarity scores, retrieved sources, ranking order, and latency patterns.

These signals support specific operational decisions rather than abstract monitoring. For example, if clearly relevant documents consistently receive low similarity scores, the issue may lie in the embedding model rather than the index or thresholds. A general-purpose model may not recognize domain terminology such as medical abbreviations, financial metrics, or legal phrases, suggesting the need for a domain-adapted embedding model or improved data segmentation.

If relevant documents appear but rank below position 10, widening the retrieval depth from top 5 to top 20 may surface critical context that's being truncated. If rankings shift significantly, verify whether the embedding model version changed. Even minor model updates can alter vector geometry enough to reorder results.

Observability therefore turns semantic retrieval into a tunable system. Instead of guessing why results appear misaligned, teams can trace issues back to embeddings, thresholds, filtering logic, or data preparation and make targeted adjustments based on evidence.

In production systems, retrieval quality rarely fails due to a single component. It degrades through the interaction of embeddings, thresholds, filters, and data preparation decisions.

Lessons I've Learned

In practice, the most common mistake is treating vector databases as a faster form of search rather than a different form of retrieval. What matters is not just how quickly results are returned but how relevance is defined, ranked, and constrained under real workload conditions. When teams focus on observable retrieval behavior, embedding alignment, and tuning thresholds instead of isolated algorithm details, they are far better positioned to deploy vector databases reliably within enterprise and RAG architectures.

CHAPTER 3

The Vector Pipeline in RAG Systems

While Chapter 2 examined how vector databases behave at query time, this chapter examines what happens after retrieval becomes part of an end-to-end system that must select, filter, and assemble evidence before generation.

To ground this, consider a user asking, "Why did customer churn increase last quarter?" In a retrieval-augmented generation (RAG) system, answering this question typically occurs in two stages. First, the query is converted into an embedding and relevant documents are retrieved from a vector database. Second, the retrieved context is passed to a large language model (LLM), which generates the response using that material as evidence.

During the retrieval stage, the query is embedded, relevant documents are retrieved from the vector database, filters such as permissions and time scope are applied, and a reranking step may reorder the candidates to prioritize the most relevant passages before a bounded context set is assembled. Only after this context is prepared does the LLM generate a response grounded in the retrieved material.

Figure 3-1 illustrates this retrieval and context assembly pipeline, showing how a user query is embedded and how candidate documents are retrieved from the vector database, filtered and reranked, and finally assembled into the bounded context that the LLM uses to generate its response.

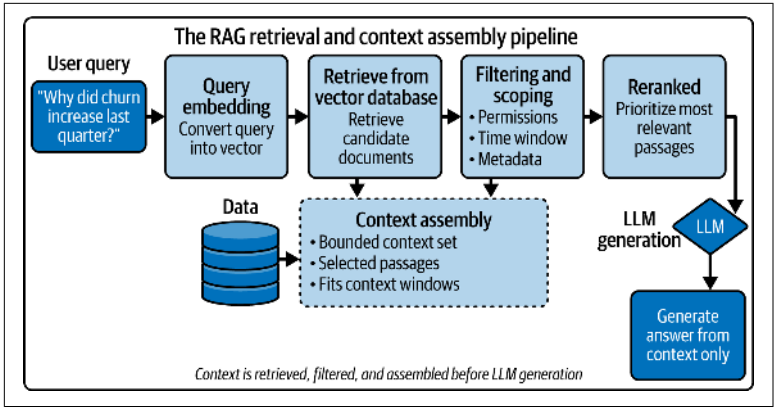


Figure 3-1. In a RAG system, context is retrieved, filtered, and assembled before generation. The model produces answers based only on this selected context, not the full dataset.

This pipeline has a critical implication. The LLM does not reason over the entire enterprise dataset because it operates within a finite context window and can only process the information that is provided to it at generation time. Instead, it reasons over the subset of information selected by the retrieval process. As a result, the reliability, specificity, and auditability of the final answer are determined primarily by how context is retrieved, scoped, and prioritized. If the assembled context is incomplete, outdated, or overly generic, the response will reflect those limitations regardless of model sophistication.

What Good RAG Retrieval Looks Like

When retrieval is well aligned with the question, the assembled context is both relevant and sufficiently specific. For a query about churn increasing last quarter, the system should surface the quarterly churn analysis, regional retention dashboards, recent customer exit summaries, and incident logs tied to the relevant time window. The generated answer is grounded, specific, and explainable because the retrieved sources directly address the question rather than merely relating to the general topic of churn.

