Knowledge Graph Architectures for Enterprise Agents

Why flat RAG breaks at scale, and how to build enterprise knowledge graphs with node-level access control

The Agile Monkeys

March 2026

Why Flat RAG Breaks

The default enterprise RAG architecture is a single vector database containing embeddings of all organizational documents. An agent embeds a query, retrieves the most semantically similar chunks, and feeds them to an LLM as context. It works well enough at small scale. It breaks in at least four ways as organizations grow.

Linear degradation. Flat retrieval searches the entire corpus for every query. As the corpus grows, retrieval quality degrades — more noise, more irrelevant results, more token waste. LATTICE (Gupta et al., 2025) demonstrated that hierarchical retrieval achieves logarithmic search complexity through LLM-guided tree navigation, compared to the linear scaling of flat retrieval.

No multi-hop reasoning. Flat RAG retrieves isolated chunks. It cannot follow chains of relationships: this person works on this project, which relates to this department initiative, which connects to this strategic objective. As Glean's engineering team observed, "LLMs are effective at capturing broad semantic associations within their context window, but they struggle with multi-hop reasoning, enterprise-specific language, and understanding process or usage patterns."

No access control granularity. In a flat vector database, either you have access to the collection or you don't. There's no way to express "this person can see project-level details but only department-level summaries" — the kind of access differentiation that most organizations beyond a small team need.

No abstraction levels. An executive asking "what are our biggest risks this quarter?" and an engineer asking "what's the status of the authentication refactor?" need fundamentally different views of organizational knowledge. A flat corpus provides the same granularity to both, forcing the LLM to synthesize abstractions at query time — an unreliable process that produces inconsistent results.

GraphRAG (Microsoft Research, 2024) quantified the difference: on global sensemaking queries directed at an entire corpus ("What are the main themes?"), hierarchical retrieval won 72-83% of head-to-head comparisons on comprehensiveness and 62-82% on answer diversity (varying by dataset). The hierarchical approach used up to 97% fewer tokens for root-level summaries.

The Knowledge Graph Model

The answer to flat RAG's failures is not a strict hierarchy — it's a knowledge graph. A network of purpose-typed nodes connected by organizational, topical, and access edges.

Enterprise knowledge naturally forms a graph because organizations are graphs. A compliance policy connects to every department. A technology decision in engineering affects product, marketing, and sales. A security vulnerability cuts across every team. A project shares context with other projects in the same domain, regardless of departmental boundaries. None of these relationships fit a tree.

The organizational hierarchy — personal, team, department, organization — is one set of edges in this graph, emerging from "belongs-to" and "summarizes" relationships. It's the most natural navigation pattern because it mirrors how people think about organizations. But it coexists with cross-cutting edges ("applies-to", "relates-to", "affects"), purpose edges ("risk-assessment", "architecture-decision", "hiring"), and agent access edges ("consumes", "produces") that operate independently of the org chart.

Every production knowledge system we examined confirms this. Neo4j's labeled property graphs, GraphRAG's community detection, Glean's multi-level knowledge graphs, and Enterprise Knowledge's consulting experience all converge on the same insight: hierarchy is an emergent property of certain edge types in a graph, not a structural constraint on the data model (Enterprise Knowledge, 2025). GraphRAG's Leiden algorithm demonstrates this empirically — when you run community detection on an enterprise knowledge graph, hierarchical clusters emerge naturally at different resolutions, without being designed in.

The formal foundation is Heterogeneous Information Networks (HINs) — typed graphs with multiple node types (users, agents, documents, knowledge buckets, organizational units) and multiple edge types (authored, consumes, produces, has-access-to, summarizes). Meta-paths — sequences of node and edge types — define meaningful traversal patterns for both navigation and access control. The meta-path Agent→AssignedTo→Project→OwnedBy→Team→InDepartment→Department resolves an agent's departmental access through organizational relationships.

The key governance insight: every node in the graph has its own access configuration. A risk-assessment node at the department level has different access requirements than a hiring-pipeline node at the same level. Access is not about which level you're on — it's about which nodes you can reach through authorized relationship paths.

Five Implementation Patterns

Five distinct patterns implement aspects of the knowledge graph model. Each has different strengths, and the right choice depends on organizational size, access control requirements, and existing infrastructure.

