Been building RAG systems for mid-size enterprise companies in the regulated space (100-1000 employees) for the past year and to be honest, this stuff is way harder than any tutorial makes it seem. Worked with around 10+ clients now - pharma companies, banks, law firms, consulting shops. Thought I'd share what actually matters vs all the basic info you read online.

Quick context: most of these companies had 10K-50K+ documents sitting in SharePoint hell or document management systems from 2005. Not clean datasets, not curated knowledge bases - just decades of business documents that somehow need to become searchable.

Document quality detection: the thing nobody talks about

This was honestly the biggest revelation for me. Most tutorials assume your PDFs are perfect. Reality check: enterprise documents are absolute garbage.

I had one pharma client with research papers from 1995 that were scanned copies of typewritten pages. OCR barely worked. Mixed in with modern clinical trial reports that are 500+ pages with embedded tables and charts. Try applying the same chunking strategy to both and watch your system return complete nonsense.

Spent weeks debugging why certain documents returned terrible results while others worked fine. Finally realized I needed to score document quality before processing:

- Clean PDFs (text extraction works perfectly): full hierarchical processing
- Decent docs (some OCR artifacts): basic chunking with cleanup
- Garbage docs (scanned handwritten notes): simple fixed chunks + manual review flags

Built a simple scoring system looking at text extraction quality, OCR artifacts, formatting consistency. Routes documents to different processing pipelines based on score. This single change fixed more retrieval issues than any embedding model upgrade.

Why fixed-size chunking is mostly wrong

Every tutorial: "just chunk everything into 512 tokens with overlap!"

Reality: documents have structure. A research paper's methodology section is different from its conclusion. Financial reports have executive summaries vs detailed tables. When you ignore structure, you get chunks that cut off mid-sentence or combine unrelated concepts.

Had to build hierarchical chunking that preserves document structure:

- Document level (title, authors, date, type)
- Section level (Abstract, Methods, Results)
- Paragraph level (200-400 tokens)
- Sentence level for precision queries

The key insight: query complexity should determine retrieval level. Broad questions stay at paragraph level. Precise stuff like "what was the exact dosage in Table 3?" needs sentence-level precision.

I use simple keyword detection - words like "exact", "specific", "table" trigger precision mode. If confidence is low, system automatically drills down to more precise chunks.

Metadata architecture matters more than your embedding model

This is where I spent 40% of my development time and it had the highest ROI of anything I built.

Most people treat metadata as an afterthought. But enterprise queries are crazy contextual. A pharma researcher asking about "pediatric studies" needs completely different documents than someone asking about "adult populations."

Built domain-specific metadata schemas:

For pharma docs:

- Document type (research paper, regulatory doc, clinical trial)
- Drug classifications
- Patient demographics (pediatric, adult, geriatric)
- Regulatory categories (FDA, EMA)
- Therapeutic areas (cardiology, oncology)

For financial docs:

- Time periods (Q1 2023, FY 2022)
- Financial metrics (revenue, EBITDA)
- Business segments
- Geographic regions

Avoid using LLMs for metadata extraction - they're inconsistent as hell. Simple keyword matching works way better. Query contains "FDA"? Filter for regulatory_category: "FDA". Mentions "pediatric"? Apply patient population filters.

Start with 100-200 core terms per domain, expand based on queries that don't match well. Domain experts are usually happy to help build these lists.

When semantic search fails (spoiler: a lot)

Pure semantic search fails way more than people admit. In specialized domains like pharma and legal, I see 15-20% failure rates, not the 5% everyone assumes.

Main failure modes that drove me crazy:

Acronym confusion: "CAR" means "Chimeric Antigen Receptor" in oncology but "Computer Aided Radiology" in imaging papers. Same embedding, completely different meanings. This was a constant headache.

Precise technical queries: Someone asks "What was the exact dosage in Table 3?" Semantic search finds conceptually similar content but misses the specific table reference.

Cross-reference chains: Documents reference other documents constantly. Drug A study references Drug B interaction data. Semantic search misses these relationship networks completely.

Solution: Built hybrid approaches. Graph layer tracks document relationships during processing. After semantic search, system checks if retrieved docs have related documents with better answers.

For acronyms, I do context-aware expansion using domain-specific acronym databases. For precise queries, keyword triggers switch to rule-based retrieval for specific data points.

Why I went with open source models (Qwen specifically)

Most people assume GPT-4o or o3-mini are always better. But enterprise clients have weird constraints:

- **Cost:** API costs explode with 50K+ documents and thousands of daily queries
- **Data sovereignty:** Pharma and finance can't send sensitive data to external APIs
- **Domain terminology:** General models hallucinate on specialized terms they weren't trained on

Qwen QWQ-32B ended up working surprisingly well after domain-specific fine-tuning:

- 85% cheaper than GPT-4o for high-volume processing
- Everything stays on client infrastructure
- Could fine-tune on medical/financial terminology
- Consistent response times without API rate limits

Fine-tuning approach was straightforward - supervised training with domain Q&A pairs. Created datasets like "What are contraindications for Drug X?" paired with actual FDA guideline answers. Basic supervised fine-tuning worked better than complex stuff like RAFT. Key was having clean training data.

Table processing: the hidden nightmare

Enterprise docs are full of complex tables - financial models, clinical trial data, compliance matrices. Standard RAG either ignores tables or extracts them as unstructured text, losing all the relationships.

Tables contain some of the most critical information. Financial analysts need exact numbers from specific quarters. Researchers need dosage info from clinical tables. If you can't handle tabular data, you're missing half the value.

My approach:

- Treat tables as separate entities with their own processing pipeline
- Use heuristics for table detection (spacing patterns, grid structures)
- For simple tables: convert to CSV. For complex tables: preserve hierarchical relationships in metadata
- Dual embedding strategy: embed both structured data AND semantic description

For the bank project, financial tables were everywhere. Had to track relationships between summary tables and detailed breakdowns too.