In this scenario, the model can point to concrete drivers such as a regional service outage, a pricing change, or a renewal policy shift because those details are present in the retrieved context. The

strength of the answer is a direct consequence of the strength of the context selection.

What Degraded RAG Retrieval Looks Like

RAG failures are subtle. The system still retrieves context and generates a fluent answer, but the grounding (the process of anchoring the model's response in retrieved source material) is weak. For the same churn query, a degraded pipeline might retrieve an annual revenue report, a generic retention strategy document, and an industry benchmark on customer loyalty. All of these documents are semantically related to business performance, yet none explain why churn increased in the specific quarter being analyzed.

The generated answer will often sound reasonable but unhelpful. It might say, "Customer churn can be influenced by pricing changes, competitive pressure, and service quality issues." This response is technically accurate yet operationally useless. The retrieved context contained only general retention theory rather than the specific regional outage reports, contract renewal failures, and support ticket spikes that actually drove last quarter's increase. The failure is not in the model's reasoning but in the specificity of the context it was given.

Chunking and Context Assembly

Before documents can be retrieved, they must be segmented into retrievable units (chunks). This decision directly determines what evidence the model is able to see.

Storing an entire 50-page report as a single chunk allows the system to retrieve it as broadly relevant, but the context passed to the model will be coarse and unfocused. Splitting that same report into paragraph-level chunks enables the pipeline to surface the exact section that explains churn drivers, but it risks separating supporting evidence across multiple fragments.

Chunk size therefore shapes the balance between precision and coherence. Smaller chunks improve targeting but can fragment causal context. Larger chunks preserve narrative continuity but reduce retrieval granularity. Fact-based queries that require pinpoint references tend to benefit from smaller, precise chunks, while analytical questions that require causal reasoning often perform

better when context is preserved in larger, semantically coherent segments.

Retrieval Depth and Generation Quality

Retrieval depth, meaning how many documents are passed into the context bundle (the set of retrieved passages provided to the language model as context), directly influences the quality of generated answers. A shallow retrieval that returns only the top few results may be precise but incomplete. Increasing depth can improve coverage of relevant evidence, but it can also introduce loosely related material that dilutes the signal.

For example, a query about regional sales decline may produce stronger answers when the system retrieves a focused set of regional performance reports rather than a larger collection of globally related sales documents. More context does not automatically produce better responses. Excessive context can introduce noise, contradictions, or generalized themes that the model must reconcile during generation.

Effective pipelines tune retrieval depth based on the query's specificity and the structure of the underlying documents rather than defaulting to a fixed number of results.

Context Window Constraints

RAG systems operate within the context window limits of the language model. Retrieved documents cannot simply be passed wholesale into generation. They must be ranked, truncated, or selectively included so that the most relevant evidence fits within the available context space.

Although modern models support increasingly large context windows, larger contexts do not automatically improve outcomes. As more material is added, the model's attention can diffuse across the context, making it harder to consistently focus on the most relevant passages.

Context assembly therefore becomes a prioritization problem: which documents provide the most decision-relevant evidence, and which must be excluded when the context window limit is reached? If a critical document is ranked slightly lower and excluded due to

space constraints, the generated answer may omit the most important supporting evidence while still appearing coherent.

This constraint makes ranking policies and context selection logic as important as retrieval itself. In practice, designing how context is selected, prioritized, and structured becomes a core part of building reliable RAG systems.

RAG-Specific Failure Modes

RAG pipelines introduce failure patterns that are distinct from traditional search systems because generation depends on the assembled context rather than a single retrieved document.

One common failure occurs when complementary evidence is split across multiple chunks that are not retrieved together. A policy change may be captured in one document while its business impact is described in another. If only the policy fragment is retrieved, the model lacks the causal linkage needed to produce a definitive explanation.

Another failure pattern involves contradictory context. If outdated guidance is retrieved alongside newer corrective documentation, the model may hedge or produce an equivocal response instead of a clear conclusion. The answer appears cautious rather than wrong, which makes the issue harder to detect.

Context dilution is another frequent issue. When the pipeline retrieves too many loosely related documents, the model may generate a generalized answer that reflects the average theme of the context rather than the specific question. The system appears knowledgeable, but the response lacks precision and actionable insight.

