

Forum:[Hack Forums](#)

Thread: [My journey building Graph-RAG in production](#)

Oh_Well - September 30, 2025, 7:21 am

My journey building Graph-RAG in production

Hi guys, so I've been building production Graph-RAG systems for serious orgs (EU/US defense orgs/contractors etc).

I was preparing a thread like this but honestly it was nightmare thinking the sequence of all that information.

I've been putting together this knowledge and work for almost a year now so I hope this thread becomes a go-to resource that people return to when exploring Graph-RAG and one they recommend to others looking to prioritize their workflow.

I think now it's the right time to share it and consider it as my 2nd entry to this amazing contest.

[Quantum Thread Contest #1 - Win BTC & HackForums Awards! - Recruitment](#)

Why this was needed in 2025

I walked into environments where mission-critical intel lived in PDFs, wikis, tickets, and legacy DBs.

Vector similarity found "close" text but couldn't explain why claims were true or trace dependencies across projects and time.

Graph-RAG turned silos into a living map you can reason over.

Graph-RAG gave us: Fewer hallucinations: every non-obvious claim cites a doc.

Multi-hop reasoning: follow dependencies, ownership, incidents.

Explainability: show the exact path and timestamp.

Complex queries]: e.g., "What's downstream risk if Component X fails next quarter?"

Graph-RAG in one line

Fuse a knowledge graph (entities + typed edges) with RAG so your LLM retrieves paths with provenance, not just similar paragraphs.

Week 1: Ship a walking skeleton

Spin up a graph store: Neo4j (or Memgraph/Neptune).

Add a vector store: pgvector (or Pinecone/Weaviate/Milvus).

Ingest 200–500 core docs, extract entities/relations, attach on edges:

Code source_id, url, timestamp, confidence
Expose a single endpoint returning answer + citations + graph paths: /ask

tip: Keep a raw-docs bucket (S3/GCS) - - store hashes and versions for reproducible answers.

Work on a minimal retrieval loop that actually works

Stage A: KNN on text (top-K chunks).

Stage B: Expand via graph 1–3 hops around entities from A.

Stage C: Re-rank on (edge-type weights, recency, entity overlap).

Graph expansion: Start at the concept that matches the user's topic.

Walk 1–2 "RELATED_TO" links to collect nearby entities.

Keep the first ~200 entities from this expansion.

For each entity, attach documents that explicitly describe it (e.g., "ABOUT" links).

Discard any documents that weren't already picked by your vector search.

Prefer newer documents and entities that are more connected in the graph.

Return the top ~50 entity–document pairs as candidates for the LLM context.

Make the LLM play by your rules

Use a system prompt that forces citations like [DOC-123] and allows "Insufficient evidence".

Assemble context as subgraphs with human-readable paths:

Code PATH: Component X -> DEPENDS_ON -> System Y -> INCIDENT -> Ticket #4412

SNIPPET: "...replaced pump Y on 2024-03-14..."

CITE: [DOC-4412] https://...

This single change killed most hallucinations for me.

Chunking that won't break everything

Keep semantic units intact (headers/sections/tables). My default: ~400–1200 tokens per chunk.

Use one embedding provider, and store embedding version + dimension on each chunk so you can migrate later.

A schema that actually survives on a production env

Start simple: Nodes:

Code :Document

:Entity {labels} Edges:

Code AUTHORED, REFERENCES, PART_OF, DEPENDS_ON, OCCURRED_IN, PRECEDES,

RISK_TO, LOCATED_IN Indices:

Code Entity.name, Document.source_id Directionality: be explicit or add inverses where reasoning needs it

Index setup & author-linking (no code):

Create an index so entities can be found quickly by name.

Create an index so documents can be found quickly by a stable source ID.

Ensure a Person node exists for the author name.

Ensure a Document node exists for the given source ID.

Update the document's title, URL, and published timestamp.

Connect the Person to the Document with an AUTHORED relationship that records when it was authored.

Two game changing moments Temporal filters matter. "Since 2023" should demote 2019 edges

even if they're central.