Production infrastructure reality check

Tutorials assume unlimited resources and perfect uptime. Production means concurrent users, GPU memory management, consistent response times, uptime guarantees.

Most enterprise clients already had GPU infrastructure sitting around - unused compute or other data science workloads. Made on-premise deployment easier than expected.

Typically deploy 2-3 models:

- Main generation model (Qwen 32B) for complex queries
- Lightweight model for metadata extraction

- Specialized embedding model

Used quantized versions when possible. Qwen QWQ-32B quantized to 4-bit only needed 24GB VRAM but maintained quality. Could run on single RTX 4090, though A100s better for concurrent users.

Biggest challenge isn't model quality - it's preventing resource contention when multiple users hit the system simultaneously. Use semaphores to limit concurrent model calls and proper queue management.

Key lessons that actually matter

1. Document quality detection first: You cannot process all enterprise docs the same way. Build quality assessment before anything else.

2. Metadata > embeddings: Poor metadata means poor retrieval regardless of how good your vectors are. Spend the time on domain-specific schemas.

3. Hybrid retrieval is mandatory: Pure semantic search fails too often in specialized domains. Need rule-based fallbacks and document relationship mapping.

4. Tables are critical: If you can't handle tabular data properly, you're missing huge chunks of enterprise value.

5. Infrastructure determines success: Clients care more about reliability than fancy features. Resource management and uptime matter more than model sophistication.

The real talk

Enterprise RAG is way more engineering than ML. Most failures aren't from bad models - they're from underestimating the document processing challenges, metadata complexity, and production infrastructure needs.

The demand is honestly crazy right now. Every company with substantial document repositories needs these systems, but most have no idea how complex it gets with real-world documents.

Anyway, this stuff is way harder than tutorials make it seem. The edge cases with enterprise documents will make you want to throw your laptop out the window. But when it works, the ROI is pretty impressive - seen teams cut document search from hours to minutes.

Happy to answer questions if anyone's hitting similar walls with their implementations.

Comments

> **Hydeoo** • 27 points •
>
> Amazing thanks for taking the time to write this down
>
> track me
>
> > **Low_Acanthisitta7686** • 11 points •
> >
> > welcome :)
> >
> > > **El_Spanberger** • 1 points •
> > >
> > > Ditto - just starting to look at this in very similar domains. You just saved me months of frustration - thank you.
> > >
> > > > **Low_Acanthisitta7686** • 1 points •

sure!

DAImighty • 13 points •

This is probably one of the best posts that I've seen in a long time. Cheers to you legend 🫰

Low_Acanthisitta7686 • 3 points •

haha, thanks :)

OverratedMusic • 8 points •

Cross-reference chains

- how do you query those relationships or embed them. I understand the importance of getting the relationships, but not on how to retrieve them in a meaningful way

Low_Acanthisitta7686 • 11 points •

Good question - this was actually one of the trickier parts to get right.

For storing the relationships, I keep it separate from the main vector embeddings. During document processing, I build a simple graph where each document is a node and citations/references are edges. Store this in a lightweight graph DB or even just as JSON mappings.

For retrieval, I do it in two phases. First, normal semantic search finds the most relevant documents. Then I check if those documents have relationship connections that might contain better answers. Like if someone asks about "Drug X safety data" and I retrieve a summary paper, I automatically check if that paper cites more detailed safety studies.

The key insight was not trying to embed the relationships directly. Instead, after getting initial results, I expand the search to include connected documents and re-rank based on relevance + relationship strength.

For queries like "find all studies related to the Johnson 2019 paper," I can do direct graph traversal. But for most semantic queries, the relationships work more like a second-pass filter that catches related content the initial search missed.

Implementation-wise, I just track citation patterns during preprocessing using regex to find "Smith et al. 2020" style references, then build the connection graph. Nothing fancy, but it catches a lot of cases where the best answer is in a paper that references the document you initially found.

doubledaylogistics • 2 points •

So once you find the connection to another paper, do you have a way to narrow the vector search for contents from just that paper, for example? Same for how you described the hierarchical sections for each paper...do you have different vector dbs for each level of detail? Or another way of searching at one "level" within a single vector db?

Low_Acanthisitta7686 • 4 points •

Yeah, I use a single vector database but with namespace/filtering approaches rather than separate databases. When I find connections to related papers, I can filter the search to just that specific document using metadata filters. So if the initial search finds "Johnson 2019" and I know it cites "Smith 2020," I can do a targeted search within just the Smith paper using document_id filters.

> > > For the hierarchical sections, I store different chunk levels in the same vector DB but with metadata tags indicating the level - document_level, section_level, paragraph_level. Then I can search within specific levels as needed. Using the metadata filtering capabilities of the vector database works better than trying to manage multiple separate databases. Qdrant handles this pretty well - you can filter by document ID, section type, hierarchical level, etc. before doing the semantic search.
>
> > > So the workflow is: find initial relevant documents → check relationship graph for connected documents → filter vector search to those specific connected documents → re-rank based on relevance + relationship strength. Keeps everything in one search infrastructure but gives you the precision to drill down into specific documents or sections when the relationship graph indicates there might be better answers there. Much simpler than trying to coordinate searches across multiple vector databases.

Flat_Brilliant_6076 • 6 points •

Hey! Thanks for writing this! Do you have usage metrics and feedback from your clients? Are they really empowered with these tools?

