lettria

The State of GraphRAG

Editorial Note

Most GraphRAG tutorials stop where the interesting problems begin: after the demo, before the production system.

This paper is written by practitioners who have spent years building graph-based retrieval systems for clients in regulated industries: legal, pharmaceutical, financial, and engineering. It is not a survey. It is not a pitch. It is a summary of what we learned, including the things we could not find written down anywhere else.

Enterprise teams are now committing real infrastructure budgets to GraphRAG. We wrote this to help them spend those budgets well.

With external perspectives from [KOL list].

Contributors: Assia Khan, Marie Drouhin, Julien Plu, Romain Albrand and Charles Borderie.

"Traditional RAG excels at summarizing and rephrasing information, but has its limitations when it comes to conducting structured, reliable, and explainable reasoning."

Patrick Meyer
AI Senior Architect, Sopra Steria

SUMMARY

Inside this report

A practitioner's guide to building GraphRAG in production, chapter by chapter.

What Is GraphRAG?

Why traditional RAG struggles with cross-document reasoning and relationship discovery.

Origins

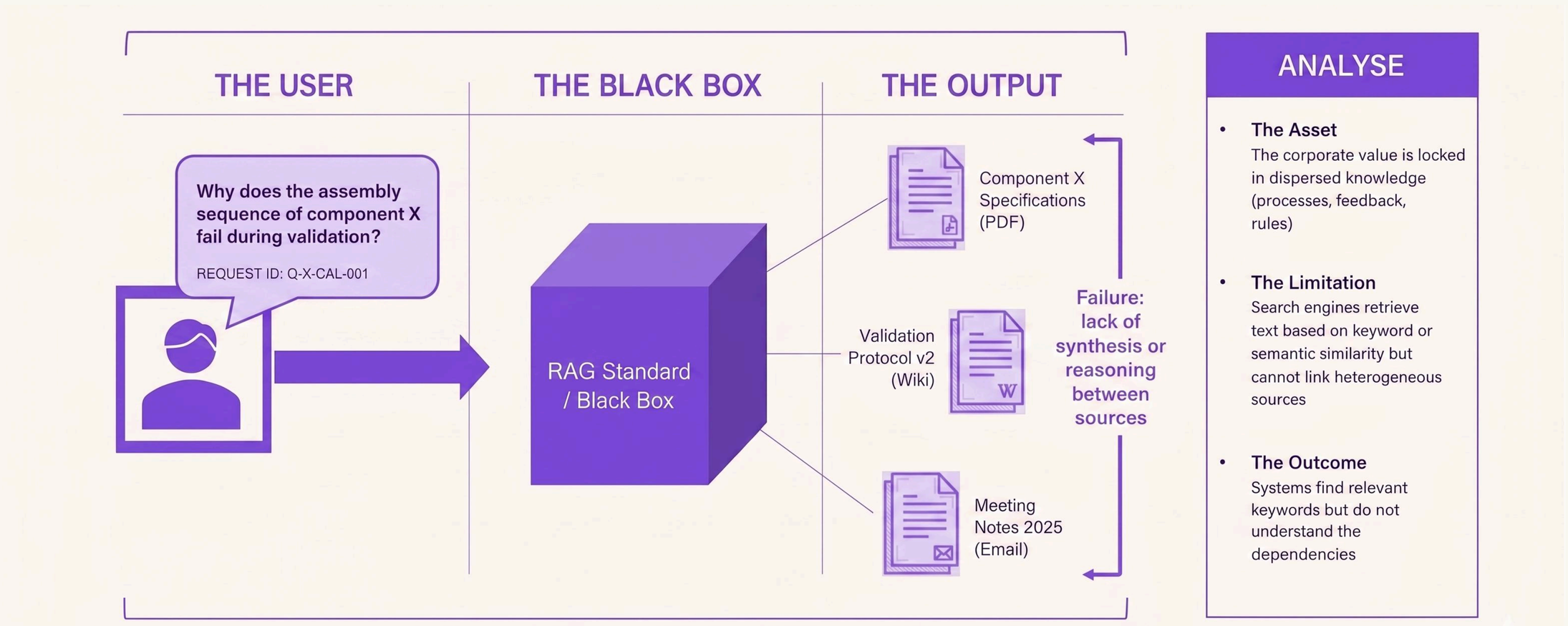
Retrieval-Augmented Generation was formalized by Lewis et al. at Facebook AI Research in 2020. The core idea is simple: instead of compressing all knowledge into model weights, retrieve the relevant fragment at query time and let the model reason over it. It worked well, but it hit a ceiling quickly.

