BRIAN GALVAN

Applied AI Engineering · Document Intelligence

WHITE PAPER

When Vector A.I. Search Isn't Enough

Extending Retrieval-Augmented Generation with Knowledge-Graph Memory: A Measured Case Study on 11,794 Pages of Aircraft Maintenance Records

Author	Brian Galvan
Date	July 2026
Version	1.0 · Public
Audience	Engineering leaders, applied-AI practitioners, Business & C-Suite executives and anyone operating retrieval-augmented generation in production
Method	All costs, token counts, and results in this paper are measured from live instrumentation, not estimated. Identifying details of the case-study aircraft and its vendors have been removed or generalized; every number is reported exactly as recorded.

Abstract

Retrieval-augmented generation (RAG) built on vector search is the default architecture for making large document collections conversational, and it deserves its popularity: it is cheap, fast, and remarkably good at finding the page that sounds like your question. But it carries structural limitations that no amount of prompt engineering or model upgrades will fix, because they are properties of the retrieval pattern itself, not of any particular model. It truncates enumerations, it cannot detect absence, it fragments multi-document answers, and it never notices when two documents disagree.

This paper does three things. First, it names and explains those failure modes precisely, so that practitioners can recognize them in their own systems. Second, it presents a complete, instrumented case study: a real 11,794-page corpus of business-aircraft maintenance records, twenty years deep, where a knowledge-graph memory layer was built alongside an existing vector-search deployment for a measured total of roughly $53, then benchmarked head-to-head on seven questions designed to probe each failure mode. Third, it reports what production actually required: the hidden billing traps, the failure modes of the extraction pipeline itself, the hybrid architecture that emerged, and a decision framework for readers weighing the same investment.

The headline result: on questions of completeness, traceability, and absence, the graph did not incrementally improve on vector search. It answered questions vector search is structurally incapable of answering, at roughly $0.003 per page, and it exposed one of the vector system's confident answers as a retrieval artifact contradicted by the underlying records.

1. Introduction: The Confident, Incomplete Answer

Ask a well-built RAG system a narrow question (“what did the shop find during the 2015 inspection?”) and it will very likely give you a good answer with a citation. Ask it a sweeping one (“list every organization that ever touched this asset”) and it will give you an answer that *looks* just as good: fluent, specific, confidently phrased, and complete-sounding. The difference is that the second answer is almost certainly missing entries, and nothing in the system, or the answer, will tell you so.

This is the central danger of production RAG, and it is worth stating plainly: **vector retrieval fails silently, and it fails in the direction of confident incompleteness.** The model composes its answer from the ten or twenty text excerpts that scored highest on semantic similarity. Whatever those excerpts happen to contain becomes the answer. Whatever they omit simply does not exist, as far as the model is concerned. There is no error message for “the other thirty relevant pages were never retrieved.”

For casual question-answering this is a tolerable flaw. For domains where the questions that matter are about *completeness* (legal discovery, medical history reconstruction, financial audit, regulatory compliance, asset due diligence), it is disqualifying. Those domains do not ask “find me something relevant.” They ask “find me *everything*, and prove there is nothing else.”

I ran into this ceiling in a real deployment: a document-intelligence portal built over twenty years of maintenance records for a midsize business jet. The vector-search layer worked well, cost almost nothing, and users liked it. And yet the questions that mattered most commercially (Is this paperwork complete? Is the maintenance timeline continuous? What is missing?) were exactly the questions it answered worst. This paper is the record of what we did about it, with every cost and result measured, not estimated.

2. Four Structural Failure Modes of Vector Retrieval

The limitations described here are not bugs in any vendor's product. They follow directly from the shape of the retrieval pattern: embed the corpus into vectors, embed the question, return the top-k most similar chunks, and generate from those. Each failure mode below is a logical consequence of that design, which is why switching embedding models or raising k mitigates but never eliminates them.

2.1 Truncation: Top-k Physically Caps Every Enumeration

A retrieval system returns a fixed number of excerpts. If the true answer to “list all vendors” is spread across forty pages and the retriever surfaces ten chunks that mention seven vendors, the answer will contain seven vendors, delivered with complete confidence. Raising k helps until it doesn't: context windows are finite, retrieval precision degrades as k grows, and no k guarantees coverage of an enumeration whose members are scattered unpredictably. The system is not wrong about what it found; it is silent about what it never looked at. In the case study, this failure mode alone accounted for a 3× undercount (7 organizations found versus 22 actually present in the records).