These are not generation failures. They are context assembly failures.

Evaluation and Feedback Loops

Evaluating a RAG system requires inspecting both the retrieved context and the generated output. When answers appear vague or incomplete, the first diagnostic step is to examine what evidence was actually retrieved.

If retrieved documents are relevant but incomplete, such as surfacing only an executive summary while the detailed appendix contains the key metrics, increasing retrieval depth is often the correct adjustment. If retrieved passages are relevant but lack surrounding context—for example, a single paragraph referencing a pricing change without its rationale or impact analysis—larger chunk sizes or overlapping chunking windows (where adjacent chunks share some of the same text so that important context is not split across boundaries) can improve coherence.

If the system consistently retrieves broadly related but nonspecific materials, similarity thresholds and filtering logic should be refined to prioritize time-bound and scoped sources. When similar queries produce inconsistent context bundles, the root cause often lies in segmentation instability, embedding updates, or threshold configuration rather than model performance.

Effective evaluation links observable symptoms to specific pipeline adjustments instead of treating retrieval as a black box.

Embedding Refresh and Controlled Change

As enterprise datasets evolve and embedding models improve, organizations often regenerate embeddings to improve semantic alignment. However, re-embedding an entire corpus can shift ranking behavior across the system. Documents that were previously top ranked may move lower even though their content has not changed, simply because the geometry of the vector space has shifted.

Controlled rollout strategies reduce this risk. Teams commonly version embeddings, compare retrieval outputs before and after model updates, and reindex incrementally rather than replacing the entire corpus at once. This allows improvements in semantic representation without introducing unexpected regressions in retrieval behavior.

Auditability and Governance in RAG Systems

In enterprise environments, retrieval must be explainable and auditable. It is not sufficient for a system to generate a plausible answer. Organizations must be able to trace which documents were retrieved, why they were selected, and whether access controls were correctly enforced during context assembly.

This requirement is especially critical in regulated domains such as finance, healthcare, and public sector systems. If a generated recommendation is challenged, teams must demonstrate that the response was grounded in authorized and traceable sources rather than opaque model inference.

Auditability therefore transforms retrieval from a relevance mechanism into a governance mechanism that supports trust, compliance, and operational accountability.

Lessons I've Learned

In production, most RAG failures do not look like failures. The system retrieves context and generates a fluent answer, and users move on. The problem is that the evidence may be incomplete, overly generic, or subtly misaligned with the question.

Organizations often focus on model selection when evaluating RAG systems, but response quality is determined primarily by how precisely the pipeline selects, scopes, and prioritizes context. A well-designed retrieval pipeline enables grounded, explainable answers even with moderate models. A poorly tuned pipeline causes even advanced models to produce confident yet weak conclusions.

The practical starting point is not model tuning but context design. Evaluate whether chunking preserves causal coherence, whether retrieval depth captures sufficient evidence, and whether the system can explain why specific documents were selected. In enterprise deployments, the most reliable RAG systems are those that treat context assembly as a first-class architectural concern rather than an implementation detail.

CHAPTER 4

Governing Vector Databases in Production

A compliance analyst asks, "Which regulatory policy governs cross-border data transfers for EU customers?" The system retrieves a deprecated draft policy, an internal legal memo written for a different jurisdiction, and a general compliance overview. All three are semantically similar to the query. The generated answer is coherent, well-structured, and confident. It is also incorrect. No error is thrown. No alert is raised. From a system perspective, retrieval worked as designed. From a governance perspective, it failed.

This is how governance failures manifest in vector-based systems. They do not appear as access violations or system crashes. They appear as plausible, grounded responses built on inappropriate, outdated, or unauthorized context. Because the output looks reasonable, these failures are difficult to detect without deliberate controls.

This chapter explains how governance operates inside semantic retrieval systems. It focuses on how controls constrain retrieval behavior; how embeddings become governed assets; and how auditability, filtering, and policy enforcement ensure that context selection remains appropriate for enterprise and regulated environments.

What Governed Retrieval Looks Like

Consider the same regulatory query in a governed system. The vector database retrieves semantically relevant policy documents. Before context assembly, the system applies structured filters. Deprecated documents are excluded, jurisdiction metadata restricts results to EU-relevant policies, and privileged legal drafts are filtered based on access role. The final context bundle contains only current, authoritative EU regulatory guidance. The generated response is still fluent but now also compliant, traceable, and defensible.