> **Low_Acanthisitta7686** • 6 points •
>
> Yeah, the feedback has been pretty solid. Most clients see dramatic improvements in document search time - like going from spending 2-3 hours hunting through folders to finding what they need in minutes. The pharma researchers especially love it because they can quickly find related studies or safety data across thousands of papers. One client told me their regulatory team went from taking days to compile compliance reports to doing it in a few hours.
>
> But honestly, adoption varies a lot. The teams that really embrace it see huge productivity gains. Others still default to their old workflows because change is hard. The key was getting a few power users excited first - they become advocates and help drive broader adoption. ROI is usually pretty clear within a few months. When your analysts aren't spending half their day searching for documents, they can focus on actual analysis. But you definitely need buy-in from leadership and proper training to make it stick.
>
> The biggest surprise was how much people started asking more complex questions once they trusted the system. Instead of just finding specific documents, they'd ask things like "show me all the cardiovascular studies from the last 5 years with adverse events" - stuff that would've been impossible to research manually.
>
> > **redditerfan** • 4 points •
> >
> > dramatic improvements in document search time - like going from spending 2-3 hours hunting through folders to finding what they need in minutes. The pharma researchers especially love it because they can quickly find related studies or safety data across thousands of papers. === Trying to understand, are you creating a database from multiple research papers?
> >
> > > **Low_Acanthisitta7686** • 4 points •
> > >
> > > Yeah, exactly what you're describing. When clients start trusting the system, they ask way more sophisticated questions than they could ever research manually. For the pharma use case - no, I'm not building a unified research database across multiple papers. That would be a massive undertaking and probably need different architecture.

What I built is more like intelligent search across their existing document collections. So when someone asks "show me cardiovascular studies with adverse events from the last 5 years," the system searches through all their research papers, regulatory docs, internal reports, etc. and finds the relevant sections. The key is the metadata tagging I mentioned - during preprocessing, I extract study types, therapeutic areas, timeframes, adverse event mentions, etc. So the query gets filtered by those criteria before semantic search.

It's not creating new knowledge or connecting insights across studies - that's still up to the researchers. But it makes finding the right papers and sections way faster than manual search through thousands of documents. The "database" is really just the vector embeddings plus structured metadata, not a curated research knowledge base.

redditerfan • 3 points •

Thanks for explaining - you mentioned you will opensource your project, is there any similar opensourced project I can use? I am in biotech we need to process a lot of patents and research paper to search for molecules and their activity tables. I am trying to come up with a workflow

KasperKazzual • 5 points •

Awesome info, thanks for sharing. How would you approach building a “consultancy” agent that is trained purely with like 5k pages of cleanly formatted PDF docs (no tables just a bunch of reports, guides, books, frameworks, strategic advice etc.) and doesnt need to answer specific questions about those docs but instead use those docs to give strategic advice that aligns with the training data or reviews attached reports based on the training data?

Low_Acanthisitta7686 • 6 points •

Interesting use case - sounds more like knowledge synthesis than traditional retrieval.

For this kind of strategic advisory system, I'd actually approach it differently than standard RAG. Since you want the model to internalize the frameworks and give advice rather than cite specific documents, fine-tuning might work better than retrieval.

Here's what I'd try: extract the key frameworks, methodologies, and strategic principles from those 5k pages during preprocessing. Create training examples that show how those frameworks apply to different business scenarios. Then fine-tune a model on question-answer pairs where the answers demonstrate applying those strategic concepts. For example, if your docs contain McKinsey-style frameworks, create training data like "How should we approach market entry?" with answers that naturally incorporate the frameworks from your knowledge base without explicitly citing them.

The retrieval component becomes more about finding relevant strategic patterns rather than exact text matches. When someone asks for advice on market positioning, you pull examples of similar strategic situations from your docs, then use those as context for generating advice that feels like it comes from an experienced consultant who's internalized all that knowledge.

I'd still keep some light retrieval for cases where they want to review reports against your frameworks - but the main value is having a model that thinks strategically using the patterns from your knowledge base. Much more complex than standard RAG but way more interesting. The model needs to understand strategic thinking patterns, not just retrieve information.

im_vivek • 4 points •

thanks for writing such a detailed post

can you provide more details on metadata enrichment techniques

Low_Acanthisitta7686 • 11 points •

For metadata enrichment, I kept it pretty simple. Built domain-specific keyword libraries - like 300 drug names for pharma, 200 financial metrics for banking clients. Just used regex and exact matching, no fancy NLP. The main thing was getting domain experts to help build these lists. They know how people actually search - like in pharma, researchers might ask about "adverse events" or "side effects" or "treatment-emergent AEs" but they're all the same thing. So I mapped all variants to the same metadata tags.

For document type classification, I looked for structural patterns. Research papers have "Abstract" and "Methods" sections, regulatory docs have agency letterheads, financial reports have standard headers. Simple heuristics worked better than trying to train classifiers. Also tracked document relationships during processing - which papers cite others, which summary reports reference detailed data. Built a simple graph of these connections that helped with retrieval later.

I wasted weeks trying to use LLMs for metadata extraction before realizing keyword matching was more consistent and faster. Having good domain keyword lists mattered more for sure!

mcs2243 • 3 points •

Are you just pulling in the metadata every time along with the corpus of RAG documents?

Low_Acanthisitta7686 • 7 points •

Yeah, exactly - I pull the metadata during initial document processing and store it alongside the embeddings. During preprocessing, I extract all the metadata (document type, keywords, dates, etc.) and store it in the vector database as structured fields. Then at query time, I can filter on those metadata fields before doing the semantic search.

So if someone asks about "FDA pediatric studies," I first filter for documents where regulatory_category="FDA" AND patient_population="pediatric", then do semantic search within that filtered subset. Way more accurate than trying to rely on embeddings to capture those specific attributes. The metadata gets stored once during ingestion, not pulled fresh every time. Much faster and more consistent that way.

exaknight21 • 4 points •

How did you deal with such large amounts of data vs. context window?

Low_Acanthisitta7686 • 5 points •

Context window management was definitely one of the trickier parts to get right.

For the embedding side, I don't try to fit everything into context at once. During preprocessing, I chunk documents into 500-token pieces and embed those separately. The retrieval step pulls the most relevant 5-10 chunks, not entire documents.

