From Similarity to Structure

A Decision Framework for Graph-Aware Retrieval in Enterprise AI Systems

Parth Majmundar
Sr. AI & Solution Architect
Version 1.0 — June 2026

Abstract

Retrieval-Augmented Generation (RAG) has become the default pattern for grounding large language models in enterprise knowledge, yet most production pipelines index that knowledge as flat text and retrieve it by vector similarity alone. This works well for lookup-style questions and fails quietly on questions whose answers live in the relationships between entities rather than in any single passage. Drawing on the recent literature on graph-augmented and agentic retrieval, and on patterns observed across enterprise solution-architecture engagements, this paper offers practitioners a workload-first decision framework: a taxonomy of five question shapes that flat retrieval cannot answer reliably, a five-level maturity ladder for graph-aware retrieval, a reference architecture, an evaluation approach that avoids common metric pitfalls, and a phased adoption roadmap. The central recommendation is deliberately conservative: audit your real query workload first, then adopt the lowest rung of the ladder that makes your critical questions answerable — structure is an investment, not a default.

1. Introduction

Since Lewis et al. formalized the retrieve-then-generate pattern [1], RAG has moved from research novelty to enterprise default. The recipe is familiar: chunk the corpus, embed the chunks, store the vectors, and at query time retrieve the top-K most similar passages to condition the model's answer. For a large class of questions — definitions, policy lookups, summarizing a known document — this is exactly the right architecture, and its operational simplicity is a feature.

But enterprise knowledge is rarely flat. Organizations run on structures: bills of materials, service dependency maps, organizational hierarchies, contract and entitlement chains, customer journeys, network topologies. When a stakeholder asks a question whose answer is a property of that structure — *which applications break if this database goes down? which vendors are we single-sourced on? who has not completed the new compliance training?* — the answer does not live in any retrievable passage. It must be computed over connections. A growing body of research documents this gap and proposes graph-augmented remedies [2, 3, 4, 7, 8, 9], including recent empirical work showing that several industrially important query classes remain unanswerable by similarity-based retrieval no matter how good the embeddings are [8].

This paper is written from the solution architect's chair rather than the researcher's. The question that matters in practice is not “is GraphRAG better than RAG?” — it is “which of *my* queries actually require

structure, and what is the cheapest, most maintainable architecture that answers them correctly?" The sections that follow are organized around that decision: Section 2 gives a taxonomy for auditing a query workload; Section 3 lays out a maturity ladder of architectures; Section 4 sketches a reference architecture; Sections 5–7 cover the decision checklist, evaluation, and a phased roadmap; Section 8 catalogs anti-patterns observed in the field.

2. Five Question Shapes That Break Flat Retrieval

The fastest way to find out whether you need graph-aware retrieval is to audit a representative sample of real user questions — not demo questions — and tag each one against the five shapes below. Each shape fails flat retrieval for a different structural reason, and the remedies differ accordingly.

2.1 Relationship chains (multi-hop dependency)

"Which customer-facing services depend on components hosted in the affected region?" The answer is assembled across several hops of a dependency structure, and no single chunk contains the full chain. Vector similarity retrieves passages that mention the query terms, not passages that complete the chain. Agentic, iterative retrieval — an LLM decomposing the question and issuing follow-up searches in a reason–act loop [5, 6] — can recover some multi-hop answers, and graph-structured passage exploration improves it further [7], but reliability drops as chains lengthen because nothing guarantees the iteration covers every branch.

2.2 Global aggregation and counting

"How many of our components have exactly one qualified supplier?" Counting requires exhaustive enumeration over the whole knowledge base. Top-K retrieval is, by construction, a sampling operation — and a sample cannot produce a trustworthy count or an exhaustive list. This failure is not fixable with better embeddings, larger K, or smarter chunking; recent benchmarking on industrial knowledge graphs finds aggregation queries remain failed across retrieval architectures until actual graph computation is introduced [8]. If your workload includes "how many," "list all," or "which ones are the only" questions whose answers must be complete to be useful, flat retrieval is the wrong tool.

2.3 Absence and complement

"Which teams have not yet remediated the vulnerability?" Retrieval finds what matches; absence leaves no trace in a similarity index. Answering requires closed-world set logic — the full population minus the matching subset — which presupposes that the system knows the full population. That is a database or graph property, not a property of a chunk store. These "inverse" questions are common in risk, compliance, and audit workloads, which is precisely where wrong-but-plausible answers are most expensive.

2.4 Temporal validity

"Who currently owns this control?" Text chunks do not expire. A 2023 runbook naming the old owner is retrieved alongside the 2025 reorg notice, and the model has no principled way to prefer one. Production knowledge needs validity windows — facts and relationships that are active, superseded, or scheduled — and retrieval that filters on them. Temporal modeling is an active research thread in graph-augmented RAG [8, 9], but even a simple active/expired flag on relationships eliminates a large share of stale-answer incidents in practice.