Standard RAG retrieves chunks. Contiguous passages of text, ranked by vector similarity to a query. This works when the answer lives in a single passage. It breaks when the answer requires connecting information across multiple documents, following a chain of relationships, or identifying patterns that only emerge at the corpus level.

The GraphRAG Shift

GraphRAG was developed as a direct response to that limitation. The underlying intuition: knowledge is not a collection of isolated passages. It is a network of entities and their relationships. By representing that network as a graph and retrieving subgraphs rather than chunks, a system can answer questions that flat retrieval cannot.

The term "GraphRAG" was popularized by Microsoft Research's 2024 paper and open-source implementation. But the foundational techniques have roots stretching back over a decade to the Semantic Web and Linked Data communities. The technology is newer than the hype suggests, but the ideas behind it are not.



Traditional RAG systems suffer from a lack of conceptual structure

Where It Is Already Working

Legal and compliance

Standard RAG retrieves clauses. GraphRAG connects provisions across contracts, exposing legal dependencies and potential risk. Large law firms use knowledge graphs for due diligence and contract review.

Pharmaceutical and life sciences

Drug development knowledge spans compounds, targets, clinical outcomes, and literature. GraphRAG links these sources to answer complex research questions that no single document can address.

Financial services

Fraud detection and risk analysis are relationship-driven problems. GraphRAG traces ownership structures, sanctions exposure, and entity connections that traditional retrieval cannot capture.

Industrial and energy

Equipment failures often require connecting maintenance records, incidents, and operational data across years. GraphRAG helps identify root cause patterns across assets and document repositories.

GraphRAG has found its strongest early traction in industries with three shared properties: large document volumes, dense relational structure, and a high cost of error.

In each of these cases, the common thread is the same: the answer is not in any single document. It is in the relationships between documents. That is the problem GraphRAG was built to solve.

Understanding what GraphRAG is and where it works is a starting point. The next question is more practical: is it the right tool for your specific use case?

When to Use GraphRAG?

TL;DR

The decision comes down to one simple question: what kind of query are you answering? GraphRAG earns its complexity from synthesis and multi-hop queries. On simple lookups, it is overhead with no upside.

The Query Is Central to Your Decision

The first question to ask before building a GraphRAG system is not "how do we build it?" It is "What are we actually trying to answer?" There are two fundamentally different query types, and they require different architectures.

Lookup queries

They have a single, locatable answer. The answer exists in one place in the corpus. Vector RAG finds it efficiently, at lower cost and with less infrastructure.

→ **Use Vector RAG**

High-level queries

They require global synthesis. They do not ask "where is X?" They ask "what is the overall pattern of X across the entire dataset?" The answer lives in relationships between documents.

