When Agents Outgrow RAG: Building Production Retrieval Systems in the Lakehouse

Chang She
LanceDB
San Francisco, CA, USA
chang@lancedb.com

Prashanth Rao
LanceDB
Toronto, ON, Canada
prashanth@lancedb.com

Abstract

With AI agents, retrieval workloads shift from sequential, human-facing lookup to parallel, multimodal, stateful search. As organizations move from pilots to production workflows, the bottleneck shifts from prompting and harness engineering to data management: context sources multiply, embeddings and schemas evolve, and indexes and derived artifacts drift from source data, weakening the link between an agent's output and the context it used.

We argue that agentic retrieval needs a multimodal lakehouse, not standalone systems arranged around a vector index. Lance, an open columnar format for AI data, and LanceDB, the multimodal lakehouse built on it, provide one design point: typed Arrow-native tables for raw assets, metadata, embeddings, and derived features; versioned snapshots for reproducible evaluation, time travel, and rollback; and random access, scans, vector search, full-text search, and SQL predicates over the same data in object storage. In this setting, retrieval becomes *operational state for autonomous workflows*.

CCS Concepts

• **Information systems** → **Data management systems**; *Information retrieval*; • **Computing methodologies** → *Artificial intelligence*.

Keywords

AI agents, retrieval-augmented generation, lakehouse, multimodal data, vector search, data versioning, LanceDB

ACM Reference Format:
Chang She and Prashanth Rao. 2026. When Agents Outgrow RAG: Building Production Retrieval Systems in the Lakehouse. In *Proceedings of Supporting Our AI Overlords (SAO) Workshop at ACM CAIS 2026 (SAO '26).* ACM, New York, NY, USA, 3 pages.

1 Introduction

Retrieval-augmented generation (RAG) has usually been framed as a question of relevance and latency: given a user query, retrieve a small set of chunks quickly enough to improve a model response [7]. That framing is useful for interactive search, chatbots, and many early RAG applications. It is incomplete for production agents.

Agents do not simply ask one question and wait for an answer. They decompose tasks, spawn parallel searches, call tools, inspect intermediate results, generate new context, and revise plans [10, 13, 14]. Their retrieval traces may include documents, code, screenshots, tables, logs, embeddings, tool outputs, and prior agent state. The result is a different systems problem. The retrieval layer is no longer a passive lookup service behind a model. It becomes shared operational state for autonomous workflows.

Our claim is that the second-order effects of production agents should be addressed in the *data layer*. This complements recent work at adjacent layers of the agentic lakehouse stack: Liu et al. argue for agent-first data systems and query processing [8]; Tagliabue and Greco study safe programmable lakehouses for untrusted agents [12]; and Sheng et al. emphasize typed contracts, versioning, and transactional pipelines for correct-by-design lakehouse execution [11]. We focus on retrieval and multimodal context on the same substrate: how context can remain reproducible, governable, and efficient as agents and use cases scale.

We use Lance and LanceDB as a concrete design point [4, 5]. The relevant claim is not that every agent workload should use one database abstraction, but that storage-layer properties determine whether retrieval remains reproducible, governable, and efficient as deployments scale.

2 Second-Order Effects of Production Agents

The first-order retrieval question is whether an agent can find relevant context. The second-order question is what happens when the retrieval layer becomes shared infrastructure for many agents, teams, tools, and versions of context. Production agents do not operate against static datasets; they continuously create, consume, and transform the data they depend on.

Mutable context. Agents generate summaries, plans, labels, tool outputs, evaluations, embeddings, and derived context. These artifacts often become future inputs, so the context layer evolves as agents operate. The storage layer therefore needs snapshots, review, branching, and rollback without copying entire datasets or rebuilding indexes from scratch.

Coordination. Agentic retrieval is a concurrent workflow rather than a single-user interaction. Multiple workers may search in parallel, refine partial results, write intermediate state, and compose evidence before any response is produced. Without explicit semantics for read versions and write visibility, teams accumulate silent drift: duplicated context, stale embeddings, inconsistent schemas, and indexes that no longer represent the raw data.

Reproducibility. When an agent gives a bad answer or takes a bad action, debugging requires reconstructing what it saw: documents, embeddings, schemas, filters, index versions, and intermediate artifacts. Snapshot isolation and time travel make evaluations meaningful because they preserve the exact context state used during a run.

Multimodality. Many RAG systems normalize the world into text chunks and embeddings. Production agents also need PDFs, screenshots, tables, logs, code, video frames, audio clips, tool traces,

SAO '26, San Jose, CA, USA
2026.