2.2 Absence Blindness: You Cannot Retrieve a Chunk That Doesn't Exist

The most consequential failure mode, and the least discussed. Questions about what is *missing* (“which installed parts have no release paperwork?”, “which patients never received a follow-up?”, “which contracts lack a signed amendment?”) require knowing the full set of things that exist, the full set of documentation that exists, and computing the difference. Similarity search cannot do this even in principle: the evidence of absence is not written on any page, so there is no chunk to retrieve. A RAG system asked an absence question will either hedge honestly or, worse, guess fluently. Neither is a punch list you can act on.

2.3 Fragmentation: Multi-Document Answers Arrive in Pieces

Real-world facts are scattered. A single maintenance event might live partly in a logbook entry, partly in a work order, partly in a regulatory release tag, connected by shared identifiers but stored pages or boxes apart. Top-k retrieval sees each fragment independently and has no mechanism for recognizing that three excerpts describe one event, or that the entity in chunk 4 is the same entity as in chunk 17. The generated answer stitches together whatever fragments arrived, often mixing events or dropping the connective tissue that makes the answer trustworthy.

2.4 Contradiction Invisibility: Disagreements Never Surface

If two documents record the same serial number differently (a scanning error, a transcription slip, an OCR misread of handwriting), a vector store holds both versions peacefully, in separate chunks. Each bad reading affects only the queries that happen to retrieve it; most questions see only the good pages, so the corruption stays invisible for years. The system has no concept of "the same real-world thing," so it can never notice that its own corpus disagrees with itself. As §14 will show, this is precisely where a knowledge graph delivers an unexpected second service: it audits your data quality as a side effect of existing.

> **The pattern behind all four:** vector search operates on *text that sounds relevant*. The hard questions operate on *things and their relationships*: entities, connections, completeness, identity. Those are different data structures, and no retrieval tuning converts one into the other.

3. Key Concepts, in Plain English

This paper uses a handful of AI-infrastructure terms. Here is what each one actually means, because the entire argument turns on the differences between them. Practitioners can skim; this section exists so that a non-specialist can read everything that follows.

OCR (Optical Character Recognition). Software that looks at a *picture* of a page (a scan or photo) and converts it into machine-readable *text*. Scanned archives are pictures; nothing downstream (search, AI, databases) can use a picture directly. OCR is the toll booth every page passes through once. Good OCR engines report a *confidence score* per page: their own estimate (0 to 1) of how sure they are about the characters they read. 0.97 on printed text is near-perfect; on 1990s handwriting it means "mostly right, occasionally a misread digit." That per-page score turns out to be operationally precious, as §15 will show.

Token. AI models don't bill by page or by word. They bill by *token*, roughly a word-chunk (about 4 characters of English on average). Every piece of text sent *to* a model (input tokens) and produced *by* it (output tokens) is metered and charged. When this paper quotes costs, they come from counting tokens on every single call with instrumentation hooks, not from estimates.

Embedding and vector search. An *embedding* converts a chunk of text into a long list of numbers (a "vector") that captures its *meaning*, so that two chunks about the same topic end up numerically close together, even if they share no exact words. "Vector search" means: convert the user's question into the same kind of vector, then find the stored chunks whose vectors sit closest to it. This is genuinely powerful: ask about "turbine damage" and it finds pages saying "blade erosion" without the word "damage" appearing anywhere.

Top-k retrieval and RAG. A vector search doesn't return *everything* relevant. It returns the *k best-matching chunks* (maybe 10 or 20), and the AI writes its answer using only those. This pattern is called RAG (Retrieval-Augmented Generation): retrieve some excerpts, then generate an answer

from them. The strength: fast, cheap, works on any question. The built-in weakness is §2 of this paper.

Knowledge graph. A different way of storing what the documents say. Instead of keeping text chunks, you extract the *entities* (every part, serial number, organization, work order, date) as nodes in a network, and the *relationships* between them as connecting edges: Fuel pump S/N 0768 →[was installed on]→ the left engine →[was overhauled by]→ the engine MRO →[under]→ a specific work order. Once facts live as a network, questions become *traversals*: "trace this serial" means "walk every edge touching this node." Crucially, a graph can answer questions about **absence** ("which part nodes have no paperwork edge?"), which no amount of similarity search can do.

Entity extraction. How the graph gets built: an AI model reads each chunk of text and is instructed to output, in a rigid machine-readable format, every entity and relationship it finds. This is the expensive step (every page flows through the model) and the fallible one (the model can misread, and the rigid format can come back malformed; both happened in this project and are documented candidly in §11).