→ **Use GraphRAG**

The Decision Table

QUERY TYPE	EXAMPLE	RIGHT TOOL
Lookup	*“What is the approved dosage of Drug X for adults?”*	**Vector RAG**
Comparison	*“How do the termination clauses in Contract A and B differ?”*	**Vector RAG or light hybrid**
Multi-hop	*“Which of our suppliers are indirectly exposed to sanctioned entities?”*	**GraphRAG**
Synthesis	*“What adverse event patterns recur across our Phase II oncology trials?”*	**GraphRAG**
Global	*“What topics are underrepresented in our FDA submissions?”*	**GraphRAG with community detection**
Analytical	*“What are the characteristics of cities with fewer than 200 people?”*	**GraphRAG**
Temporal	*“Who was the first president of the United States after World War II?”*	**GraphRAG with Temporal KG**

Source: Notion — When to use GraphRAG? (Marie Drouhin, Lettria)

"Knowing what GraphRAG can do is useful. Knowing when to use it is more important."

Marie Drouhin
Data Analyst at Lettria

lettria

High-Level Questions in GraphRAG

By 2026, the industry consensus has shifted: simple fact retrieval is now a solved problem. If a user needs to find a specific clause in a contract or a dosage in a medical manual, traditional Vector RAG is sufficient. The real frontier of RAG, and the primary reason organizations are investing in graph architectures, is the possibility of answering high-level queries.

High-level queries are questions that require global synthesis. They don't ask "Where is X?" but rather "What is the overall trend of X across the entire dataset?" Addressing these requires moving beyond simple text-chunk matching to understand the full set of documents.

The Synthesis Wall: Why Traditional RAG Fails

Traditional RAG hits a "synthesis wall" when faced with global questions. To answer a summary-style query, a vector-based system must either:

Flood the context: Retrieve numerous chunks, creating a "noisy context" that leads to LLM distraction and high token costs.

"By making dependencies explicit, GraphRAG enables the system to navigate between concepts, cross-reference multiple layers of information, and reconstruct complete business contexts."

Patrick Meyer
AI Senior Architect, Sopra Steria

The Architecture of Global Answers

Community Partitioning

To answer global questions effectively, we lean on community partitioning. The process involves partitioning the graph's nodes into clusters based on their relational density. Once clustered, the system generates community summaries for each partition. This transforms thousands of raw data points into a manageable set of high-level abstractions.

> Our Perspective: For these communities to be meaningful, a unified graph is essential. This is where the ontology becomes a strategic asset rather than an academic exercise. A strict, well-defined ontology ensures that entities from 1,000 different files are resolved into the same nodes, allowing communities to form across the entirety of the corpus.

The Quality Feedback Loop

Construction quality is the foundation of synthesis. If the initial graph extraction (the "Leaf Level") is riddled with hallucinated relations or missed entities, those errors are compounded as they move up the hierarchy. A 10% error rate in entity resolution can lead to a 50% distortion in a community summary.

To Sum Up

- Community partitioning allows the system to summarize vast amounts of data without hitting context limits.
- Hierarchical structures (RAPTOR-style) provide the flexibility to answer questions at multiple levels of granularity.
- Query scope analysis is mandatory to prevent summaries from becoming "noise" for local queries.

- Global synthesis is the "Killer App" of GraphRAG, moving beyond simple search.

- Decomposition turns vague, high-level prompts into actionable, multi-step answers.

NLP vs Linked Data

The graph construction problem nobody talks about

TL;DR

Two communities have built graphs for decades without talking to each other. That gap shapes every architectural decision in a GraphRAG system, and explains why construction quality is the bottleneck most teams never see coming.

Two Schools, Two Visions

Two paradigms dominate graph-based information retrieval: text-extraction graphs and formal Knowledge Graphs (KGs).

Text-extraction graphs, used in frameworks like GraphRAG and LightRAG, are built directly from raw text. Nodes represent entity mentions as extracted by language models, without mapping to any canonical reference. Systems rely on vector similarity or LLM prompting to link different mentions of the same concept.

Formal Knowledge Graphs resolve text mentions into unique, disambiguated identifiers (URIs) governed by a schema or ontology. Nodes represent canonical concepts, not raw strings, enabling deterministic querying (SPARQL), logical reasoning, and reliable data integration.