2.5 Whole-corpus synthesis

"What were the dominant failure themes across this year's incident reports?" The question is about the corpus as a whole, not any retrievable passage. Microsoft's Graph RAG work addressed exactly this query-focused global summarization gap by building an entity graph, detecting communities, and summarizing hierarchically [2]; lighter-weight graph-indexing variants followed [3, 4]. If your users ask thematic, portfolio-level, or trend questions, plan for a global-summary path alongside top-K retrieval.

The audit heuristic. Tag 50–100 real queries. In the author's experience most enterprise workloads are dominated by lookup questions that flat RAG handles well — but if a meaningful minority (roughly 15–20% or more) fall into the shapes above, and those questions carry operational or regulatory weight, the ceiling is real and no amount of prompt engineering will lift it.

3. A Maturity Ladder for Graph-Aware Retrieval

Graph-aware retrieval is not a binary choice but a ladder of increasing capability and cost. The discipline is to climb only as far as the workload audit demands.

Level	Architecture	What it unlocks	Marginal cost
L0	Flat vector RAG	Lookups, definitions, single-document QA	Baseline; cheapest to build and run
L1	Hybrid retrieval (metadata filters, recency, structured fallback to SQL)	Temporal filtering, scoped search, simple counts from systems of record	Low; mostly data plumbing
L2	Graph-indexed retrieval (entity/relation extraction, community summaries)	Whole-corpus synthesis, better multi-hop recall	Significant indexing-time LLM cost; extraction QA needed
L3	Agentic graph traversal (LLM planner + small vocabulary of typed traversal tools)	Reliable relationship chains; explainable hop-by-hop evidence	Graph store + tool layer; per-query latency from agent loop
L4	Graph computation (centrality, components, simulation, set operations as tools)	Aggregation, complement/absence, what-if and risk-propagation analysis	Graph analytics runtime; careful tool design and guardrails

Two design lessons from the recent literature are worth underlining for L3 and L4. First, give the model a *small, typed, composable* tool vocabulary — get-neighbors, filter-by-edge-type, find-paths, and the like — rather than a bespoke handler per anticipated question. Benchmarking on an industrial knowledge graph found that a planner equipped with a compact set of typed traversal primitives both outperformed hand-built query handlers and generalized to unseen question phrasings, and that adding genuine graph-computation operators (component analysis, centrality, removal simulation) was what finally made aggregation and what-if queries answerable [8]. The practical implication: the constraint on LLM-over-graph reasoning is usually the operator set you expose, not the model's intelligence. Second, keep deterministic paths deterministic — if a question maps cleanly to a SQL count or a fixed graph query, dispatch it there; reserve the agent loop for questions that need planning.

4. Reference Architecture

A production graph-aware retrieval system has six load-bearing components. None is exotic; the value is in how they divide responsibility.

Ontology before extraction. Decide what entities and relationship types matter, with an explicit schema. Where the knowledge originates in systems of record (ERP, CMDB, HR, contract systems), build typed edges directly from those systems — this sidesteps the extraction-noise problem that plagues LLM-built graphs. Reserve LLM entity extraction [2, 3] for genuinely unstructured corpora, and put human review on the extracted triples in regulated domains.

Dual store with provenance. Run a graph store and a vector index side by side, and keep links from every node and edge back to the source passages and records that justify it. Provenance is what turns a graph answer into an auditable answer.

Query router. Classify each incoming question by shape (Section 2) and dispatch: lookup questions to the vector path, structural questions to the graph path, thematic questions to the global-summary path, and mixed questions to a hybrid plan. The router is also the natural place to log the workload distribution that informs future investment.

Typed tool layer. Expose the graph to the LLM through a deliberately small set of typed, documented, individually testable operations — traversal primitives at L3, analytics operators at L4. Name them by what they compute, validate their arguments, and return structured results the model can cite.

Incremental freshness. Support atomic mutations — add or retire an entity, add or expire a relationship — with validity timestamps and a changelog, so the graph tracks reality without full re-indexing. Temporal correctness (Section 2.4) is bought here.

Governance. Node- and edge-level access control, audit logging of agent tool calls, and clear boundaries on which tools an agent may invoke autonomously. A traversal agent that can read the entire org graph is a data-governance event; treat it as one.

5. The Decision Checklist

Before committing to a level of the ladder, an architecture review should produce defensible answers to six questions. **Workload:** what fraction of real queries fall into the five structural shapes, and what do wrong answers to those queries cost? **Data shape:** is there a natural ontology and are there systems of record to source it from, or would the graph have to be extracted entirely from prose? **Freshness:** how quickly must the knowledge reflect reality, and can the sources emit change events? **Explainability:** do regulators, auditors, or executives need to see *why* an answer is what it is? Graph paths produce legible evidence; similarity scores do not. **Cost envelope:** L2 indexing carries a real LLM bill at corpus scale, while L3/L4 are cheap per query once the graph exists — the cost curves differ in shape, not just size. **Team:** who will own the ontology, the tool layer, and the evaluation harness after launch? The rule of thumb that falls out: **stop at the lowest level that makes your highest-stakes queries correct.** Many organizations genuinely need only L0–L1; the mistake is needing L4 and discovering it in production.