Communities matter. Dense neighborhoods (Louvain method - u can look it up) give richer, safer context than isolated nodes.

Validate before you insert (avoid a dirty graph)

Or better - how to avoid garbage in - garbage out lol

LLMs extract noisy triples. I run a validator: "Is this relationship supported by the snippet? Return 0..1." Insert only if ≥ 0.7 and always connect via evidence.

Validated insert flow - what happens (no code): Ensure a source Entity exists (by canonicalized name).

Ensure a target Entity exists (by canonicalized name).

Ensure the referenced Document exists (by source_id).

Ensure a relationship of the requested type connects source -> target.

Increase the relationship's confidence score by half of the extraction confidence.

Update the relationship's "last seen" timestamp to now.

Connect the relationship to the evidencing Document so provenance is explicit.

Entity resolution

Cache canonical names - reconcile nightly with string sim + embedding sim + org IDs. Keep a manual override list. Basically treat this as a service, not a one-off script.

Performance & cost cheats Batch inserts - parallel ETL.

Cache hot subgraphs per topic/team.

Cap traversal depth dynamically based on result quality.

Push filters into Cypher (time ranges, labels) instead of post-filtering.

Keep re-rankers small, run after narrowing to ~50–200 candidates.

The prompt & path trick

Render edge types with tiny natural-language glossaries (e.g.,

Code `DEPENDS_ON` = "requires to function").

LLMs reason better when edge semantics are explicit!!!!

Evaluation that won trust Retrieval: Recall@K, Path Accuracy, Coverage

Generation: Factual Accuracy, Citation Accuracy, Groundedness

System: P50/P95 latency, orphan-node rate, average degree, token efficiency

Build a 30–50 question eval set from real stakeholders. Re-run after every change.

When you dont want heavy pre-processing

Look at lightweight/"lazy" Graph-RAG approaches that reduce up-front indexing (useful for pilots). You can also auto-tune prompts per domain if your corpora vary a lot.

OrchestrationRoll-your-own: small Python service + Neo4j/pgvector drivers.

Frameworks: LangChain's graph retriever, LangGraph for control-flow. LlamaIndex KG-RAG examples if you want quick demos.

Advice: Start simple and add agents/tools later.

A tiny 30-day plan that worked repeatedly

Week 1: ingest ~300 docs, naive extraction, barebones endpoint: /ask

Week 2: hybrid retrieval (KNN > graph hops), re-ranker, strict citations.

Week 3: eval harness, logging/traces, hallucination checks.

Week 4: caching + indices, RBAC on nodes/edges, target P50 < 2.5s.

What you don't wantOver-modeling the schema. Keep it small ---- grow with real questions!!!!

Stuffing prompts with 100% text & 0% paths. Paths are the whole point.

Ignoring timestamps. Stale edges look authoritative to LLMs.

Treating entity resolution as a batch job, not a service.

What this enabled in practice

Analysts got answers with traceable "whys".

Engineers saw downstream blast radius.

Support agents got troubleshooting chains instead of isolated KB hits.

They trusted it because every non-obvious claim had a doc ID and a path.

Copy-paste snippets

System prompt for cited, grounded answers:

Code SYSTEM: You are a precise analyst. Only use the provided CONTEXT.

Every non-obvious claim MUST include [SOURCE_ID]. If evidence is missing, say so.

USER QUESTION:

{user_query}

CONTEXT:

{assembled_context}

Simple rerank scoring (fast + effective):

Code score = 0.5semantic_sim(query, text)
+ 0.2entity_overlap(query_entities, node_entities)
+ 0.2edge_type_weight(edge_types_in_path)
+ 0.1recency(doc.published)

Nightly graph quality checks (no code):Count how many Entities have no relationships (the orphan rate). Aim to drive this down or justify why some nodes should be isolated.

Look at the degree distribution (e.g., the 90th percentile of connections per node) to catch abnormal hubs or overly sparse areas.