Reasoning or "thinking" tokens. Newer AI models can privately "think": generate internal scratch-work before their visible answer. Useful for hard problems; financially dangerous for bulk work, because **the invisible scratch-work is billed as output tokens**. A model that thinks for 700 tokens and answers in 180 costs you 880. This project measured 75% of billed output going to invisible thinking before it was explicitly disabled (§11).

Prompt caching. When thousands of requests share the same long preamble (instructions, format definitions), the provider can recognize the repetition and charge a discounted rate for the repeated part. This is why, counterintuitively, the full corpus in this study cost *less per page* than the small pilot: at scale, most of each request was cache-discounted. Small pilots overestimate large runs.

4. What a Knowledge Graph Changes

The combination of a knowledge graph with LLM-based question answering has come to be called *GraphRAG*, a term popularized by Microsoft Research's open-source project of the same name in 2024. The ecosystem has matured quickly: graph databases like Neo4j offer LLM integration paths, orchestration frameworks like LlamaIndex ship property-graph indexes, and standalone open-source engines now package the whole extract-store-query pipeline. This project used **cognee**, an open-source Python library (Apache 2.0, roughly 25,000 GitHub stars at this writing) that runs entirely on infrastructure you control: no account, no subscription, no new vendor holding your data. Its only external calls go to whatever LLM API key you already use.

Cognee builds its "memory" in three stages:

1. **Cognify (ingestion).** Split each document into chunks, send each chunk to an AI model (this project used a cheap, fast model tier) with instructions to extract entities and relationships, generate a summary of each chunk, and compute embeddings for everything. This is where nearly all the cost lives, and it's paid once per document.

2. **Store.** Entities and edges go into a graph database, embeddings into a vector store, bookkeeping into SQL. In a test setup these are all just local files (no servers to run); in production they live wherever your services do.
3. **Search.** Cognee ships roughly 17 retrieval modes. The one that matters here is graph completion: it finds the entities relevant to your question, *walks their connections* to pull in related facts, and hands that subgraph (not raw text excerpts) to the model to compose an answer. The answer is built from structured knowledge, with source document IDs preserved.

An important subtlety: a graph system still *uses* embeddings. The graph doesn't replace vector search; it's built on top of it. The difference is what gets stored and what the AI reasons over at question time: connected facts instead of loose excerpts.

> **The one-line difference:** vector search answers "which pages sound like this question?" A knowledge graph answers "what do we know about this thing, and how is it connected to everything else we know?"

5. The Case Study: Twenty Years of Aircraft Maintenance Records

The corpus is real: six boxes of maintenance documents for a midsize business jet that has been in service for more than two decades. 296 PDFs. 11,819 scanned pages. Logbooks, work orders, FAA Form 8130-3 airworthiness release tags, traceability bundles, OEM manuals, weight-and-balance records; a mix of clean print, dot-matrix output, fax artifacts, rubber stamps, and twenty years of handwriting.

A note on anonymization: because this paper is public, the aircraft's registration, component serial numbers, work-order numbers, and vendor names have been removed or generalized. Every count, cost, timing, and result is reported exactly as measured; only the identifiers have been altered.

Aircraft records are worth studying because they are a *graph-shaped* dataset wearing a document costume. The same physical part appears in an engine logbook entry, an FAA release tag, a work-order envelope, and a traceability bundle, connected by part number, serial number, date, and work order, but physically scattered across boxes filled over twenty years. The questions that matter commercially (pre-purchase due diligence, audit, compliance) are mostly *join* questions, questions whose answers live in the connections:

- Trace this serial number across every document it touches.
- Which installed parts have release paperwork, and which don't?
- What did this repair station do to this engine, ever?
- Is the maintenance timeline continuous, or are there gaps?

If this shape sounds familiar from your own domain, it should. A litigation document set is the same structure (parties, agreements, amendments, correspondence, all cross-referencing). So is a patient's medical history (encounters, medications, labs, referrals). So is a supply chain's certification trail, an M&A data room, and a construction project's submittal log. The aviation corpus is simply an unusually honest specimen: the joins are explicit (serial numbers, work orders), the stakes are regulatory, and the ground truth is checkable against physical scans.

Top-k retrieval sees every one of the questions above as "find me some relevant excerpts." The graph sees them as what they actually are: walks through a network of connected facts. That mismatch, not any deficiency in the underlying language models, is why a well-built RAG chat produces beautiful answers to narrow questions and incomplete answers to sweeping ones.