The difference is not model intelligence. It is governance applied directly to the retrieval pipeline described in Chapters 2 and 3. Similarity scoring identifies candidates, but governance determines what evidence is allowed to reach the model.

Common Governance Failure Patterns

Governance failures in semantic systems rarely look like security incidents. They look like successful queries built on an inappropriate context. A sales representative searching for "customer renewal risk" may retrieve accounts they do not own if access filters are not applied after similarity ranking. A US-based application may surface EU customer records if residency constraints are not encoded in retrieval filters. A query about current policy may retrieve an outdated document that ranks highly due to semantic similarity, but that document should have been excluded based on its status metadata.

In each case, the system behaves normally. It retrieves relevant information according to similarity metrics. The failure occurs because relevance is not the same as appropriateness. Governance must therefore operate at the same decision points that shape retrieval behavior: filtering, ranking constraints, context assembly, and policy enforcement.

Governance Controls in Practice

Governance in vector databases is not a separate framework layered on top of retrieval. It operates inside the same pipeline described in Chapters 2 and 3, directly constraining filtering, ranking, and context assembly. Figure 4-1 illustrates how governance controls operate across the retrieval pipeline. The arrow represents the flow

of the retrieval process (filtering, ranking, and context assembly), while the controls apply across these stages to constrain how context is selected.

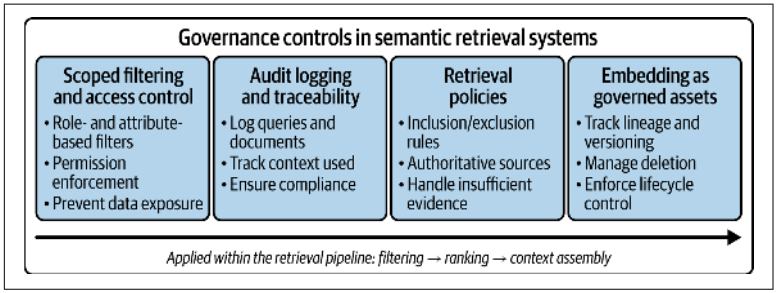


Figure 4-1. Governance in semantic retrieval systems operates through practical controls that constrain retrieval, ensuring that only appropriate, authorized, and traceable context is used to generate responses.

Scoped Filtering and Access Enforcement

As shown in Chapter 2, enterprise retrieval is typically hybrid. The system retrieves semantically similar candidates and then applies structured constraints. Governance depends heavily on this second step in the retrieval process.

For example, when a sales user queries "customer churn drivers," the vector index may surface highly similar records across the organization. A governed system applies metadata filters such as `account_owner == user_id` or role-based access attributes after candidate retrieval and before context assembly. Documents that fail these checks are excluded, even if their similarity score is 0.95.

This prevents semantic search from bypassing established access control boundaries. Access control must apply to embeddings and retrieval results with the same rigor as source data.

Audit Logging and Traceability

Governed retrieval requires traceability across the full retrieval-augmented generation (RAG) pipeline. Each query event must record the user or system that made the request, the filters and policies applied, the retrieved candidate documents, the final context bundle provided to the model, and the generated output.

This is a practical requirement: the model can only be as compliant as the evidence it is allowed to see. If a regulatory or legal question arises, teams must be able to reconstruct why specific documents were retrieved and how they influenced the response. Without retrieval-level auditability, organizations cannot explain or defend AI-assisted decisions.

Retrieval Policy as an Enforceable Constraint

Retrieval policies formalize governance rules at query time rather than relying on downstream review. These policies define which documents must be included, excluded, or prioritized for specific query classes.

For example, a policy for regulatory questions may require that at least one document marked `authoritative=true` and `status=current` be included in the final context bundle, while any document labeled `status=draft` or `status=deprecated` is excluded regardless of similarity score. If the candidate set does not contain an authoritative source, the system may widen the retrieval scope, adjust filters, or return an explicit "insufficient evidence" response instead of generating an answer from an incomplete or noncompliant context.

In this model, governance directly constrains context selection at query time, not just how data is stored.