Pattern 1: Tree-Structured Summarization

Examples: RAPTOR (Stanford, ICLR 2024), LATTICE (2025)

RAPTOR builds a tree from documents upward through recursive clustering and summarization. Text chunks are embedded, clustered using soft clustering (allowing membership in multiple groups), and each cluster is summarized by an LLM. The summaries are re-embedded, re-clustered, and re-summarized, building successive tree layers. At query time, retrieval can happen from any level — detailed leaf nodes or abstract root summaries.

LATTICE extends this with LLM-guided tree navigation, achieving zero-shot hierarchical retrieval with logarithmic search complexity. The LLM reasons about which branches to explore before descending into detail, avoiding exhaustive search.

Enterprise mapping: Leaf nodes are individual documents and conversations. Mid-level nodes are project and team summaries. Higher nodes are department overviews and organizational themes. RAPTOR demonstrated a 20% absolute accuracy improvement over prior state-of-the-art on the QuALITY benchmark for complex multi-step reasoning tasks, using GPT-4 (Sarthi et al., ICLR 2024).

Best for: Organizations with large document corpora that need both fine-grained retrieval and thematic overview capabilities.

Pattern 2: Community-Based Hierarchy

Example: GraphRAG (Microsoft Research, 2024)

GraphRAG extracts entities and relationships into a knowledge graph, then applies the Leiden community detection algorithm to identify hierarchically nested communities of densely connected nodes. Each community at each level is pre-summarized. At query time, partial responses from relevant community summaries are aggregated into a final answer.

Enterprise mapping: Communities naturally form around teams, projects, and departments. Cross-cutting communities emerge around shared topics (compliance, technology choices, customer segments) that span organizational boundaries — something a strict hierarchy misses.

Best for: Organizations where cross-departmental knowledge connections matter as much as within-department depth. Particularly strong for "What are the themes?" type queries.

Pattern 3: Multi-Layer Semantic Architecture

Examples: Salesforce Agentic Enterprise Architecture, Enterprise Knowledge's KI Architecture

Salesforce's reference architecture organizes capabilities across 11 layers, with a Semantic Layer centered on an Enterprise Knowledge Graph (EKG) providing shared vocabulary and relationship models across the organization. The EKG translates natural language queries into precise data queries, harmonizes results across organizational silos, and returns contextual answers. As described by Salesforce, the architecture includes granular, just-in-time access verification for agents based on their specific task.

Enterprise Knowledge's three-layer model provides a cleaner abstraction: a Semantic Layer (foundation — metadata, taxonomies, ontologies, knowledge graphs), a Knowledge Capture Layer (intelligence — structured repositories of tacit and explicit expertise), and a Retrieval Layer (access — RAG frameworks for LLM-mediated knowledge access). The semantic layer creates "a more intuitive and connected representation of organizational data entities without having to physically move the data."

Best for: Large enterprises with complex, siloed data landscapes that need a unifying abstraction without physically consolidating all data.

Pattern 4: Domain-Specialized Agents

Example: Agentic RAG (Xenoss)

Instead of a single retrieval system serving all queries, this pattern deploys specialized agents for distinct knowledge domains — one for HR knowledge, one for engineering docs, one for financial data — each maintaining its own knowledge scope and access controls. A coordinator agent routes queries to the appropriate specialist.

Enterprise mapping: One agent per department or function, each with access limited to its domain. Scale horizontally by adding agents without architectural overhauls. PII removal and data preprocessing happen at ingestion time per domain.

Best for: Organizations that want to start simple and scale incrementally. Each domain agent can be built and deployed independently.

Pattern 5: Multi-Level Knowledge Graphs

Example: Glean

Glean maintains parallel graphs at different scopes: a personal graph (individual work patterns and activities), an enterprise graph (organizational entity relationships), and a system-of-context graph (processes and workflows). The personal graph clusters atomic actions ("created document X," "attended meeting Y") into tasks and projects, hierarchically linked to organizational objectives like OKRs.

Fine-grained access controls ensure employees see only data shared with them in source systems. Edge properties include timestamps and access control metadata, enabling governance-aware reasoning.

Best for: Organizations that want to combine individual productivity (personal knowledge management) with organizational intelligence, with permission-aware reasoning built into the graph structure.