6. The Baseline: What Was Already Running

Before the graph experiment, the deployment already gave users three access modes (chat, voice, file browser) backed by two data tracks:

Track	Technology	Contents	Cost to build
Narrative / search	Managed vector search (Gemini File Search: embeddings plus built-in OCR)	All 296 documents	~$2-3 total
Structured / numeric	Document AI OCR + premium vision-model extraction → SQL	One box only (1,823 pages, 19,071 extracted fields)	~$63-75

The structured track was deliberately paused after one box. It worked (19,071 verified data fields with page-level citations), but it was built by having a premium vision model *look at page images*, which is the most expensive way to read a page, and hidden thinking-token billing (§3) roughly doubled the expected cost. Extending that approach to all six boxes was estimated at $500-600.

The open question: *is there a cheaper way to get machine-usable structure out of the records?* The insight this project tests: the expensive part of the earlier run wasn't reading the pages (OCR text costs almost nothing); it was making a premium vision model stare at images. If a cheap text model can extract structure *from OCR text you already have*, the economics change completely. A knowledge-graph engine is the packaged version of exactly that idea.

7. The Pilot: One Engine's Paper Trail (611 Pages, $4.33)

Design principle: never scale what you haven't measured. Before spending real money, the pilot targeted one engine's interlocking paper trail, documents chosen specifically because they *should* cross-reference each other, making cross-document linking testable:

Document	Pages
Engine logbook (subject engine)	321
Engine MRO part-change record (same engine)	44
FAA 8130-3 release-tag bundle	124
Traceability bundle from a major independent MRO (2010)	122

The input was already-paid-for OCR text sitting in the pipeline database. The graph engine never touched a scanned image and needed zero new OCR spend. Each page carried a provenance header (document ID, type, title, serial, page number) so that graph answers can cite back to the exact scan, preserving the compliance-grade rule that every claim must be traceable to a page a human can look at.

Measured results:

Metric	Value
Pages ingested	611
Wall time	24 minutes
AI calls	1,345 (fast model tier, thinking disabled)
Tokens	2.46M in / 1.10M out, **0 thinking**
Embeddings	3,675 calls / 837K tokens
Total cost	**$4.33 → $0.0071/page**

For scale: the earlier vision-model extraction (thinking enabled) cost roughly $0.019-0.041 per page depending on how you count. The text-extraction route is 3-6× cheaper per page, running on text the OCR step had already produced for a tenth of a cent per page.

8. The Benchmark: Seven Questions Vector Search Can't Answer Well

Seven questions were designed, each aimed at a specific structural weakness from §2, and run against both systems: the live vector-search deployment and the new graph.

An honesty note on the first round: it was *not* corpus-fair. Vector search had all 187 records documents indexed; the graph had only the 611-page pilot slice. Round one is therefore capability evidence, not a score. The fair rematch on equal corpora follows in §9.

#	Question type	Failure mode probed (§2)
1	Full maintenance timeline	Fragmentation: top-k excerpts ≠ a complete ordered history
2	Logbook ↔ release-tag join	Fragmentation: answer spans two unrelated-looking documents
3	**Missing paperwork (absence)**	Absence blindness: you cannot retrieve a chunk that doesn't exist
4	Per-organization work rollup	Fragmentation: facts scattered across 44+ pages
5	Same part in two paperwork trails	Contradiction and identity: one entity in two places
6	Enumerate all organizations	Truncation: top-k physically caps enumerations
7	Temporal gaps > 3 years	Truncation and fragmentation: requires the *complete* sequence first

The Standout: Question 3 (Absence Detection)

Asked: *"Which parts or components mentioned in the engine records do NOT have any matching 8130 tag or traceability paperwork?"*

Understand why this question is special. An 8130-3 is the FAA's airworthiness release form, and a part installed without one is a paperwork gap that a pre-purchase inspector will flag. Answering requires knowing every part that *was* installed, knowing every tag that *does* exist, and subtracting. A retrieval system can't subtract, because the evidence of absence isn't written on any page it could retrieve.

Vector search behaved exactly as the theory predicts: it hedged. It surfaced one component marked "UNK" and then conceded that *the records do not provide matching tags for every sub-component listed in the logbook*. Honest, but useless as a punch list.

The graph returned a confident, enumerated list of **14 components lacking release paperwork** (the digital engine controller, the exhaust nozzle, 56 low-pressure turbine blades, the fuel control, four carbon seals, the combustion liner, fuel manifolds, a thermocouple harness, a speed transducer), each with its part number. The graph can do this because absence is a *query over structure*: find part-nodes with no paperwork-edge.