The real challenge is when you need broader context for generation. What I do is hierarchical retrieval - start with paragraph-level chunks, but if the query seems like it needs more context, I grab the parent sections or even the full document structure. For really large documents, I use a two-pass approach. First pass finds the relevant sections, second pass expands context around those sections if needed. Most queries don't actually need the full 50-page document - they need specific sections with enough surrounding context to make sense.

The key insight was not trying to stuff everything into one context window. Instead, I built smart retrieval that pulls just enough context for each specific query type. Document summarization helps too - I keep high-level summaries that can fit in context along with the detailed chunks.

Memory management gets tricky with multiple concurrent users though. Had to implement proper caching and lazy loading so the system doesn't crash when everyone hits it at once.

Agreeable_Company372 • 1 points •

embeddings and chunking?

> **exaknight21** • 2 points •
>
> The context received back is loaded into the context window to give a comprehensive answer. Self hosting even a smaller model means scaling for the extra VRAM as well for multiple users no matter VLLM or llama.cpp.
>
> Embedding models aren’t an issue because we do context chaining and mainly feed it 500 chunks or whatever at a time - typical window there is around 8000. Generating requires context to be in the window if it’s multiple documents so unless pre-retrieval filters are narrowing it down to select few documents, context window can be problematic. The same window, iirc, is used to do generation so it could pose an issue - which is the part i am asking the question for.

mcs2243 • 4 points •

What was your chunking process like? I’m using cognee right now for memory/document chunking and graph creation. Wondering if you found other OSS better. Thanks!

> **Low_Acanthisitta7686** • 3 points •
>
> Haven't used cognee specifically, but for chunking I kept it pretty straightforward. Built my own pipeline because most OSS tools didn't handle the hierarchical approach I needed. My process: first detect document structure (headers, sections, tables), then chunk at different levels - document level metadata, section-level chunks (usually 300-500 tokens), paragraph-level for detailed retrieval. The key was preserving parent-child relationships between chunk levels.
>
> For graph creation, I used a simple approach - just tracked document citations and cross-references during preprocessing, stored as JSON mappings. Nothing fancy like Neo4j, just lightweight relationship tracking. Most OSS chunking tools assume you want fixed-size chunks, but enterprise docs need structure-aware chunking. Research papers have abstracts, methods, results - each section should be chunked differently than a wall of text.
>
> I looked at LangChain's document loaders but ended up building custom parsers for each document type. More work upfront but way more control over how different document types get processed.

> **Puzzleheaded_Fold466** • 2 points •
>
> Fantastic lessons learned summary. Thanks for sharing this.

> **Hot-Independence-197** • 2 points •
>
> Thanks a lot for sharing your experience. I’m currently studying RAG systems and I’m still at the very beginning of my journey. In the future, I would really like to help companies implement these solutions and provide integration services just like you do. Do you have any tips or advice for a beginner who’s genuinely interested in this field? Maybe you can recommend good tutorials, courses, or resources to really learn both the engineering and practical sides? Any suggestions would be greatly appreciated!

Low_Acanthisitta7686 • 11 points •

The best way to learn this stuff is by building with messy, real documents - not the clean tutorial datasets.

Pick a specific domain like real estate and understand their pain points. If you don't know anyone in that space, you can use ChatGPT to understand their workflow challenges, then try building something that actually solves those problems. Real estate has complex property docs, floor plans, images, legal paperwork - way messier than basic chatbot tutorials.

Start simple but increase complexity as you go. Don't get stuck building basic Q&A chatbots - that's been solved for years. Focus on the hard stuff like processing property images, extracting data from scanned contracts, handling regulatory documents with weird formatting.

The build-and-break approach works best. Grab real estate PDFs, try processing them, watch your system fail on edge cases, then figure out how to handle those failures. That's where you actually learn the engineering challenges. Skip most online courses - they're too focused on the happy path. Learn by doing: spin up Qdrant locally, try different chunking strategies on the same documents, see how results change.

BTW don't be afraid to get creative with simple solutions that actually work.

> **Hot-Independence-197** • 2 points •
>
> Thank you so much for this advice! Your comment is truly insightful and motivating. I really appreciate you sharing your experience and practical tips.
>
> > **Low_Acanthisitta7686** • 2 points •
> >
> > sure :)
> >
> > > **Hot-Independence-197** • 1 points •
> > >
> > > I wonder, wouldn’t it be simpler for companies to use something like NotebookLlama instead of building a full RAG system from scratch? It already supports document ingestion, search, and text/audio generation out of the box. Or is there something about the internal knowledge base and enterprise requirements that I might be missing? Would love to hear thoughts on when it’s better to build a custom RAG pipeline versus adopting open-source solutions like NotebookLlama

Skiata • 2 points •

Noice....thanks for making the effort.

> **Low_Acanthisitta7686** • 1 points •
>
> 💥

Independent_Paint752 • 2 points •

This is gold. thanks.

> **Low_Acanthisitta7686** • 1 points •
>
> welcome :)

Operator_Remote_Nyx • 2 points •

Saved and coming back thank you!

marceloag • 2 points •

Hey OP, thanks for sharing this! Totally agree with you on the points you made, especially the struggles of making RAG actually work in real enterprise cases.

Any chance you could share how you handled document quality detection? I'm running into that problem myself right now.

Appreciate you putting this out there!