The key difference is entity resolution. String-based graphs use text embeddings to approximate semantic similarity; formal KGs use relational machine learning, Knowledge Graph Embeddings and Graph Neural Networks, to model structured, typed relationships.

This creates an asymmetry: formal structure can govern raw-text graphs, but string-graph heuristics cannot support the logical requirements of canonical knowledge graphs.

Two traditions have built graphs for decades, and structure only flows one way.

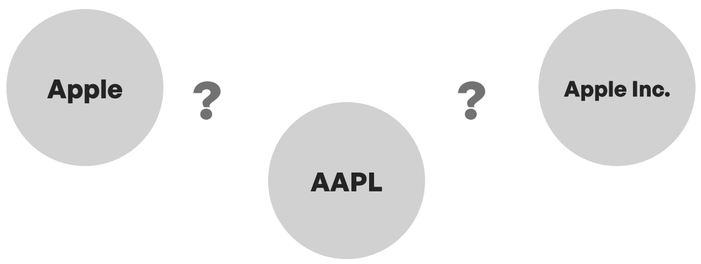
Text-Extraction Graphs

Built directly from raw text

GraphRAG · LightRAG

- Nodes are raw entity mentions
- Linked by vector similarity or LLM prompting
- No mapping to any canonical reference

SAME ENTITY · THREE UNLINKED NODES



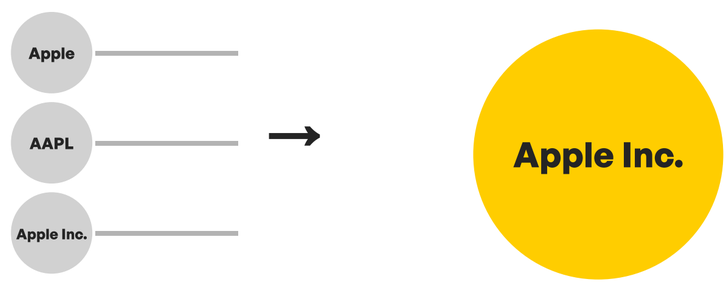
Formal Knowledge Graphs

Resolved against a schema / ontology

Wikidata · Enterprise KGs

- Nodes are canonical concepts (URIs)
- Deterministic querying with SPARQL
- Supports logical reasoning & integration

ALIASES RESOLVED TO ONE NODE



Formal structure can govern raw-text graphs.

The reverse does not hold: string-graph methods can't meet the logical requirements of a canonical knowledge graph.

Where We Sit

Our approach bridges these two paradigms through a unified pipeline. We use the scalable, flexible extraction capabilities of NLP as an ingestion mechanism, then immediately resolve those raw extractions into a structured, typed ontology.

For enterprise use cases, this sequential workflow is highly effective. It lets organizations process unstructured documents at scale while establishing the referential integrity, strict traceability, and query reliability that only a formal, canonical schema can guarantee over time.

The Construction Quality Crisis

The consequences of getting this wrong show up in production. GraphRAG systems frequently underperform naive vector RAG in real-world applications, and the cause is rarely retrieval logic or generation quality. It is poor graph construction.

Graph-based retrieval increases recall, but it also introduces noise from extraction errors. When the graph contains hallucinated relations or inconsistently typed nodes, retrieval returns contaminated subgraphs. The generation model produces confident, coherent, wrong answers. No error is thrown; the system fails quietly.

A 2% relation hallucination rate sounds negligible. In a graph built from one million documents, it means tens of thousands of false edges, structurally coherent, but factually wrong.

The Ontology as Guardrail

This is where the ontology becomes an operational necessity rather than an academic exercise.

The ontology acts as a governance layer: it enforces business rules, validates concept consistency, and rejects structurally absurd relations, turning a raw graph into a robust knowledge graph. Without it, the “Subject–Predicate–Object” triple structure becomes anarchic.