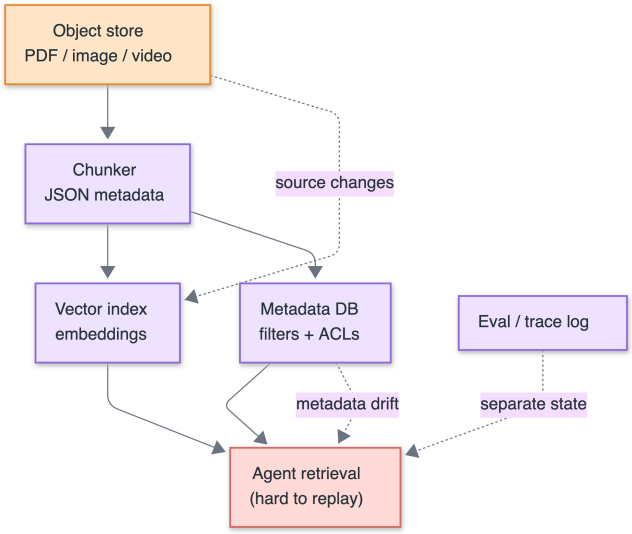

Figure 1: In a classic RAG stack, context is often split across object storage, JSON chunk metadata, a vector index, metadata services, and evaluation logs. Drift between these layers makes agent retrieval difficult to replay or govern.

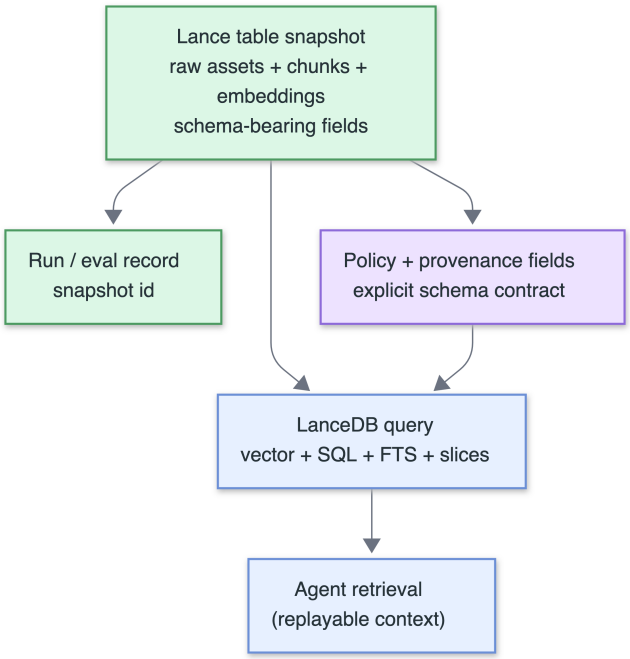

Figure 2: A context lakehouse treats retrieval as part of the storage layer, not only as a vector index. Raw assets, chunks, embeddings, policy fields, provenance, and derived features live in a typed, versioned table snapshot that retrieval and evaluation can reference.

and multimodal embeddings. Flattening these artifacts into text discards structure that agents need for grounded action.

Governance. As agent use cases multiply, teams need to know which agents can access which context, which embeddings were generated by which model, which artifacts are stale, and which datasets are approved for production. Split retrieval pipelines can support these invariants, but usually through additional orchestration across vector indexes, blob storage, metadata databases, and evaluation logs.

3 A Lakehouse View of Agentic Retrieval

Conventional retrieval pipelines often distribute context across vector indexes, object stores, metadata catalogs, evaluation logs, and orchestration code. This architecture can be effective, but its core invariants live between systems: which source version produced an embedding, which index served an evaluation run, when derived artifacts become visible, and how access policies propagate. A context lakehouse gives observability tools, policy engines, and agent orchestration a shared data contract: typed, versioned, multimodal context rather than implicit conventions between services [2, 11].

Typed columns address a different problem than vector similarity: they make the context interface explicit. In many document stores, fields such as page offsets, embedding model names, access labels, source versions, and derivation metadata appear as JSON metadata attached to chunks, as in Figure 1. That flexibility is useful, but at scale it becomes a source of drift because application code must remember which fields exist, what types they have, and which retrieval paths preserve them. In Lance, these become schema-bearing Arrow fields colocated with the raw assets and embeddings they describe, as in Figure 2 [1, 4]. An agent or orchestration layer can inspect this contract through a tool call such as `table.schema`, then use the exposed fields for filtering, replay, access control, and provenance checks.

Versioned snapshots give the context layer replay semantics. Agent runs and evaluations can reference exact table states, which makes it possible to reconstruct the context observed by an agent and to roll back derived artifacts that should not become shared state. Cheap schema and feature evolution let developers add new embedding columns, extracted features, and derived signals as agent workloads mature, without rewriting entire tables or disconnecting those features from the source data.