Other Clear Graph Wins in Round One

- **Q4, shop-visit rollup:** the graph assembled the engine MRO's April 2015 visit into one coherent event (a single work order, seven named inspections, 56 new turbine blades installed under a service bulletin) with the exact logbook page citations. Vector search returned fragments of *different* visits without unifying them.
- **Q6, enumeration:** seven organizations, each with its role, including the airframe OEM's own service arm, which was missing from vector search's answer.

- **Q2, the cross-document join:** release-tag part numbers correctly tied to the engine's parts record.

The Honest Counterexample: Question 7, Round One

The graph reported a "7.8-year maintenance gap (2015-2023)." Wrong: 2018 and 2021 work exists, in documents outside its four-document slice. Vector search, seeing the whole corpus, found it. The lesson is not that graphs are bad at temporal reasoning; it's that **a graph is confidently wrong about exactly the material it was never given.** An incomplete graph doesn't say "I don't know." It says "there's a gap," and it sounds authoritative. For compliance-flavored questions, corpus completeness is non-negotiable, which is why the full-library build below preceded any production judgment.

A Bonus Find: The OCR Cross-Check

The two systems disagreed on a controller serial number: one OCR engine read the handwritten serial as ending in **BY1189**; the other read the same characters as **311189** (a handwritten "BY" misread as "31," or vice versa). One of them is wrong, and only a human looking at the scan can say which. This is the known failure mode of digitizing handwriting, and it surfaced only because two independent OCR engines read the same page. It is the concrete argument for why citation-back-to-scan design remains mandatory no matter which engine answers: the AI's job is to find and connect; the scan remains the authority.

9. Scaling Up: The Fair Test on the Full Corpus

Step 1: OCR backfill. The 9,995 pages that had never been OCR'd went through a commercial OCR service in 2.3 hours for **$14.99 with zero failed pages**. Per-folder average confidence was 0.965-0.974, with printed OEM manuals scoring highest, exactly as expected, since OCR confidence tracks print quality. The complete corpus (**11,794 pages, 19.0 million characters of verbatim text**) now lives in the pipeline database. Worth underlining: *independent of anything to do with knowledge graphs*, the archive now has a permanent, searchable, machine-readable text layer that any future tool can build on without ever re-scanning a page. If you take one tactical lesson from this paper, take this one: OCR your archive once, store the text and the confidence scores, and every subsequent AI experiment becomes an order of magnitude cheaper.

Step 2: full-corpus graph build. All 296 documents cognified into one graph:

Metric	Value
Documents / pages graphed	296 / 11,794 (all six boxes)
Extraction cost (measured, incl. crash reruns)	**~$33**
Processing time	~1.5 h of extraction across attempts; final resumed pass 24 min
Graph database size	206 MB (embedded graph DB, local files)

Metric	Value
Query latency	30-72 s per graph question

The full-corpus extraction came in **well under** the $84 straight-line projection from the pilot. The reason is prompt caching (§3): all ~2,200 extraction calls share the same long instruction preamble, and at scale the provider recognizes and discounts the repetition. Small pilots overestimate large runs; budget accordingly, but verify with metering.

The Seven-Question Rerun on Equal Corpora

With both systems now seeing the complete records, the graph's answers changed character entirely:

- **Q1 (timeline):** a **~55-event maintenance history of the subject engine spanning 2002-2023**, drawn from at least seven documents across multiple boxes (early acceptance testing, a 2004 rotor replacement, major periodic inspections in 2006 and 2009, the 2014-2015 engine-MRO campaign, 2021 gearbox seal work, 2023 inspections by an engine-services provider), each event tagged with its source document. Vector search's best effort was ~15 events with visible holes.
- **Q3 (missing paperwork):** the absence list grew to **18 components, now with part AND serial numbers** (disks, an impeller, a shaft, a seal plate, nozzle assemblies, and more). This is a document-the-gaps punch list a records auditor could work from directly. Vector search still could not produce one, and never will, for the structural reason in §2.2.
- **Q4 (MRO rollup):** now complete across **three work orders spanning seven years** (2014, 2015, and 2021), each with its parts list. Previously, the 2021 work was invisible to the pilot-slice graph *and* the 2014-2015 detail was invisible to vector search: each system had half the picture, and the full graph has all of it.
- **Q6 (organizations): 22 organizations with FAA repair-station certificate numbers and roles**, versus vector search's 7. The additions included component OEMs, an oil-analysis lab, an NDT shop, and rotables shops. This is what “top-k truncates enumerations” looks like in practice: vector search wasn't wrong; it just stopped at what its ten retrieved excerpts happened to mention.
- **Q7 (gaps): the verdict flipped.** On the full corpus, the graph identified a single ~3.6-year quiet period (early 2016 to late 2019), plausible, since the aircraft changed hands and activity slowed. Meanwhile **vector search's answer, a claimed 2009-2014 gap, is refuted by the graph's own timeline**, which shows recorded activity in 2010, 2011, 2012, and 2013 that the retrieval simply never surfaced. Read that carefully: the system that was “confidently wrong from incomplete data” in round one is now the *vector search*, permanently, because its incompleteness is per-question and structural, while the graph's was a one-time corpus problem that $48 fixed.