6. Evaluating What You Build

Evaluation is where graph-aware retrieval programs most often fool themselves, in three ways. First, **aggregate metrics hide categorical failure.** A system can post a respectable average while scoring zero on every aggregation and complement query — the failures concentrate in exactly the categories that

motivated the investment. Score per question shape, never only in aggregate. Second, **entity-overlap metrics mismeasure structural answers.** Token- or entity-level F1 penalizes comprehensive, correct structural answers that legitimately mention many entities, a measurement gap documented empirically in recent graph-retrieval benchmarking [8]; complement with task-specific correctness checks (is the count right? is the list complete?) and calibrated LLM-as-judge scoring with periodic human agreement checks. Third, **demo queries overfit.** Hold out unseen phrasings and entirely unseen question intents; a system built from typed primitives should generalize to them, and testing only rehearsed queries will hide it if it does not [8]. Finally, include negative cases — questions whose correct answer is "not present in the knowledge base" — because absence-blindness (Section 2.3) applies to evaluation, too.

7. A Phased Adoption Roadmap

Phase 1 — Audit and baseline (weeks 1–3). Collect and tag the real query workload against the Section 2 taxonomy. Stand up the evaluation harness first, including per-category ground truth, and measure the existing flat-RAG system against it honestly. Sketch the minimal ontology implied by the failing queries.

Phase 2 — Structure where it pays (weeks 4–8). Build the graph from systems of record for the audited domain; add metadata filtering and temporal validity (L1); add graph-indexed summarization (L2) only if thematic queries demand it. Introduce the query router and start logging workload distribution.

Phase 3 — Computation and hardening (weeks 9–12). Where the audit shows chains, counts, or what-if questions, add the typed traversal tool layer (L3) and the specific analytics operators those queries need (L4). Harden governance: tool-call audit logs, access controls, and the negative-case evaluation suite. Re-run the Phase 1 benchmark and report per-category deltas — that comparison, not a demo, is the business case for the next domain.

8. Anti-Patterns from the Field

Boiling the ontology ocean — spending two quarters modeling the enterprise instead of the dozen entity types the failing queries actually touch. **Unreviewed extraction in regulated domains** — LLM-extracted triples are noisy, and a graph that encodes hallucinated relationships fails more convincingly than no graph at all. **A handler per question** — bespoke query handlers feel fast to ship and rot immediately; small typed primitives generalize where handlers cannot [8]. **Timeless graphs** — relationships without validity windows reintroduce the stale-answer problem the graph was meant to fix. **Demo-driven evaluation** — if the test set was written by the people who built the system, it measures memory, not capability. **Structure as a substitute for data quality** — a knowledge graph faithfully propagates whatever is wrong in the source systems, at higher speed and with more confidence.

9. Conclusion

Flat vector RAG earned its position as the enterprise default, and for lookup-shaped workloads it should keep it. But the questions that organizations most need answered in risk, operations, and compliance are frequently questions about structure — chains, counts, complements, currency, and corpus-wide patterns — and those questions are not retrieval problems at all. The emerging research consensus [2, 7, 8, 9] and field experience point the same way: represent relational knowledge as an explicit, temporally aware graph; expose it to language models through small vocabularies of typed, auditable operations; route queries by shape; and evaluate per category against held-out questions. Climb the ladder only as far as your workload

demands — but know, before production finds out for you, which rung your hardest questions actually require.

References

[1] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2005.11401.

[2] D. Edge et al., “From Local to Global: A Graph RAG Approach to Query-Focused Summarization,” arXiv:2404.16130, 2024.

[3] Z. Guo et al., “LightRAG: Simple and Fast Retrieval-Augmented Generation,” arXiv:2410.05779, 2024.

[4] B. J. Gutiérrez et al., “HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models,” in *NeurIPS*, 2024. arXiv:2405.14831.

[5] S. Yao et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” in *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

[6] A. Asai et al., “Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection,” arXiv:2310.11511, 2023.

[7] H. Liu et al., “HopRAG: Multi-Hop Reasoning for Logic-Aware Retrieval-Augmented Generation,” arXiv:2502.12442, 2025.

[8] G. Chethan, “Beyond Vector Similarity: A Structural Analysis of Graph-Augmented Retrieval for Industrial Knowledge Graphs,” arXiv:2606.06003, 2026.

[9] B. Peng et al., “Graph Retrieval-Augmented Generation: A Survey,” arXiv:2408.08921, 2024.