LanceDB builds on these properties to serve vector search, full-text search, SQL predicates, scans, and object access over the same underlying data [5]. This matters because useful agent pipelines rarely perform pure nearest-neighbor search. They combine semantic retrieval with structured constraints, provenance filters, recency windows, access controls, and direct inspection of raw artifacts. Random access and scans also reduce wasted I/O by fetching a page, frame, clip, or region instead of materializing a whole PDF, video, or audio file. For agentic workloads, the control-plane implication is that compute, policy, and orchestration can operate against explicit data contracts rather than reconstructing them from sidecar systems.

4 Examples Across Scales

These pressures are visible across scales, from large organizations to open source agent infrastructure. Uber [6] uses Lance for petabyte-scale multimodal AI workloads where model training and agentic search can involve thousands of concurrent readers over object storage. At that scale, the storage layer must preserve a single logical dataset while scaling access across distributed object-store layouts. The relevant systems lesson is that typed schemas, versioned manifests, and portable table semantics are not only governance features; they also become part of the scaling strategy for agentic retrieval.

At a different scale, Omnigraph [9] uses Lance as a typed, snapshot-backed graph context layer. Typed nodes and edges are stored as Lance datasets, while traversal, search, branching, and time travel operate over consistent graph snapshots. Despite their different settings, both examples converge on the same requirement: shared agent context needs multimodal data, explicit schemas, scalable object-store layouts, and versioned state.

5 Implications for Agent System Design

The second-order effects in Section 2 translate into concrete design principles. Because context becomes mutable infrastructure, it needs a governed lifecycle: runs and evaluations should reference exact snapshots, while durable agent outputs such as summaries, labels, extracted features, and generated assets should enter shared context with schemas and lineage [3]. Because agent workloads evolve quickly, data evolution should be cheap. Agent developers should be able to add embeddings, features, and derived signals over time without rebuilding the corpus or detaching them from source assets.

Because retrieval becomes a coordination problem, access paths should compose over one source of truth. Multimodal storage, vector search, full-text search, SQL predicates, scans, and raw object access are complementary, not separate systems of record. Because the unit of context is no longer just text, retrieval should minimize wasted fetches. Agents often need a specific page, frame, clip, or region; lazy scans and cached slices can provide that context while reducing object-store reads.

6 Conclusion

Production agents shift retrieval from a user-facing lookup problem to a data-systems problem. As organizations deploy more agentic workflows, the hard failures appear in consistency, reproducibility, governance, and lifecycle management of multimodal context. We argue that these are storage-layer concerns.

Lance and LanceDB illustrate one way to build this layer: typed columnar storage for AI and multimodal data, versioned snapshots, efficient object-store access, and multiple retrieval modes over the same lakehouse tables. In this architecture, retrieval is not merely an index behind an agent. It is the operational state on which autonomous workflows coordinate, evolve, and are audited.

References

[1] Apache Arrow Project. 2026. Arrow Columnar Format. https://arrow.apache.org/docs/format/Columnar.html. Accessed: 2026-04-29.

[2] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. 2021. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In *Proceedings of the Conference on Innovative Data Systems Research*. https://www.vldb.org/cidrdb/2021/lakehouse-a-new-generation-of-open-platforms-that-unify-data-warehousing-and-advanced-analytics.html

[3] Lance Format. 2026. lance-context: Manage Multimodal Agentic Context Lifecycle with Lance. https://github.com/lance-format/lance-context. Accessed: 2026-04-29.

[4] LanceDB. 2026. Lance Format. https://lance.org. Accessed: 2026-04-29.

[5] LanceDB. 2026. LanceDB: Multimodal Lakehouse for AI. https://docs.lancedb.com/. Accessed: 2026-04-29.

[6] LanceDB. 2026. Rethinking Table File Paths: Lance Multi-Base Layout. https://www.lancedb.com/blog/rethinking-table-file-paths-lance-multi-base-layout#the-uber-use-case-anchor. Accessed: 2026-04-29.

[7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[8] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. 2026. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. In *Proceedings of the Conference on Innovative Data Systems Research*. https://www.cidrdb.org/cidr2026/papers/p32-liu.pdf

[9] Omnigraph. 2026. Under the Hood. https://www.omnigraph.dev/docs#under-the-hood. Accessed: 2026-04-29.

[10] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[11] Weiming Sheng, Jinlang Wang, Manuel Barros, Aldrin Montana, Jacopo Tagliabue, and Luca Bigon. 2026. Building a Correct-by-Design Lakehouse: Data Contracts, Versioning, and Transactional Pipelines for Humans and Agents. arXiv:2602.02335. https://arxiv.org/abs/2602.02335

[12] Jacopo Tagliabue and Ciro Greco. 2025. Safe, Untrusted, "Proof-Carrying" AI Agents: Toward the Agentic Lakehouse. arXiv:2510.09567. https://arxiv.org/abs/2510.09567

[13] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. arXiv:2308.08155. https://arxiv.org/abs/2308.08155

[14] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*. https://iclr.cc/virtual/2023/oral/12647