Comparison

Pattern	Scalability	Access Control	Cross-Cutting Knowledge	Setup Complexity
Tree summarization	O(log n)	By tree level	Soft clustering helps	Medium
Community hierarchy	Good	By community	Excellent (community detection)	High
Multi-layer semantic	Enterprise-grade	By layer + ontology	Via semantic layer	Very high
Domain agents	Horizontal scaling	Per agent/domain	Requires coordination	Low per domain
Multi-level graphs	Good	Per edge/node	Via graph traversal	High

Access Control Across the Hierarchy

ReBAC Over RBAC

Traditional Role-Based Access Control assigns users to roles and roles to resources. In a hierarchical knowledge base, this fails because access is contextual and transitive. A manager should see their team's knowledge because they manage the team, not because someone assigned them a "team-knowledge-reader" role. When they move to a different team, their access should change automatically — through the relationship, not through manual role reassignment.

Relationship-Based Access Control (ReBAC) models permissions as paths through a graph of relationships. A user → manages → team → owns → project → contains → document chain grants read access to the document. Changing the "manages" relationship automatically updates all downstream permissions.

Permit.io demonstrated this pattern concretely with a custom filter component that intercepts retrieved results and validates each against the user's relationship chain (Permit.io, 2025). In a healthcare context: Doctor → Medical Center → Visit → Diagnosis — the access follows the care relationship, not a static role assignment.

For hierarchical knowledge bases, ReBAC is the natural fit because the knowledge hierarchy is a relationship graph. Permissions flow through organizational relationships rather than being statically assigned at each level.

Where to Enforce Access Control in the Retrieval Pipeline

Where in the RAG pipeline should access control be enforced? Four options exist, and the choice matters for both security and performance:

At indexing time (embed permissions in vectors or maintain permission-aware partitions): Most secure in principle — unauthorized documents never enter the retrieval pipeline for a given user. The trade-off is flexibility: permission changes may require re-indexing, which at enterprise scale means lag between access changes and enforcement. Permission-aware partitioning (separate indexes per tenant or access class) is a common middle ground.

Pre-retrieval filtering (apply permission predicates before the vector search runs): This is the preferred pattern when the database engine supports it. By filtering the candidate set first, the search engine compares fewer records — improving both security and performance. Most production vector databases (Weaviate, Pinecone, Qdrant) support metadata filters that can be applied before or during the vector search; the database's query planner decides the optimal order. The key point: the pipeline should express permissions as pre-retrieval constraints wherever possible, and let the engine optimize. Expressing permissions only as post-retrieval filters forces the engine to rank irrelevant candidates it could have eliminated earlier.

Post-retrieval filtering, pre-generation (filter after vector search, before the LLM sees anything): The fallback when pre-retrieval isn't feasible — for example, when permissions depend on content that can only be evaluated post-retrieval, or when the relationship graph needed to make the decision is too expensive to flatten into a pre-retrieval predicate. This still provides strong enforcement: the LLM never sees unauthorized

content. The downside is wasted retrieval work and, if the filtered set is too small, degraded answer quality.

At generation time (instruct the LLM to redact): Unreliable. LLMs may inadvertently reveal filtered information through summaries, implications, or refusal patterns. Never use this as a primary control.

The general rule: push permission evaluation as early in the pipeline as it can correctly be expressed. Pre-retrieval if possible, post-retrieval as a fallback, never at generation time as the only gate.

ABAC for Purpose-Segmented Knowledge

Within each hierarchy level, not all knowledge serves the same purpose. A department-level knowledge base contains risk assessments, hiring decisions, architecture records, and operational metrics — each with different sensitivity levels and different audiences. An engineering manager needs access to engineering risk assessments but not to HR's hiring pipeline, even though both exist at the department level.

Purpose-focused segmentation addresses this by adding topic-typed attributes to knowledge containers. In vector database terms, this maps to Weaviate's class-per-purpose model: separate classes (RiskAssessments, HiringDecisions, ArchitectureRecords) with tenant isolation per organizational unit. In streaming architectures, it maps to Kafka topic partitioning: raw events flow into purpose-specific derived topics with independent access control lists.

The authorization model is Attribute-Based Access Control (ABAC), as defined in NIST SP 800-162, layered within the ReBAC graph. ABAC evaluates access based on attributes of the subject (role, department, clearance), the resource (purpose, sensitivity, organizational level), and the environment (time, context, risk level). A policy rule might read: "A principal with role=engineering-lead AND department=platform can read resources with purpose=risk-assessment AND level=department AND sensitivity<=confidential."

This is not a replacement for ReBAC — it's a complement. ReBAC determines which relationships grant access. ABAC determines which attributes of the knowledge itself are accessible. Together they express rules that neither can alone: "access this knowledge if you manage the team that owns it (ReBAC) AND the knowledge's purpose matches your current task scope (ABAC)."