It also handles entity deduplication. “K8s” and “Kubernetes” must resolve to the same node; “Apple Inc.”, “Apple”, and “AAPL” must resolve to the same node. Without reconciliation logic grounded in the ontology, these become disconnected nodes, and every multi-hop query that should traverse them fails silently.

The ontology is to GraphRAG what the SQL schema is to a relational database: the guarantee of system integrity.

Patrick Meyer
AI Senior Architect, Sopra Steria

Putting It to the Test: The Perseus Benchmark

This is what led us to build Perseus, a text-to-graph extraction benchmark comparing 29+ models across 19 ontologies and 14,882 manually verified triples, measuring the construction-quality dimensions that determine production outcomes.

The single most important metric is Output Reliability: the share of outputs that are perfectly valid and schema-compliant. A model that produces unparseable outputs can’t be used at scale, regardless of its F1 score.

The Perseus Benchmark

MODEL	OUTPUT RELIABILITY	AVG
Lettria Perseus fine-tuned · Mistral Small 3.2	99.95%	
GPT-5.2 no-thinking	93.35%	
Claude 4.5 Sonnet no-thinking	91.96%	
Gemini 2.5 Pro minimal thinking	85.9%	
GPT-5 minimal thinking	66.63%	
Gemini 2.5 Flash no-thinking	56.85%	

Source: perseus.lettria.com/leaderboard · Fine-tuned models are Lettria’s purpose-built text-to-graph extractors.

At their default (no-thinking) settings, frontier models trade reliability for speed. GPT-5.2 and Claude 4.5 Sonnet land in the low 90s; GPT-5 and Gemini 2.5 Flash fall to between 57% and 67%. Lettria's fine-tuned Perseus extractors reach 99.95% reliability at a fraction of the cost ($0.01) and comparable latency, with no reasoning required.

The Perseus Benchmark

MODEL	OUTPUT RELIABILITY	AVG LATENCY	AVG COST	AVG F1
Lettria Perseus fine-tuned · Mistral Small 3.2	**99.95%**	7.9 s	$0.01	**0.89**
GPT-5.2 no-thinking	**93.35%**	3.0 s	$0.02	**0.66**
Claude 4.5 Sonnet no-thinking	**91.96%**	5.3 s	$0.03	**0.72**
Gemini 2.5 Pro minimal thinking	**85.9%**	3.6 s	$0.01	**0.68**
GPT-5 minimal thinking	**66.63%**	5.0 s	$0.01	**0.49**
Gemini 2.5 Flash no-thinking	**56.85%**	1.4 s	$0.01	**0.42**

Source: perseus.lettria.com/leaderboard · Fine-tuned models are Lettria's purpose-built text-to-graph extractors.

Three findings stand out

Fine-tuned models win without thinking. Every Lettria SFT extractor clears 98% reliability (Mistral Small 3.2 at 99.95%, Qwen 3 32B at 99.9%) at roughly $0.01 per run and single-digit-second latency.

Frontier reliability is rented, not owned. The same models reach 99%+ only with heavy reasoning, and pay for it in latency and cost; at their default setting they drop into the 57% to 93% range.

Smaller general-purpose models are unusable. Nano-class models score in the single digits, failing structurally on almost every attempt. Graph construction rewards fine-tuning, not raw scale.

The operational takeaway is direct: use a fine-tuned extraction model feeding the graph store, and reserve frontier models for generation, where reasoning capacity is genuinely needed.

From Retrieval to Reasoning

Building a flexible Graph RAG pipeline: deep reasoning across query types, under strict latency constraints.

In a modern RAG (Retrieval-Augmented Generation) pipeline, user queries span a wide spectrum of complexity. As highlighted by comprehensive datasets like GraphRAG-Bench, a robust system must seamlessly handle diverse tasks: fact retrieval, contextual summarization, complex reasoning, and even creative generation.

At Lettria, balancing this need for deep reasoning across varied prompt types with strict latency constraints led us to rethink the traditional chunk-based RAG architecture. The five core concepts below define our Graph RAG Chat Pipeline.