(Verification note: graph answers carry document citations, but the individual claims above were spot-checked rather than exhaustively hand-audited against scans. The citation-to-scan design

exists precisely to make that audit a five-minute job per claim, and any compliance-grade use should perform it.)

10. The Hybrid Architecture

Complement, don't replace. The two systems answer different question classes, fail in different ways, and are both cheap enough to run side by side:

Layer	Engine	Question class
Search & cite	Managed vector search (existing, unchanged)	“Find the page that says X”: narrative retrieval with citations, 7-20 second answers
Reason & audit	Knowledge graph (new)	Traces, joins, rollups, enumerations, absence and completeness checks, 30-120 second answers
Verbatim record	SQL pages table (full OCR text + confidence)	Full-text search, future extractions, ground truth

In practice, the application's chat layer routes questions, or consults both engines and merges: “where does it say…” goes to vector search; “trace / list all / what's missing…” goes to the graph. Voice interfaces deliberately stay on the fast path (a spoken conversation cannot absorb a minute of dead air). The graph deploys as ordinary infrastructure: local files plus a small Python service on a private server, behind the application's existing authenticated API layer, with a one-query-at-a-time guard and a memory ceiling so a heavy graph question can never crowd out the rest of the application.

11. Engineering Findings: What It Actually Took

Recorded candidly, because a build-vs-buy judgment depends on these as much as on the benchmark wins. None of them is a reason not to proceed; all of them are the difference between a demo and a system.

1. **Hidden thinking-token billing.** Reasoning-capable models privately “think” before answering and bill that scratch-work as output tokens. Measured on this project: **75% of billed output was invisible thinking** until it was explicitly disabled, which cut cost 3.2× with extraction quality intact. Standing rule worth adopting: *never run bulk extraction on a reasoning model without explicitly configuring the thinking budget, and always meter actual token usage with a counting hook rather than trusting estimates.* This one paragraph has paid for the reader's time with this paper.
2. **Thinking-off has its own failure mode.** With reasoning fully disabled, roughly 10% of extraction calls returned malformed output: the model would echo back the format definition instead of filling it in, or append garbage after valid data. (Demand rigid machine-readable

output from a model that isn't allowed to think, and it occasionally sleepwalks.) Mitigation: re-enable a small thinking budget for just the affected call type, retry failed calls, and checkpoint progress per document so a single bad response can't kill an hours-long run. After mitigation, failures halved and the remainder were absorbed by retries; the final pass ran clean end to end.

3. **Ingestion grain matters enormously.** Feeding the engine 11,794 tiny per-page items stalled its loader for three hours (bookkeeping overhead is per-item); regrouping the same text into 296 per-document files with inline page markers loaded in 45 seconds. Identical content, ~200× difference. Every batch-processing tool has a preferred grain; find it experimentally before scaling.
4. **The operational contrast, stated plainly.** Managed vector search: upload PDFs, done. The provider OCRs, chunks, embeds, hosts, and scales everything, and this entire corpus cost ~$3 to index. The graph: a real pipeline with a database, an extraction budget, failure modes, retry logic, and a 206 MB artifact you are responsible for hosting and rebuilding. The graph's capabilities are real and demonstrated; they are paid for in *ownership*, not just dollars. Any honest recommendation includes both facts.

12. Cost Ledger

Every figure below is measured from live token metering, except the final row, which is the avoided estimate for the originally planned approach.

Item	Cost	Notes
Prior vision-model extraction, one box (for context)	~$63-75	The approach this project supersedes
Vector-search indexing, whole corpus	~$2-3	Still the search backbone
Graph pilot: 611-page slice	$4.33	Measured, $0.0071/page
OCR backfill: 9,995 remaining pages	$14.99	Permanent asset, zero failures
Full-corpus graph build (incl. crash-rerun waste)	~$33	Under projection; prompt caching
Probes, smoke tests, graph queries	~$1	
Total project spend	**~$53**	
Originally planned alternative (vision model, all boxes)	**$215-425**	Avoided

The number worth internalizing is not $53. It is **$0.003-0.007 per page** for graph extraction over pre-existing OCR text, versus $0.02-0.04 per page for vision-model extraction. At archive scale, that ratio is the difference between a project that gets approved and one that doesn't.