A practical governance concern: bucket proliferation. Without controls, departments will create increasingly granular purpose categories until the segmentation becomes unmanageable — a recurring tag-sprawl problem familiar from enterprise knowledge management systems. Guardrails

include: a maximum number of purpose types per organizational level, mandatory review for new purpose types, and periodic consolidation of underused categories.

Provenance and Traceability

When knowledge nodes contain derived artifacts — summaries, aggregations, extracted themes, community reports — they must maintain structured references back to the original source content. This is not optional. LLMs make mistakes, and if a summary cannot be traced back to the documents it was derived from, errors become impossible to detect, correct, or audit.

Every derived node in the graph should carry provenance metadata as a first-class property:

- Source references: the specific source nodes (or raw documents) the derived artifact was built from, with stable identifiers that survive re-indexing
- Generation timestamp: when the derivation was run
- Generator identity: which process or model produced the artifact, with a version identifier (so a regression in a summarization model can be linked to affected artifacts)
- Derivation logic version: which pipeline logic was applied (schema version, prompt version, or code commit)

Provenance makes the knowledge base self-correcting. When a source document is updated or found to be inaccurate, every derived artifact that depends on it can be identified through reverse lookups and regenerated. When a user exercises GDPR erasure rights, the tombstone can propagate through provenance links to identify every downstream artifact that needs to be rebuilt without the erased content.

Provenance is also the technical substrate for trust: when an agent cites a claim, it should be able to point to the specific knowledge node that supports it, and that node should be able to point to its source facts. A chain of citation from agent output back to raw organizational data is what separates a useful agent from an unaccountable one.

In practice, provenance metadata lives in node attributes and in edge types like "derived-from" or "summarizes" — which makes it naturally queryable through the same graph traversal mechanisms used for retrieval.

Collaborative Memory and Controlled Sharing

Rezazadeh et al.'s Collaborative Memory framework (2025) directly addresses multi-user memory sharing with dynamic access control. The system uses bipartite graphs linking users, agents, and resources to encode permissions. Memory is divided into private memory (visible only to

the originator) and shared memory (selectively shared fragments that benefit others when permissions allow).

The results validate the approach: the fully collaborative scenario with 5 users and 6 domain agents reduced resource usage by up to 61% at 50% query overlap, while maintaining strict access compliance even as permissions were dynamically granted and revoked.

This demonstrates that controlled sharing across a knowledge hierarchy isn't just a governance exercise — it's an efficiency gain. When team members can share relevant knowledge through permission-aware mechanisms, the system avoids redundant processing, redundant storage, and redundant LLM calls.

Practical Challenges and Mitigations

Summary Staleness

When underlying data changes, higher-level summaries become stale. A project status summary generated last week may contradict decisions made today. This is a frequently cited concern about hierarchical KBs.

Mitigations: Event-driven re-summarization triggered by significant changes in source data (using Change Data Capture from the data pipeline layer). Staleness indicators on summaries showing when they were last regenerated. Tiered update frequencies: personal-level summaries update hourly, project-level daily, department-level weekly.

Information Loss at Higher Levels

Each summarization step loses detail. Critical nuances — a single dissenting voice in a project discussion, a subtle risk signal — can be dropped when aggregating team knowledge into department summaries.

Mitigations: Maintain bidirectional links from summaries to source data, allowing drill-down. Use extractive + abstractive summarization: the abstract summary captures themes, but key quotes and data points are preserved as extractive highlights. Flag high-confidence anomalies (risks, blockers, disagreements) for explicit inclusion in higher-level summaries.

Cross-Cutting Knowledge

Not all knowledge fits organizational hierarchy. A compliance policy affects all departments. A technology decision in engineering impacts product, marketing, and sales. Security vulnerabilities cut across every

team.

Mitigations: GraphRAG's community detection handles this naturally — cross-cutting communities emerge from the relationship graph regardless of organizational boundaries. Alternatively, maintain tagged cross-cutting categories that are indexed at all relevant hierarchy levels.

Permission Boundary Conflicts

A document may be relevant to multiple organizational levels with different access controls. Project notes might be authored by a junior engineer but relevant to a VP's strategic query. Who determines access?

Mitigations: Use source system permissions as ground truth. The document's access in its originating system (Google Drive, Confluence, Slack channel membership) determines who can see it in the knowledge hierarchy. When in doubt, apply the most restrictive access. Maintain provenance metadata so access decisions are auditable.