> **Low_Acanthisitta7686** • 5 points •
>
> Thanks! Document quality detection was honestly a lifesaver once I figured it out.
>
> I built a simple scoring system that checks a few key things: text extraction success rate (how much actual text vs garbled characters), OCR artifact detection (looking for patterns like "rn" instead of "m"), structural consistency (proper paragraphs vs wall of text), and formatting cues (headers, bullet points, etc.).
>
> For scoring, I sample random sections from each document and run basic text quality checks. Documents with clean text extraction, proper spacing, and recognizable structure get high scores. Scanned docs with OCR errors or completely unstructured text get flagged for simpler processing.
>
> The key insight was routing documents to different pipelines based on score rather than trying to make one pipeline handle everything. High-quality docs get full hierarchical processing, medium quality gets basic chunking with cleanup, low quality gets simple fixed-size chunks plus manual review flags. Pretty straightforward stuff but made a huge difference in consistency. Way better than trying to debug why some documents returned garbage while others worked perfectly.
>
> What specific quality issues are you running into? Might be able to suggest some targeted checks.

smirk79 • 2 points •

Excellent post. Thanks for the write up! What do you charge and why this vs single company work?

> $deleted$ • 5 points •
>
> This is gold! The amount of work going into getting the pipeline right is insane and I agree that the demos and tutorials online barely scratch the surface. Would be really Interesting to see your approach whenever, or rather, if you decide to open-source it.
>
> > **Cruiserrider04** • 2 points •
> >
> > This is gold! The amount of work going into getting the pipeline right is insane and I agree that the demos and tutorials online barely scratch the surface. Would be really Interesting to see your approach whenever, or rather, if you decide to open-source it.

Code_0451 • 2 points •

As a BA in banking I'm not surprised, this is a huge challenge to any automation attempt. Problem is that a lot of people at executive level or in tech are *NOT* aware of this and hugely underestimate the time and effort.

Also your example is still relatively "easy". Company size and number of documents are not particularly high and they probably were still fairly centralized in one location. Wait till you see a large org with fragmented data repositories…

tibnine • 2 points •

Easily the best write-up on this. Thank you!

Few Qs; how do you evaluate the e2e system? More specifically how do you set a performance bar with your clients and avoid anecdotal one off assessments.

Related, how do you know when's enough fine tuning for your models? Are there general benchmarks (beyond the ones you construct for the specific use-case) you try to maintain performance over while you fine tune?

Once again, you rock 🤘

> **Low_Acanthisitta7686** • 5 points •
>
> Thanks! Evaluation was honestly one of the hardest parts to get right.
>
> For setting performance bars, I work with domain experts to create golden question sets - like 100-200 questions they'd actually ask, with known correct answers. We agree upfront on what "good enough" looks like - usually 85%+ accuracy on these test questions. The key is making the evaluation questions realistic. Not "What is Drug X?" but "What were the cardiovascular safety signals in the Phase III trials for Drug X in elderly patients?" - the complex stuff they actually need to find.
>
> For ongoing evaluation, I track retrieval accuracy (did we find the right documents?) and answer quality (did the model give useful responses?). Simple thumbs up/down from users works better than complex scoring systems. For fine-tuning, I stop when performance plateaus on the validation set and users stop complaining about domain-specific terminology issues. Usually takes 2-3 iterations. I don't worry much about general benchmarks - if the model handles "myocardial infarction" correctly in context, that matters more than MMLU scores.
>
> The real test is when domain experts start trusting the system enough to use it for actual work instead of just demos.

SadCod2634 • 2 points •

Amazing work

> **Low_Acanthisitta7686** • 1 points •
>
> thanks!!!

VinnyChuChu • 2 points •

fantastic stuff, saving this

WelcomeMysterious122 • 2 points •

nice

hiepxanh • 2 points •

How do you process excel with multi sheet? Convert to xml and cache? How about slide too?

> **Low_Acanthisitta7686** • 3 points •
>
> For Excel files, I extract each sheet separately and treat them as individual documents with metadata linking them back to the parent workbook. Don't convert to XML - just use pandas to read each sheet, preserve the structure, and create embeddings for both the raw data and a semantic description of what each sheet contains.
>
> For multi-sheet relationships, I track which sheets reference others (like summary sheets pulling from detail sheets) and store those connections in the document graph I mentioned earlier.

Slides are trickier. I extract text content obviously, but also try to preserve the slide structure - title, bullet points, speaker notes. For charts and images, I generate text descriptions of what they show. Each slide becomes its own chunk with slide number and presentation metadata.

The key insight was treating complex documents as collections of related sub-documents rather than trying to flatten everything into one big text blob. An Excel workbook might have 15 sheets that each answer different questions, so I need to be able to retrieve from the right sheet.

For both file types, I keep the original structure info in metadata so I can return answers like "this data is from Sheet 3 of the Q4 Financial Model" rather than just giving raw numbers without context.

Not perfect but works way better than trying to convert everything to plain text.

> **hiepxanh** • 1 points •
>
> You are corrected, I think that is the best solution we can appoarch, only last question, how you can find your job? Or you are provide custom service for company? Appoarch company need that solution so hard

Johnintheuk99 • 2 points •

Brilliant post thanks

> **Low_Acanthisitta7686** • 1 points •
>
> :)

MyReviewOfTheSun • 2 points •

Were you ever asked to include low signal-to-noise ratio data sources like chat logs or transcripts? how did you handle that?

> **Low_Acanthisitta7686** • 3 points •
>
> Yeah, ran into this with a few clients. One bank wanted years of internal Slack messages and meeting recordings included. Massive pain because 90% was just noise. For Slack, I filtered aggressively - dropped anything under 20 words, removed obvious social stuff, focused on threads with business keywords. Even then, results were pretty underwhelming. People write differently in chat than formal docs.
>
> Meeting transcripts were worse. Built simple extraction for decision points and action items, but transcripts are full of incomplete thoughts and missing context. Someone saying "yeah, let's go with option B" means nothing without knowing what the options were. Tried creating quality tiers - formal decisions got full processing, casual discussions got basic treatment, pure chatter got filtered out. But users still complained about getting irrelevant chat snippets mixed in with real documents.
>
> Honestly told that client it probably wasn't worth the effort. The signal-to-noise ratio was terrible compared to their formal reports and documentation. Unless they had specific cases where informal communications were critical, I'd skip it. Most clients are better off focusing on structured documents first, then maybe experimenting with chat data later if they really need it.