The GraphRAG Pipeline

Ingestion builds the graph. Chat reasons over it.

PHASE 1 · INGESTION

1

Document Parsing

Normalize raw PDFs, HTML, and exports into clean, processable text.

→

2

Contextual Summaries

Precompute per-document summaries so the chat layer fetches only what matters.

→

3

Graph Construction

Resolve raw mentions into canonical nodes and relations defined by the ontology.

→

4

Community Detection

Build hierarchical graph communities, pre-indexed into a vector database.

PHASE 2 · CHAT

1

Query Scanning

Route the query and tune the hybrid search strategy on the fly.

→

2

Hybrid Search

Flexible bi-encoder lexical + dense retrieval via Qdrant, tuned per query.

→

3

Graph & Community Retrieval

Pull linked concepts, neighborhoods, and community summaries for macro context.

→

4

Context Filtering

Rerankers and temporal-validity checks strip noise before generation.

→

5

Noise-Free Generation

The LLM reasons over a clean, grounded, strictly relevant context.

1. Front-loading Computation at Ingestion

To maintain a low-latency chat experience, the heavy lifting is shifted to the ingestion phase. By precomputing critical elements (contextual summaries per document and hierarchical graph communities), we structure the data upfront, so the chat pipeline can quickly and accurately fetch exactly what it needs to answer a given prompt.

2. The Power of Graph-Based Retrieval

A clever person is often defined by their ability to connect the dots. A sophisticated RAG system is no different. Learnings from GraphRAG-Bench show that as datasets grow, traditional chunk-based retrieval fails on tasks requiring cross-document synthesis. Our graph-based approach solves this by:

- **Linking Concepts**: capturing the intricate relationships between entities across multiple sources. A complete, richly connected graph is essential for the complex reasoning and contextual summarization the benchmark evaluates.
- **Macro-Level Visibility**: providing a broader view of the ingested dataset to answer high-level summary questions.
- **Community Retrieval**: inspired by Microsoft's GraphRAG, we pre-index graph communities into a vector database, expanding context with high-level information while staying compatible with a low-latency chat pipeline.

3. Fast & Flexible Hybrid Search

Our retrieval mechanism is powered by Qdrant (built in Rust) to strictly preserve latency. It uses a highly flexible bi-encoder that supports both lexical and dense retrieval, letting the system adapt its fetching strategy to the specific needs of each question.

4. Dynamic Query Scanning

Not all questions require the same effort. Clever query scanning acts as the routing engine that gradually guides the pipeline. Because a slightly exotic creative-generation prompt needs a very different search strategy than a strict fact-retrieval question, the scanner analyzes the input to dynamically decide what data to fetch and how to tune the hybrid vector-search parameters on the fly.

5. Delivering a “Noise-Free” Generation Context

A common pitfall is relying entirely on a sophisticated LLM to filter messy context. To harness an LLM's true reasoning power, and to avoid latency spikes that can easily exceed 10 seconds, you must put yourself in the shoes of the generation model. We ensure a clean context by:

- **Filtering**: using rerankers and temporal-validity checks to strip out useless items.
- **Simplification**: providing appropriate textualization for graph elements so the LLM easily understands the structured data.
- **Clarity**: sending a strictly noise-free, highly relevant context, so the LLM can focus entirely on reasoning rather than parsing irrelevant data.

Three under-rated techniques for higher accuracy

Brad Bebee
AWS Director · Expert perspective

Ontology bootstrapping
An LLM proposes entity types, relationships, and cardinality constraints from a representative corpus, validated against expert feedback, a schema filter that cuts hallucinated edges by constraining what the graph can even represent.

Agentic GraphRAG
A planner–executor loop replaces single-shot retrieval: decompose the query into subgraph traversals, check each result for sufficiency, then backtrack and refine, raising multi-hop recall 20–40% over baseline RAG.