GDPR Erasure in Hierarchical Knowledge

A user exercises their right to erasure. Their data exists at the personal level (their own notes and contributions), but it may have been summarized into team, department, and organizational knowledge. Deleting the source data doesn't delete the summaries that incorporated it.

Mitigations: Three complementary approaches. Tombstone events in event streams mark source data as deleted, preventing future inclusion in any regenerated summaries. Summary regeneration re-runs summarization pipelines excluding tombstoned data, replacing stale summaries. Cryptographic erasure encrypts personal data with per-user keys; destroying the key makes the data irrecoverable even if physical storage persists.

Cost

The NStarX analysis estimates federated RAG architectures at 2-3x baseline cost due to privacy-preserving computation requirements. For hierarchical KBs specifically, multi-level summarization (LLM calls at each level), multiple indexes, and permission evaluation at query time introduce additional compute costs beyond the base retrieval layer.

Mitigations: Lazy summarization — only generate higher-level summaries when they're queried, not proactively. Tiered caching — cache frequently accessed summaries. Use smaller, cheaper models for routine re-summarization, reserving expensive models for novel synthesis. Monitor cost-per-query at each hierarchy level to identify optimization

opportunities.

When NOT to Use Hierarchical KBs

Hierarchical knowledge architectures add real complexity and cost. They are not always the right choice.

Under ~1,000 employees with uniform access levels: If everyone can see everything (or access is binary: internal vs. external), the hierarchy provides minimal access control benefit. Flat RAG with good chunking and metadata filtering is simpler and cheaper.

Homogeneous knowledge domains: If the organization's knowledge is a single type (e.g., a law firm's case database, a research lab's paper collection), hierarchy adds overhead without meaningful benefit. Flat search with good filtering works fine.

Early-stage agent adoption: If you're at Level 1-2 on the maturity model (copilots and single-purpose agents), the overhead of building a hierarchical KB before you have agents that can leverage it is premature optimization. Start flat, identify the pain points, then add hierarchy where it matters.

When fast iteration matters more than organization: If the knowledge base is changing rapidly (early product development, active incident response) and organizational structure is fluid, the maintenance burden of keeping hierarchical summaries current may outweigh the retrieval benefits.

The honest recommendation: start with flat RAG. Monitor where it fails — queries that return irrelevant results, access control requests that can't be expressed, executives asking for summaries that agents can't produce. Those failure patterns tell you exactly which parts of the hierarchy to build first.

References

- Sarthi, P. et al. "RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval." ICLR 2024. arXiv:2401.18059
- Edge, D. et al. "From Local to Global: A Graph RAG Approach to Query-Focused Summarization." Microsoft Research, 2024. arXiv:2404.16130
- Gupta, N. et al. "LATTICE: LLM-guided Hierarchical Retrieval." 2025. arXiv:2510.13217

- Rezazadeh et al. "Collaborative Memory: Multi-User Memory Sharing in LLM Agents with Dynamic Access Control." 2025. arXiv:2505.18279
- Enterprise Knowledge. "Graph-Based Security & Entitlements." 2026. enterprise-knowledge.com
- Salesforce Architecture. "Agentic Enterprise IT Architecture." 2025. architect.salesforce.com
- Permit.io. "Building AI Applications with Enterprise-Grade Security Using FGA and RAG." 2025. permit.io
- Glean. "Knowledge Graphs as Agentic Engines." 2025. glean.com
- NStarX. "The Next Frontier of RAG: How Enterprise Knowledge Systems Will Evolve 2026-2030." 2025. nstarxinc.com
- Xenoss. "Enterprise AI Knowledge Base Architecture with RAG and Agentic AI." 2025. xenoss.io
- Enterprise Knowledge. "Knowledge Intelligence Architecture." 2025. enterprise-knowledge.com
- Enterprise Knowledge. "From Taxonomies to Knowledge Graphs." 2025. enterprise-knowledge.com
- NIST. "Guide to Attribute Based Access Control (ABAC)." SP 800-162, 2014 (updated 2019). csrc.nist.gov
- Weaviate. "Multi-Tenancy in Weaviate." 2025. weaviate.io
- Neo4j. "Knowledge Graphs for AI." 2025. neo4j.com
- Shi, C. et al. "A Survey of Heterogeneous Information Network Analysis." IEEE TKDE, 2017. arXiv:1611.10230