_Passi • 2 points •

Really nice post. Thank you!

Did you face sone cases in which the client documents were pretty large e.g. 2500 pages. How did you handle these kind of documents?

Low_Acanthisitta7686 • 2 points •

Yeah, definitely dealt with massive documents. Had pharma clients with clinical study reports that were 2000+ pages, financial clients with comprehensive audit reports around that size too. For documents that large, I break them into logical sections first rather than just chunking blindly. Clinical reports have standard sections - protocol, patient demographics, efficacy results, safety data, appendices. Each section gets processed separately with its own metadata.

The key insight was treating them like collections of related documents rather than one giant document. A 2500-page clinical report might have 15-20 distinct sections that each answer different types of questions. I also implement lazy loading - don't try to process the entire document at once. Process the table of contents and section headers first, then drill down into specific sections as needed. This prevents memory issues and makes the system more responsive.

For retrieval, I use the hierarchical approach I mentioned - start with section-level search, then narrow down to specific pages or subsections. Users can ask broad questions and get section-level answers, or specific questions that drill down to exact page references. Storage-wise, I chunk each section separately but maintain the document structure in metadata. So I can return answers like "this finding is from the Safety Analysis section, pages 450-475 of the Clinical Study Report." The processing time is longer obviously, but the results are way better than trying to flatten everything into uniform chunks.

Caden_Voss • 2 points •

Amazing post, best on the sub so far. Can you share some good tutorials on RAG that you recommend?

Low_Acanthisitta7686 • 1 points •

never learnt through tutorials, but check this - https://www.reddit.com/r/LLMDevs/comments/1n98lsf/comment/nclwnbn/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button

visarga • 2 points •

Great presentation. But I am wondering about iterated search, you did not mention it. I find that even a limited search tool, such as embedding based RAG, can be great if the model retargets the search a few times before answering. Read, think, read some more, think more... In the last 6-12 months models have become good at operating tools like this. RAG+search orchestration beats one step RAG.

Can confirm the metadata generation step prior to embedding, I have been doing this for a year.

Low_Acanthisitta7686 • 2 points •

You're absolutely right about iterative search - that's definitely something I should have covered more. I do use it but maybe didn't emphasize it enough. For complex queries, I often do multiple retrieval passes. First pass gets initial results, then the model analyzes what's missing and does targeted follow-up searches. Like if someone asks about "Drug X cardiovascular safety in elderly patients," first search might find general safety data, then the model realizes it needs more specific elderly population data and does a second search with refined terms.

I think it's important to let the model drive the search refinement rather than trying to get everything in one shot. Models have gotten way better at this kind of tool orchestration like you mentioned. I implement it as a simple loop - retrieve, analyze results, decide if more context is needed, search again with refined queries. Usually 2-3 iterations max before generating the final answer. Works especially well for research questions where you need to piece together information from multiple sources.

> The metadata approach definitely helps here too - the model can see what types of documents it found and specifically search for missing document types or time periods. You're right that RAG + search orchestration beats single-step retrieval, especially for complex enterprise queries. The iterative approach catches a lot of cases where the initial search term interpretation wasn't quite right.

Wild_Ad74 • 2 points •

this is awesome! thank you for sharing this, Sir.

> **Low_Acanthisitta7686** • 1 points •
>
> for sure!

Mindless_Copy_7487 • 2 points •

Very interesting insights, thank you. What was your approach towards clean PDFs with a high layout complexity (multi-column, tiles, boxes etc)?

> **Low_Acanthisitta7686** • 1 points •
>
> Complex layout PDFs are honestly where most traditional parsing libraries completely fall apart. Multi-column documents especially - they'll read left column, then right column, then jump back to left column from the next page. Complete mess. For high layout complexity, I ended up using a hybrid approach. First, I try layout-aware parsing with libraries like pdfplumber or pymupdf, but set my expectations low. They work maybe 60% of the time for complex layouts.
>
> When that fails, I convert PDF pages to images and use VLMs to understand the layout structure. The vision models can actually see that there are two columns, or that there's a sidebar with different content, or that there are text boxes with specific information. For really structured documents like financial reports or research papers with consistent formatting, I sometimes build custom extractors that look for specific layout patterns. Like if I know section headers are always in bold at the left margin, I can extract those reliably.
>
> The multi-column problem is particularly brutal because reading order matters. VLMs help here since you can ask them to "read this document in the correct order" rather than trying to guess the flow programmatically. For documents with lots of figures, charts, and text boxes mixed together, I often just accept that some content will be extracted out of order and handle that in post-processing by looking for logical groupings.
>
> The reality is there's no universal solution. Each document type needs its own approach based on how consistent the layout patterns are.
>
> > **Mindless_Copy_7487** • 1 points •
> >
> > That's also our experience so far and we ended up basically use the same hybrid approach (libs + VLM). Thank you for your insights. PDF is a challenge but thousands of PDF with tables, complex layouts is really tough.

substralhofer • 2 points •

Thanks to OP - so much for this piece of art, and for all of your effort answering so many questions so well.

> **Low_Acanthisitta7686** • 1 points •
>
> welcome :)))

CodGreedy1889 • 2 points •

Thanks for sharing! What technological stack did you use? Libraries, frameworks, programming languages, DBs etc.?

> **Low_Acanthisitta7686** • 2 points •
>
> actually depends on each project, but usually my stack is python, ollama, vllm, react/nextjs, qdrant, nomic embeddings, pymupdf, tesseract, postgres

Repulsive_Panic4 • 2 points •

Very cool. Not everybody is willing to share the details :)

> **Low_Acanthisitta7686** • 1 points •
>
> thanks, just trying to be helpful!

Captain_BigNips • 1 points •

Fantastic write up. I have been dealing with similar issues myself and have found that the document processing part is the most frustrating aspect of the whole thing.... I really hope a tool comes along soon to help sort this kind of stuff out.