Incremental graph maintenance
Change-data-capture propagates document diffs as localized node/edge upserts with provenance, avoiding accuracy drops between full rebuilds and enabling recency-based confidence scoring.

Conclusion

What we have learned building GraphRAG in production.

GraphRAG is neither a silver bullet nor hype. It is a specific tool for a specific class of problem: questions whose answers live in the relationships between documents rather than inside any single one. When your queries are lookups, vector RAG stays the right, cheaper choice. When they demand synthesis, multi-hop reasoning, or global structure, the graph earns its complexity.

One theme recurs throughout this paper: the hard part of GraphRAG is not retrieval or generation, it is construction. A graph built from noisy, unresolved extractions fails quietly, returning answers that are confident, coherent, and wrong. The ontology is what turns a fragile string-graph into a robust knowledge graph, and purpose-built extraction models are what make construction reliable at scale.

For teams committing real budgets, the path is pragmatic: start from the query, treat the ontology as a first-class asset, invest in construction quality before retrieval cleverness, and reserve frontier models for the reasoning step where they genuinely add value.

Five things to remember

1. **Start from the query, not the architecture.**
2. **The ontology is infrastructure, not paperwork.**
3. **Construction quality is the real bottleneck.**
4. **Fine-tuned extraction beats frontier APIs on graphs.**
5. **Synthesis, not lookup, is where GraphRAG wins.**

Build the graph well, and the reasoning takes care of itself.

EXPERT PERSPECTIVE

Where GraphRAG Goes From Here

Anthony Alcaraz
Agentic Engineering Lead · AWS

You can tell a research area has matured when it stops producing prototypes and starts producing benchmarks. ICLR 2026 accepted both LinearRAG and the GraphRAG Benchmark; AAAI 2026 accepted LogicRAG. Enterprise context isn't one problem but three, and the research is diversifying a distinct portfolio of answers for each.

1 · The agent's own context

Working memory (per-task) and knowledge memory (durable) get conflated. Build a full graph for throwaway scaffolding and you overpay; serve durable knowledge from raw retrieval and the agent re-derives it every session. Price structure per task: lightweight, query-time structures (LinearRAG, LogicRAG) for volatile work; full construction where volume amortizes the build and provenance must survive an audit.