13. An Unexpected Benefit: The Graph as a Data-Quality Auditor

During deployment testing, a simple sanity question ("list the engines and APU with serial numbers") returned something quietly remarkable: the graph reported the auxiliary power unit's serial as ending in **690, "also identified as" ending in 609**.

The registry, and the APU logbook covers, agree on 690. So why did the graph hedge?

Somewhere in 11,794 pages, at least one document records that serial in a form OCR read with the final digits transposed: a handwritten 9 and 0 swapped, a smudged stamp, a worn carbon copy; classic handwriting-digitization noise. A vector-search system would never notice. The bad reading just sits in one chunk, waiting to mislead whichever question happens to retrieve it, while every other question sees the good pages.

The graph *can't* ignore it. During entity extraction, the engine tried to resolve every mention of the APU into a single entity; that's the whole point of a knowledge graph, one node per real-world thing. When mentions disagree about an identifier, the conflict becomes *visible in the structure*: the APU node ends up carrying two competing serials, and the answer surfaces both instead of silently picking one. The same dynamic surfaced the controller-serial disagreement in §8.

> **Vector search hides data-quality problems** (each bad reading affects only the queries that happen to retrieve it). **A knowledge graph concentrates them** (all mentions collide on one entity, and disagreements surface). The graph doesn't just answer questions; it audits the corpus as a side effect.

For anyone operating a document AI system over messy real-world archives, this second service may justify the graph on its own. Your corpus contains errors you do not know about. A graph is the only architecture in common use that will *volunteer* them.

14. Productionizing: The Slow Path, the Fast Path, and the Wait

A production service nobody can reach is a trophy, not a tool. The graph was wired into the application's existing chat as an explicit mode the user chooses, and the interface design around that choice carries lessons of its own.

Why a mode, not a replacement. The two engines have opposite personalities: vector search answers in 7-20 seconds and excels at "find me the page"; the graph takes 30-120 seconds and excels at "connect everything you know." Forcing every question through the slow, deep path would make the common case miserable; hiding the deep path would waste it. So the user chooses, and the interface teaches the choice: the deep mode is styled in a visually distinct color, and the input placeholder changes to nudge toward the questions the graph is actually good at (*"trace a component, list all of something, find what's missing…"*).

Design for the wait. A 60-second spinner with no explanation reads as "broken." The deep-analysis loading state makes the wait feel like work being done, because it is: a live elapsed counter plus rotating hints that narrate the actual mechanics (*"Following serial numbers between logbooks and release tags…"*). This small thing changes whether people ever use the feature twice.

A starter that sells the feature. The chat's welcome screen gained one suggestion chip that auto-switches to deep mode and fires the showcase question (the 18-component missing-paperwork list). The best demo of a graph is one tap from a fresh login.

Closing the Loop: Machine-Directed Human Review

Everything above *detects* problems: low OCR confidence scores, identifier conflicts the graph surfaces. Detection without a place to act is just a smarter pile of worries, so the final build was a review workflow where a human settles what the machines flagged. Its philosophy generalizes to any document AI deployment:

- **Machine-directed attention.** Nobody proofreads 11,794 pages, and nobody should. The pipeline already knows where its doubts are: per-page OCR confidence (sort ascending; the shakiest pages float up) and graph-surfaced identifier conflicts. Every queue item exists because a machine registered a specific, explainable doubt. Human effort lands exactly where it changes an answer, and nowhere else.
- **A correction is a layer, never an eraser.** A human verdict never edits the stored OCR text; it writes a new row in a separate reviews table (verdict, corrected text if any, a free-text note, who, when). Overwriting destroys the chain of custody; layering preserves it forever, keeps every correction reversible, and lets the note field carry human context no scan can ("stamp illegible; serial confirmed from the matching release tag"). For any audit-sensitive domain, this is the design decision that matters most.
- **Attribution must be server-side.** The reviewer's identity on every verdict comes from the server-side login session, never from anything the browser sends, so a tampered request cannot forge someone else's name onto a review. Trustworthy attribution is what makes the layer audit-grade rather than a shared scratchpad.
- **Conflicts require a resolution note.** The tool refuses to mark a graph-surfaced conflict resolved with an empty note, because "resolved" without "how" is worthless to the next person who hits the discrepancy.