Embeddings as Governed Assets

Traditional governance focuses on source data. Vector systems introduce a new class of governed artifacts: embeddings, which encode meaning derived from enterprise data and therefore inherit its sensitivity, lineage, and compliance obligations. Treating embeddings as secondary or disposable artifacts creates governance blind spots that become difficult to close at scale.

Lineage and Versioning of Embeddings

Each embedding should be linked to its source document, embedding model version, and creation timestamp. This allows teams to trace retrieval results back to the exact data and model state that produced them.

Versioning ensures that changes to embedding models or source content do not silently alter retrieval behavior. When a new embedding model or model version is introduced, regenerated embeddings should be tagged with that version identifier. This allows systems to track which vectors were produced by which model and makes it possible to compare retrieval behavior across versions before fully replacing the older embeddings. In practice, embedding versioning is often implemented using metadata fields in the vector index combined with standard data or machine learning versioning practices used in data pipelines.

In operational systems, lineage is implemented by linking each embedding to a stable source identifier, model version, and creation timestamp. For example, if a policy document is updated on March 15, the platform can query for all embeddings where `source_document_id = policy_123` and `embedding_created < 2026-03-15`, flag the embeddings as stale, and trigger regeneration. Without this level of traceability, teams cannot reliably distinguish between embeddings that reflect current guidance and those that still encode outdated content, even though retrieval behavior may appear normal on the surface.

Deletion Propagation and Data Retention

When source data is deleted for compliance or retention reasons, associated embeddings must also be removed or regenerated. Deleting the source record while leaving derived embeddings in the index can allow semantically similar content to be indirectly retrieved, creating hidden compliance risks.

Governed systems therefore treat deletion as a cascading operation that affects both structured data and vector representations.

Access Control for Derived Representations

Embeddings derived from sensitive data can still reveal semantic information about that data. Governance must ensure that access restrictions applied to source datasets are consistently enforced during semantic retrieval. This includes applying role-based and attribute-based access policies during candidate filtering and context assembly, not only during initial data ingestion. The same controls are also critical in multitenant systems, where embeddings from

one tenant must remain isolated from others to prevent unintended cross-tenant retrieval.

Governance Across the RAG Pipeline

Chapters 2 and 3 established that retrieval depth, chunking strategy, filtering, and reranking and context assembly determine what information reaches the model. Governance operates by constraining these same levers.

Chunking decisions influence what evidence can be retrieved and audited. Retrieval depth affects whether critical context is included or excluded. Filtering rules determine whether unauthorized or irrelevant documents are included in the context window. Reranking models can reorder retrieved candidates to prioritize the most relevant or authoritative sources before context assembly. Context assembly defines which documents ultimately shape the generated response.

Governance is not a separate layer. It is a control system applied directly to the mechanisms that shape retrieval behavior.

Model and Embedding Lifecycle Governance

Vector systems evolve over time as embedding models are updated and data changes. Regenerating embeddings with a new model can improve semantic alignment but may also shift ranking behavior across the entire corpus.

Documents that were previously top ranked may drop in rank due to changes in vector geometry, even though their content has not changed. Without versioning and side-by-side evaluation, these shifts can silently degrade retrieval quality and policy compliance.

Controlled rollout practices, such as embedding version tracking and staged reindexing, allow teams to evaluate behavioral impact before full deployment.

Governance in Edge and Distributed Environments

When vector databases operate at the edge or within local AI deployments, governance becomes more complex. Centralized policy enforcement, unified logging, and real-time access revocation may not always be available. In these environments, governance requires local enforcement of enterprise policies, synchronization of metadata and lineage, and controlled data movement between edge and central systems.

Treating edge deployments as governed zones rather than exceptions allows organizations to maintain consistent controls while respecting data sovereignty, residency, and operational constraints.

Lessons I've Learned

In production environments, governance failures rarely appear as obvious system errors. They appear as fluent and well-formed answers yet built on incomplete or inappropriate context. This makes them more dangerous than overt failures because they erode trust silently while appearing correct.

The most effective organizations do not treat governance as a separate compliance layer. They design it directly into retrieval mechanisms: filtering rules, lineage tracking, audit logging, and retrieval policies that constrain what evidence can be used. When governance is embedded into the same pipeline that shapes retrieval and generation, semantic systems remain explainable, auditable, and trustworthy as they scale across enterprise and regulated workloads.