2 · Institutional knowledge

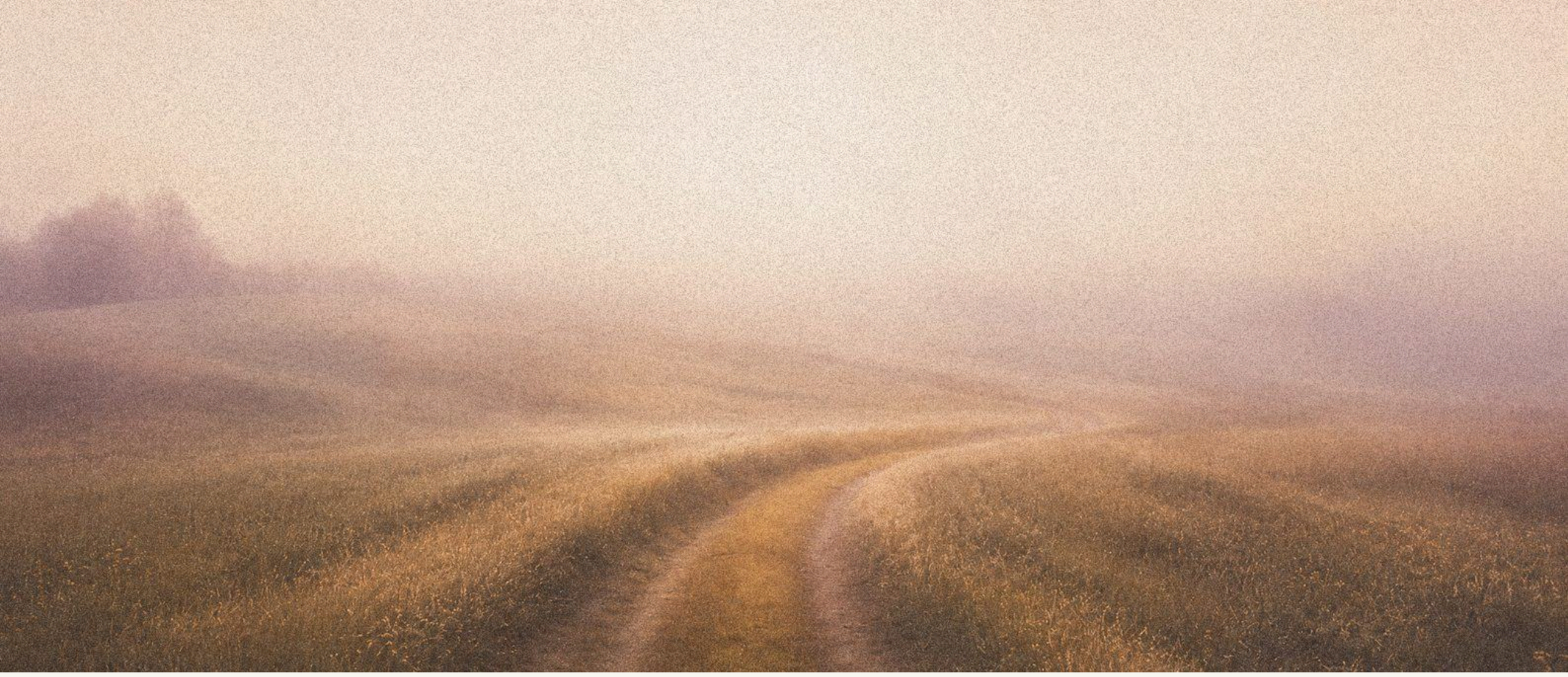
It is fragmented across applications and across modalities, decks, diagrams, recordings, tickets. Text-only graphs miss what lives in the figures. Multimodal KG construction makes figures and tables first-class nodes; a minimal ontology defines only where your meaning diverges from the model's defaults, using agent confusion as the signal for what to formalize next.

3 · Systems of record

Where context stops being read and starts grounding action. Data is bimodal in trust, access is governed, and the graph itself is an attack surface, poisoning launders provenance through edges. The answer is a control plane around the graph: provenance-gated writes, agent-native authorization, control/data-plane separation, and temporal validity.

The model is the same one your competitor rents. The context layer is the only part that is yours, and graph structure is its governable, auditable, ownable form.

Anthony develops the full architecture, from knowledge-graph construction to agentic retrieval to the governance control plane, in Agentic Graph RAG (Alcaraz & Julien, O'Reilly Media).



GET IN TOUCH

Ready to build with GraphRAG?

Lettria builds production-grade GraphRAG systems for enterprise. Let’s talk.

Charles Borderie
CEO, Lettria
charles@lettria.com
lettria.com

ABOUT LETTRIA

Lettria is the missing context layer for enterprise AI. Our platform turns structured and unstructured data into ontology-powered knowledge graphs ; so your AI agents reason instead of guess. The result: 30% more accurate answers, fully traceable and audit-ready. We serve the most demanding industries : healthcare, finance, industrial, from prototype to production. Whether you build with our developer tools or our embedded experts, Lettria makes enterprise AI grounded and trusted.

CREDITS

Written by the Lettria team
Expert perspectives: Anthony Alcaraz & Brad Bebee (AWS), Patrick Meyer (Sopra Steria), Yoan Chabot & Carmelle Meli Songuon (Orange)
Engineering & Research Team: Marie Drouhin & Romain Abrand, Julien Plu
Marketing & Design: Assia Khan

lettria

The Graph Context Layer beneath the AI stack