What it found in its first minute. While verifying the deployed endpoints, the queue's #1 item, the single worst page in the entire corpus at 54% OCR confidence, turned out to begin: *2NLEKCEDED · BICKED…* Stare at it for a second: that is **"SUPERSEDED" and "AIRPLANE" read upside-down.** The page had been scanned rotated 180°; the OCR dutifully read the inverted glyphs as garbage, the confidence score collapsed exactly as it should, and the queue put it at position one. No AI heroics, no human drudgery: the scoring did the finding, and a human does the two-minute fix. The tool justified its existence before its first real user session.

With detection, review, and (in the next build) rebuild-time consumption of corrections, the system forms a closed quality loop: digitize, understand, serve, detect, settle, improve. Using the system

generates the maintenance list for the system. That is the difference between a digitization *project* (finished, aging, decaying) and a records *practice* that compounds.

15. Should You Add a Graph? A Decision Framework

A knowledge graph is not a default upgrade. It is an investment with an ownership cost, and plenty of RAG deployments genuinely don't need one. Use these tests.

Your Corpus Is Probably Graph-Shaped If...

- The same real-world entities (people, parts, parties, patients, accounts) recur across many documents under shared identifiers.
- The questions that matter are joins, traces, rollups, or enumerations rather than “find the passage.”
- Absence questions have commercial or regulatory weight: what's missing, what's undocumented, what never happened.
- Answers must survive audit: every claim needs a citation trail back to a source page.
- The corpus accumulated over years from multiple sources, making internal contradictions likely and valuable to surface.

You Probably Don't Need One If...

- Questions are overwhelmingly narrative and local (“summarize this section,” “what does the policy say about X”).
- The corpus is small enough that generous retrieval plus a large context window effectively reads everything relevant.
- Nobody will act differently on completeness guarantees; “a good answer” is genuinely good enough.
- You cannot commit to operating a pipeline: extraction budgets, retries, rebuilds, and an artifact you own.

If You Proceed, in This Order

1. **OCR first, everything, once.** Store text and per-page confidence scores in a database you control. It is the cheapest step, it is a permanent asset independent of every downstream tool, and it makes each subsequent experiment nearly free.
2. **Pilot on a deliberately interlocking slice.** Choose documents that *should* cross-reference each other, so linking is testable. Measure everything with token-counting hooks.
3. **Design the benchmark before the build.** Write the questions that probe truncation, absence, fragmentation, and contradiction in *your* domain, and run them against your existing RAG first, so the comparison is honest.

4. **Disable or cap model thinking for bulk extraction, then meter.** Expect a malformed-output rate when thinking is fully off; build retries and per-document checkpoints before the long run, not after it crashes.
5. **Only scale after the fair test.** An incomplete graph is confidently wrong about exactly the material it was never given. Corpus completeness precedes production judgment.
6. **Ship it as a mode beside your fast path, not a replacement,** and route by question shape. Then close the loop: surface machine doubts to humans, layer corrections, and feed them back into rebuilds.

16. Conclusion

Vector search is not broken, and nothing in this paper argues for abandoning it. For the questions it is shaped for, it remains astonishing value: this project's entire corpus was indexed for about three dollars. But its failure modes are structural, silent, and biased toward confident incompleteness, and in domains where completeness *is* the product (audit, diligence, compliance, discovery), that bias is disqualifying on exactly the highest-stakes questions.

The measured finding of this case study is that the fix is neither exotic nor expensive. For roughly $53 all-in (about three-tenths of a cent per page over pre-existing OCR text), a knowledge-graph layer answered the questions the vector system structurally could not: a 55-event timeline where retrieval managed 15, a 22-organization enumeration where retrieval found 7, an 18-item missing-paperwork punch list where retrieval could only hedge, and the exposure of a confidently asserted five-year “maintenance gap” as a retrieval artifact. Along the way, the graph volunteered data-quality defects (transposed serial digits, an upside-down scan) that had sat invisible in the corpus for years.

The deeper point is architectural. Text that sounds relevant and things with relationships are different data structures, and systems built on the first cannot be tuned into answering questions about the second. If the questions that matter in your domain are about completeness, connection, and absence, then the graph is not an optimization of your RAG stack. It is the missing half of it.

> Retrieval finds pages. Graphs know things. Production systems that matter need both, and now you have the measured numbers to argue for it.

About this paper: all costs, token counts, timings, and benchmark results are measured from live instrumentation on a production deployment, not estimated, except where marked as projections. Raw question and answer transcripts for both systems, both benchmark rounds, are preserved in the project archive. The case-study aircraft's registration, serial numbers, work-order numbers, and vendor identities have been removed or generalized for publication.