CHAPTER 5

Evaluating Trade-Offs and Planning Adoption

A pilot retrieval-augmented generation (RAG) assistant is deployed internally. It answers questions correctly most of the time. Stakeholders are impressed. The demo works.

Six months later, the same system is slow under load, occasionally surfaces outdated documents, and requires manual embedding refresh after data updates. What began as a promising prototype is now treated as fragile infrastructure rather than a trusted platform capability.

This is the inflection point most organizations reach with vector databases. The question is no longer whether semantic retrieval works. It is whether it can be operated, governed, and funded as part of the core data architecture rather than maintained as an isolated initiative.

This chapter focuses on that transition. It examines real trade-offs around performance, cost, governance, and architecture, and outlines a pragmatic path from pilot to production.

What Successful Production Adoption Looks Like

In successful deployments, vector databases behave like platform services rather than experimental components.

A system that began answering a few hundred internal queries per day evolves into a production service supporting multiple applications. Retrieval is observable. Embeddings are versioned. Governance filters are consistently applied. When source documents change, embeddings are regenerated automatically.

Answer quality becomes predictable. Not perfect but stable. Current policies surface instead of deprecated drafts. Authorized documents are prioritized. When evidence is insufficient, the system signals uncertainty rather than generating confident but weak responses.

The system is judged by consistency under real workloads, not by occasional impressive demos.

What Failed Adoptions Look Like

Most failures occur during operationalization, not prototyping. A common pattern is embedding generation without lifecycle management. Teams embed documents once and never refresh them as data evolves. Retrieval gradually drifts even though the system appears unchanged.

Another failure pattern is treating retrieval as a black box. Latency increases and answer quality degrades, but no one monitors similarity scores, retrieval depth, or ranking stability.

Fragmented ownership is equally damaging. Application teams own prompts, data teams own ingestion, and platform teams own infrastructure, but no team owns retrieval quality end to end. When issues arise, remediation is slow and inconsistent.

These failures rarely stem from the vector technology itself. They stem from treating semantic retrieval as an isolated capability instead of a core data function.

Performance, Scale, and Cost Considerations

Vector workloads introduce different cost dynamics than transactional or analytical systems.

Storage scales with both corpus size and embedding dimensionality. A corpus of millions of documents, when embedded with higher-dimensional models, can double memory and storage requirements compared to smaller embeddings. At enterprise scale, dimensionality becomes an infrastructure decision, not just a modeling choice.

Latency trade-offs are similarly concrete. Retrieving the top five ranked results may deliver fast responses suitable for interactive applications. Expanding retrieval depth to the top 20 results can improve context coverage but increase compute cost and query latency, especially under high concurrency.

Approximate search configurations also introduce trade-offs. Higher recall settings typically increase latency and resource consumption. For chat assistants, even modest latency increases can affect user experience. For analytical workflows, the same trade-off may be acceptable.

The objective in production is consistent relevance within acceptable latency and cost boundaries, not theoretical maximum accuracy.

Accuracy Versus Efficiency in Practice

Similarity search returns sufficiently relevant context, not perfect matches.

In a support assistant, retrieving the fifth-most similar document instead of the third rarely changes the outcome if both contain the needed procedure. However, missing the most relevant document due to overly aggressive approximation can cause the system to return incomplete answers despite having the correct information available.

Success should therefore be measured by whether the retrieved context supports the downstream task, not whether the exact nearest neighbor was found.

Architectural Placement: Standalone or Integrated

As outlined in Chapter 1, architectural placement is not merely a deployment preference. It shapes long-term governance, operational sustainability, and the ability to manage AI workloads at scale.

Standalone vector systems enable rapid experimentation and fine-grained tuning. They are well suited for early stage or narrowly scoped semantic applications.

Integrated platforms align vector retrieval with existing governance, access control, and observability mechanisms. This reduces fragmentation and simplifies policy enforcement as semantic workloads expand.

A practical heuristic is straightforward. If semantic retrieval supports a single use case, standalone deployment is often sufficient. If multiple applications rely on shared semantic context and governance, integration into the core data platform becomes more sustainable.

Build, Buy, or Extend Decisions

Organizations typically choose between open source components, specialized vector systems, or extending existing platforms.

Open source tools are effective for prototyping but often lack enterprise-grade observability, governance, and lifecycle management. Specialized systems offer strong performance and scalability but introduce additional operational overhead. Extending existing platforms simplifies governance and integration but may limit deep tuning flexibility.