Shap3rz • 1 points •

This was great - thanks. I've done data ingestion for rag before but was only personally responsible for some over excel and csv files so didn't have such a metadata conundrum. My instinct was hybrid with financial docs and already have a similar schema. But hadn't considered having such a large keyword lexicon. Thank you!

> **Low_Acanthisitta7686** • 1 points •
>
> Nice! Yeah, I initially tried to be clever with NLP-based metadata extraction but simple keyword matching just worked better and was way more predictable. Financial docs especially benefit from this approach since the terminology is pretty standardized. Having those 200+ financial metrics mapped out made queries so much more accurate than relying purely on semantic search. The hybrid approach definitely pays off - semantic search for broad conceptual stuff, keyword filtering for precise financial terms and time periods. Sounds like you're on the right track with your schema setup.
>
> > **Shap3rz** • 1 points •
> >
> > When you say match a keyword for a metric, what sort of logic did you have to ensure you're capturing the right value from the right place if you don't mind my picking your brain one last time? At the moment I'm only grabbing a few so LLM based is working ok at this early stage (I'm iterating on someones draft solution that is context stuffing) but I'm conscious it won't be that reliable. I do have spaCy in the env too.

mailaai • 1 points •

In future we have embedding model that can understand the context much better. It realized such model needs resource to run it. I had experiment with very large model's embedding, and the way understood the context was surprising to me. It took one minutes to generate embedding 24k dim.

ObjectiveAd7906 • 1 points •

Thanks man, simply fantastic, your summary and the logic developed

itfeelslikealot • 1 points •

Brilliant write-up thank you u/Low_Acanthisitta7686. Hey are you getting feedback yet from your customers with finished implementations about ROI and other feedback? I'm super curious about how your customers articulate what ROI looks like to them.

Also if you are comfortable revealing any follow-up stories about measures you needed to take after implementation was in production for say a month?

> **Low_Acanthisitta7686** • 2 points •
>
> Thanks! ROI conversations are interesting because it varies a lot by client. The pharma company tracks it pretty precisely - their researchers were billing 25-30 hours per week just on literature searches, now it's down to 8-10 hours. At their billing rates, that paid for the entire system in like 4 months.
>
> Banking client measures it differently - they can now do competitor analysis that used to take weeks in a couple days. Hard to put exact numbers on strategic work, but they're definitely seeing value. Post-production, the main thing was getting people comfortable with the new workflow. Takes time for teams to trust the system enough to change how they work. Found that having a few power users demo results to their colleagues helped adoption way more than formal training sessions.
>
> Also learned to set expectations better around edge cases. Users would find the one document type that broke the system and get frustrated. Now I'm more upfront about what works well and what needs manual review. Document versioning was trickier than expected - clients kept uploading new versions without removing old ones, so users got conflicting results. Had to build better update workflows to handle that.
>
> The technical stuff mostly worked as expected. The workflow integration piece needed more attention than I initially planned for.

BeginningReflection4 • 1 points •

Thanks for this. Can you point to the best tutorial you have worked with/seen? I am going to try and build an MVP for a law firm and now you have me second guessing it so some additional guidance would be great.

> *deleted* • 0 points •

haloweenek • 1 points •

Nice. Thanx !

Can you share some docs how to start doing this stuff ? I'd love to peek into

> *deleted* • 0 points •

van0ss910 • 1 points •

If you think about it, just 1 year ago probably you would use US-based Llama, but now Chinese models are handling some of the most critical documents for the US economy. Not giving judgment, but observing the fact. Crazy times

UPD: not sure why I assumed it's US based clients, but keeping the comment

> **Low_Acanthisitta7686** • 2 points •
>
> Yeah, crazy times for sure. Llama did not cut it for enterprise - peak hallucination, improper tool calling, and generally the model intelligence was so low. Qwen is a game changer, really did a good job on reasoning and stable function calling, overall was super reliable. Yes, most of them were US companies and some from the APAC region as well.

Demlo • 1 points •

Curious why you didn't go with Gemma, I heard googles open weights models are excellent

> **Low_Acanthisitta7686** • 2 points •
>
> Had success with Qwen and now with OSS models. Initial POC with Gemma did not work as expected, so it did not give it a second look!

dibu28 • 1 points •

Why not gpt-oss-20B instead of Qwen? Have you tried ColbertV2 for embeddings?

> **Low_Acanthisitta7686** • 2 points •
>
> Yes, currently I am deploying both 20B and 120B variants. Should mention the 20B is super stable and reasoning is top-notch even being a low parameter model. I use Nomic for embeddings.

deleted • 0 points •

> *deleted* • 0 points •

Oregon_Oregano • 1 points •

More of a business question, how did you get started? Is it just you or do You work on a team?

> **Low_Acanthisitta7686** • 2 points •
>
> Single wizard + Claude Code! I mean I usually work with a small team when needed, but most of the RAG and agents work is something I handle myself. Features, UI and other work I would let someone else work on.
>
> > **Oregon_Oregano** • 1 points •
> >
> > Makes sense, that's awesome! How did you find clients?
> >
> > > **Low_Acanthisitta7686** • 1 points •
> > >
> > > Initially through Upwork, but now mostly through personal networks. I also collaborate with other engineers, agencies, and partners.
> > >
> > > > **Oregon_Oregano** • 1 points •
> > > >
> > > > That's awesome, congrats

thebeardedjamaican • 1 points •

Thanks for sharing

xvmakh • 1 points •

How long did each of these projects take for you?

> **Low_Acanthisitta7686** • 1 points •
>
> 3+ months

vikasgoddubarla • 1 points •

I also worked on the RAG system, actually it is very hard processing the data cleaning the data

I use GPT 4o for that actually what I did you know I used PGVECTOR to store the data along with QDRANT.. I used 512 tokens for each chunk so when we are uploading the document I extract and split the text again give to the AI to clean the text with formatting (May this will help you a lot with qwen try it).