List any relationship types currently in the graph that are not part of your approved schema and tally how many times each occurs (schema drift report).

Sources & starter kitsMicrosoft GraphRAG (GitHub): <https://github.com/microsoft/graphrag>

Neo4j Docs: <https://neo4j.com/docs/>

Neo4j Developer Guides (AI/Graph-RAG): <https://neo4j.com/developer/>
LangChain Docs: <https://python.langchain.com/docs/>
LangGraph (control flow for LLM apps): <https://python.langchain.com/docs/langgraph/>
LlamaIndex Docs: <https://docs.llamaindex.ai/>
Memgraph Docs: <https://memgraph.com/docs/>
Pinecone Docs: <https://docs.pinecone.io/>
Weaviate Developers: <https://weaviate.io/developers/weaviate>
pgvector: <https://github.com/pgvector/pgvector>
OpenTelemetry: <https://opentelemetry.io/>
Grafana: <https://grafana.com/>

There is also CrewAI but I didn't include it because I'm currently working with it - so I will probably come back with a separate thread if I find it extraordinary.

If you're building something similar and want a second pair of eyes, drop a reply or send me a message and I will share exact patterns that worked for me.
If you read this whole thing - you're an absolute legend.

?

kapakie - September 30, 2025, 3:32 pm

Nice read, you covered it very well

Hackers - September 30, 2025, 8:46 pm

Awesome, this is definitely a hot topic

here are my two favorite comments from you:

Over-modeling the schema. Keep it small ---- grow with real questions!!!!

Ignoring timestamps. Stale edges look authoritative to LLMs.

We specifically dealt with these two in a past project and came to the same conclusions.

Timestamps for data is absolutely required for organizations that can change processes, priorities, and update info over time.

Even if it's old static data... timestamp it, order it, etc if you can.

If you can't, make sure you go through and remove contradictions that can't be assessed properly.

"I was sad yesterday, I am happy today" is much different than "I am sad, I am happy".

Allowing our system to adjust with real user feedback (reject, approve, real comments) was way more useful. It's "obvious" but at the same time, the improvements we saw after implementing this was night & day, to the point where it was not as "obvious" how impactful it would be ahead of time.

> Be careful of contamination / bad users / mistakes in data.

> We actually don't allow the overall system (in the specific project I'm referencing) to be influenced by everyone. Customers are isolated, and within customer accounts there's separation between entry types, entries, etc.

itcorp - October 1, 2025, 10:20 am

Interesting read my friend, definitely high quality content.

Oh Well - October 3, 2025, 1:47 am

(Sep 30, 2025 03:32 PM)kapakie Wrote: [\[CITE\]](#) Nice read, you covered it very well

(Sep 30, 2025 08:46 PM)enki Wrote: [\[CITE\]](#) Awesome, this is definitely a hot topic

here are my two favorite comments from you:

Over-modeling the schema. Keep it small ---- grow with real questions!!!!

Ignoring timestamps. Stale edges look authoritative to LLMs.

We specifically dealt with these two in a past project and came to the same conclusions.

Timestamps for data is absolutely required for organizations that can change processes, priorities, and update info over time.

Even if it's old static data... timestamp it, order it, etc if you can.

If you can't, make sure you go through and remove contradictions that can't be assessed properly.

"I was sad yesterday, I am happy today" is much different than "I am sad, I am happy".

Allowing our system to adjust with real user feedback (reject, approve, real comments) was way more useful. It's "obvious" but at the same time, the improvements we saw after implementing this was night & day, to the point where it was not as "obvious" how impactful it would be ahead of time.

> Be careful of contamination / bad users / mistakes in data.

> We actually don't allow the overall system (in the specific project I'm referencing) to be influenced by everyone. Customers are isolated, and within customer accounts there's separation between entry types, entries, etc.

Good points!

(Oct 1, 2025 10:20 AM)itcorp Wrote: [CITE](#) Interesting read my friend, definitely high quality content. Thank you, mate!

greenhunter2 - July 14, 2026, 2:23 am

Good points!