In practice, many organizations start with specialized or open source components to validate use cases and converge toward more integrated platforms as workloads mature.

When Not to Adopt Vector Databases

Vector databases are not universally necessary. Workloads dominated by exact lookups, structured queries, or deterministic reporting often gain limited benefit from semantic retrieval while incurring additional complexity.

Organizations without clear ownership for governance, observability, and lifecycle management may also struggle to sustain vector systems beyond pilots. In such cases, hybrid or traditional search approaches may provide more predictable outcomes until operational maturity improves.

Measuring Production Readiness

Production readiness requires explicit metrics beyond anecdotal answer quality.

At the retrieval layer, teams should monitor similarity score trends, ranking stability, and retrieval depth required to surface relevant context. Sudden shifts may indicate embedding drift or corpus misalignment.

At the pipeline level, context sufficiency is critical. Frequent incomplete answers may signal shallow retrieval or poor chunking. Fluent but weak responses may indicate overly broad context selection.

Operational signals such as latency under load, storage growth, and embedding regeneration time determine whether the system is sustainable at scale. User behavior also provides insight. Repeated query reformulation often points to retrieval misalignment rather than prompt issues.

A Pragmatic Path from Pilot to Production

A successful transition is incremental and observable.

During the pilot phase, teams focus on well-scoped use cases where relevance can be directly evaluated. Governance and observability may be lightweight but visible.

During operationalization, embedding versioning, governance filters, and retrieval monitoring are introduced alongside functional

improvements. Retrieval behavior is measured continuously rather than only during demonstrations.

During scaling, automated embedding refresh pipelines, formal governance controls, and service-level expectations for latency and relevance are established. At this stage, vector retrieval becomes a stable architectural component rather than an experimental feature.

Many organizations stall in what is often called the *pilot trap*: successful demonstrations that never evolve into operational systems. Early prototypes prove that semantic retrieval works, but they rarely include the governance, observability, and lifecycle management required for production environments. Moving beyond this stage requires deliberate operationalization. Embedding pipelines must become automated, retrieval behavior must be monitored, and governance controls must be applied consistently. Only when these practices are established does semantic retrieval transition from an interesting experiment to a reliable platform capability.

Figure 5-1 summarizes this progression from pilot experimentation to operationalized production systems, highlighting how governance, observability, and automation mature as deployments scale.

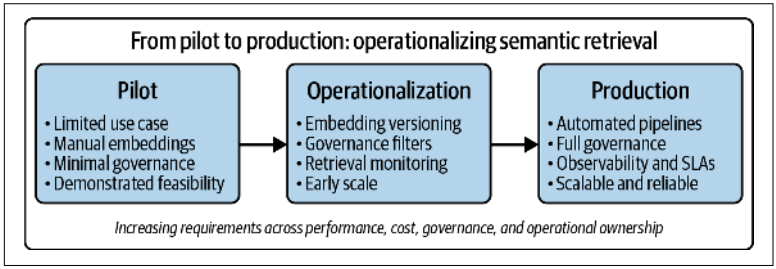


Figure 5-1. From pilot to production, vector retrieval evolves from isolated experimentation to a governed, observable, and scalable platform capability.

Lessons I've Learned

In practice, successful vector database initiatives rarely feel like dramatic technology bets. They resemble disciplined architectural evolution. Performance, cost, governance, and relevance must be evaluated together because they shape the same retrieval pipeline described throughout this report.

When they remain experimental side systems, quality drift, fragmentation, and silent trust erosion inevitably follow. The organizations that derive sustained value are those that treat semantic retrieval as foundational infrastructure rather than a clever add-on. When vector capabilities are aligned with platform governance, lifecycle management, and clear ownership, they become durable, observable, and trustworthy components of the enterprise data platform.

About the Author

Emma McGrattan is the Chief Technology Officer at Actian, where she leads the strategy behind some of the industry's most innovative data solutions. With over two decades of experience delivering mission-critical data technologies, she has helped organizations across industries design and modernize their data architectures, embed governance by design, and become AI-ready. A recognized thought leader and sought-after speaker, Emma is known for turning complex data challenges into clear, actionable blueprints. Her work bridges the gap between technology and business, enabling teams to move faster with trusted, scalable data.