So I give a prompt like “make this text ready to embed the data” I worked on not only documents but also sitemaps and website links.

Trust me it was worked very well.

Low_Acanthisitta7686 • 1 points •

That's an interesting approach with the LLM text cleaning step. Using GPT-4o to preprocess and format text before embedding could definitely help with consistency, especially for messy documents. The "make this text ready to embed" prompt is clever - lets the model normalize formatting, fix OCR artifacts, and standardize terminology before vectorization. That probably improves embedding quality quite a bit compared to raw extracted text.

Running both PGVector and Qdrant seems like overkill though - any particular reason for the dual vector storage? Most use cases work fine with just one. For website and sitemap processing, that preprocessing step probably helps a lot since web content has so much formatting noise, navigation elements, ads, etc. Getting clean, embedable text from HTML is always a pain.

512 tokens per chunk is pretty standard. Did you experiment with different chunk sizes or stick with that throughout? The text cleaning approach could work well with Qwen too, though might be slower than GPT-4o. Worth testing if the quality difference justifies the speed/cost tradeoff.

BTW How much does the preprocessing step slow down your document ingestion pipeline? That's usually the main downside of adding LLM calls to the preprocessing flow.

vikasgoddubarla • 1 points •

Sorry, I mentioned it was two, I’ve worked with different vector databases depending on the project size. For smaller projects with fewer documents, I used PgVector. For larger ones, like 500GB to 2TB, I used Qdrant. For very large datasets, around 10TB–12TB, I went with Pinecone.

I experimented with different embedding sizes like 1536 and 2000, but found that 512 tokens works best, even with free embedding models. With a 50–70 token overlap, it still answers accurately.

I also stored metadata inside the chunks, so when querying, the system prioritizes the latest data even when using similarity search.

For news sites, the challenge was handling large sitemaps (around 20,000 pages) that needed to be scraped, embedded, and processed in the background. A big issue was the noise from things like sidebars, navigation bars, and featured posts. BeautifulSoup helped remove about 80% of that noise, but some still slipped through. To fix this, I used the model during chunking to clean out the rest, which worked really well.

jannemansonh • 1 points •

Document quality and metadata really are where most enterprise RAG projects break. With Needle we handle those as separate tools, that the agent can call...

davispuh • 1 points •

Typically deploy 2-3 models:

> Main generation model (Qwen 32B) for complex queries
>
> Lightweight model for metadata extraction
>
> Specialized embedding model

which models you used for these?

Also which open source libraries you used ?

heronlon • 1 points •

Thank you so much for your post, you've opened my eyes for couple new things as I've started to look at a topic you've described recently.

I have a question for you, maybe you'd have some experience or intuition that you'd be willing to share. Beside the project documentation I would like to add to RAG also transcripts from long discussions (couple of hours per one meeting, 10+ participants) that we gathered across couple of last months. The challenge is to chunk those transcripts in a smart way.

Do you have any idea what would be the best approach to go with in that case?

Apart-Touch9277 • 1 points •

100% agree, data quality and remediation has been 80% of the effort in data analysis and data science forever. Same rules apply here

> **Low_Acanthisitta7686** • 1 points •
>
> true!

Barry_Jumps • 1 points •

Excellent writeup. Thank you! The real question is how do we contact you for an engagement?

> **Low_Acanthisitta7686** • 1 points •
>
> thanks, send me a dm!

Expert-General-4765 • 1 points •

This was an excellent post — the information you shared was very helpful and made me reconsider a few of my approaches, thank you. At the moment, I'm working on a large-scale OCR project and facing a few challenges. I'm struggling to decide whether I should go with a cloud-based OCR solution or an open-source one. On top of that, I'm also working on designing my chunking and metadata strategy. For chunking, I'm considering implementing semantic chunking. How difficult is it to implement this in practice? Have you had any prior experience with it? Your insights would be extremely valuable for me. Thanks in advance!

New_Ratio_8058 • 1 points •

can u share which tools have you been using for the RAG pipeline, from collecting data to parsing, cleaning, embedding, vector database...
for such a huge amount of documents from different resources, how do you pre-process it? manually with team effort OR figure out some patterns first, then do some sample and give the rest to an LLM?
If you can make a detail case study with everything you have done, from the begining to the end, that would be precious

Wrong-Resolution4838 • 1 points •

> This stuff is way harder than tutorials make it seem.
>
> I wanna put this on a t-shirt and show it to every executive.
>
> > **Low_Acanthisitta7686** • 1 points •
> >
> > haha... nice 😂 💯

> **FalseDescription5054** • 1 points •
>
> Wow man totally agree with you and funny enough I also ended up with qwen as favorite llm.
>
> People sleep on the quality of the metadata for each document extraction. I was also using different extraction methods and end up via script with OCR tools but 0 AI involved. AI only for really bad documentation that maybe needing reworking.
>
> However I found rag and vector database was not good enough for me for semantics search. Well it depends better hybrid with knowledge graph.
>
> I combine it with knowledge graph to increase the RAG from 70% to 85% / 90% accuracy. Connected to qwen I got amazing results

> **jannemansonh** • 1 points •
>
> Totally resonates... messy docs + metadata are where RAG breaks.
> At Needle we make these first-class: files go through automated indexing, quality-aware chunking, and domain-specific connectors.
> Hybrid retrieval (semantic + filters) is built-in, with references so results are trustworthy.
> We've seen the same ROI gains once teams stop treating metadata as an afterthought.
> Love this post! Thanks for sharing the real-world pain points....

> **PykeAbuser** • 1 points •
>
> Have you tried fact / table extraction into DBs and allowing the models to query as a tool?

> **rishiarora** • 2 points •
>
> How much do you charge ?
>
> > **Low_Acanthisitta7686** • 1 points •
> >
> > starting is around 100k