Hey everyone, I'm Raj. Over the past year I've built RAG systems for 10+ enterprise clients – pharma companies, banks, law firms – handling everything from 20K+ document repositories, deploying air-gapped on-prem models, complex compliance requirements, and more.

In this post, I want to share the actual learning path I followed – what worked, what didn't, and the skills you really need if you want to go from toy demos to production-ready systems. Even if you're a beginner just starting out, or an engineer aiming to build enterprise-level RAG and AI agents, this post should support you in some way. I'll cover the fundamentals I started with, the messy real-world challenges, how I learned from codebases, and the realities of working with enterprise clients.

I recently shared a technical post on building RAG agents at scale and also a business breakdown on how to find and work with enterprise clients, and the response was overwhelming – thank you. But most importantly, many people wanted to know *how I actually learned these concepts*. So I thought I'd share some of the insights and approaches that worked for me.

The Reality of Production Work

Building a simple chatbot on top of a vector DB is easy — but that's not what companies are paying for. The real value comes from building RAG systems that work at scale and survive the messy realities of production. That's why companies pay serious money for working systems — because so few people can actually deliver them.

Why RAG Isn't Going Anywhere

Before I get into it, I just want to share why RAG is so important and why its need is only going to keep growing. RAG isn't hype. It solves problems that won't vanish:

- Context limits: Even 200K-token models choke after ~100–200 pages. Enterprise repositories are 1,000x bigger. And usable context is really ~120K before quality drops off.
- Fine-tuning ≠ knowledge injection: It changes style, not content. You can teach terminology (like "MI" = myocardial infarction) but you can't shove in 50K docs without catastrophic forgetting.
- Enterprise reality: Metadata, quality checks, hybrid retrieval – these aren't solved. That's why RAG engineers are in demand.
- The future: Data grows faster than context, reliable knowledge injection doesn't exist yet, and enterprises need audit trails + real-time compliance. RAG isn't going away.

Foundation

Before I knew what I was doing, I jumped into code too fast and wasted weeks. If I could restart, I'd begin with fundamentals. Andrew Ng's deeplearning ai courses on RAG and agents are a goldmine. Free, clear, and packed with insights that shortcut months of wasted time. Don't skip them – you need a solid base in embeddings, LLMs, prompting, and the overall tool landscape.

Recommended courses:

- Retrieval Augmented Generation (RAG)
- LLMs as Operating Systems: Agent Memory
- Long-Term Agentic Memory with LangGraph
- How Transformer LLMs Work
- Building Agentic RAG with LlamaIndex
- Knowledge Graphs for RAG

- Building Apps with Vector Databases

I also found the *AI Engineer* YouTube channel surprisingly helpful. Most of their content is intro-level, but the conference talks helped me see how these systems break down in practice. **First build:** Don't overthink it. Use LangChain or LlamaIndex to set up a Q&A system with clean docs (Wikipedia, research papers). The point isn't to impress anyone – it's to get comfortable with the retrieval → generation flow end-to-end.

Core tech stack I started with:

- Vector DBs (Qdrant locally, Pinecone in the cloud)
- Embedding models (OpenAI → Nomic)
- Chunking (fixed, semantic, hierarchical)
- Prompt engineering basics

What worked for me was building the same project across multiple frameworks. At first it felt repetitive, but that comparison gave me intuition for tradeoffs you don't see in docs.

Project ideas: A recipe assistant, API doc helper, or personal research bot. Pick something you'll actually use yourself. When I built a bot to query my own reading list, I suddenly cared much more about fixing its mistakes.

Real-World Complexity

Here's where things get messy – and where you'll learn the most. At this point I didn't have a strong network. To practice, I used ChatGPT and Claude to roleplay different companies and domains. It's not perfect, but simulating real-world problems gave me enough confidence to approach actual clients later. What you'll quickly notice is that the easy wins vanish. Edge cases, broken PDFs, inconsistent formats – they eat your time, and there's no Stack Overflow post waiting with the answer.

Key skills that made a difference for me:

- Document Quality Detection: Spotting OCR glitches, missing text, structural inconsistencies. This is where "garbage in, garbage out" is most obvious.
- Advanced Chunking: Preserving hierarchy and adapting chunking to query type. Fixed-size chunks alone won't cut it.
- Metadata Architecture: Schemas for classification, temporal tagging, cross-references. This alone ate ~40% of my dev time.

One client had half their repository duplicated with tiny format changes. Fixing that felt like pure grunt work, but it taught me lessons about data pipelines no tutorial ever could.

Learn from Real Codebases

One of the fastest ways I leveled up: cloning open-source agent/RAG repos and tearing them apart. Instead of staring blankly at thousands of lines of code, I used Cursor and Claude Code to generate diagrams, trace workflows, and explain design choices. Suddenly gnarly repos became approachable.

For example, when I studied OpenDevin and Cline (two coding agent projects), I saw two totally different philosophies of handling memory and orchestration. Neither was "right," but seeing those tradeoffs taught me more than any course.

My advice: don't just read the code. Break it, modify it, rebuild it. That's how you internalize patterns. It felt like an unofficial apprenticeship, except my mentors were GitHub repos.

When Projects Get Real

Building RAG systems isn't just about retrieval — that's only the starting point. There's absolutely more to it once you enter production. Everything up to here is enough to put you ahead of most people. But once you start tackling real client projects, the game changes. I'm not giving you a tutorial here – it's too big a topic – but I want you to be aware of the challenges you'll face so you're not blindsided. If you want the deep dive on solving these kinds of enterprise-scale issues, I've posted a full technical guide in the comments — worth checking if you're serious about going beyond the basics.

Here are the realities that hit me once clients actually relied on my systems:

- **Reliability under load**: Systems must handle concurrent searches and ongoing uploads. One client's setup collapsed without proper queues and monitoring — resilience matters more than features.
- **Evaluation and testing**: Demos mean nothing if users can't trust results. Gold datasets, regression tests, and feedback loops are essential.
- **Business alignment**: Tech fails if staff aren't trained or ROI isn't clear. Adoption and compliance matter as much as embeddings.
- **Domain messiness**: Healthcare jargon, financial filings, legal precedents — every industry has quirks that make or break your system.
- **Security expectations**: Enterprises want guarantees: on-prem deployments, role-based access, audit logs. One law firm required every retrieval call to be logged immutably.

This is the stage where side projects turn into real production systems.

The Real Opportunity

If you push through this learning curve, you'll have rare skills. Enterprises everywhere need RAG/agent systems, but very few engineers can actually deliver production-ready solutions. I've seen it firsthand – companies don't care about flashy demos. They want systems that handle their messy, compliance-heavy data. That's why deals go for $50K$–200K+. It's not easy: debugging is nasty, the learning curve steep. But that's also why demand is so high. If you stick with it, you'll find companies chasing *you*.

So start building. Break things. Fix them. Learn. Solve real problems for real people. The demand is there, the money is there, and the learning never stops.

And I'm curious: what's been the hardest real-world roadblock you've faced in building or even just experimenting with RAG systems? Or even if you're just learning more in this space, I'm happy to help in any way.

Note: I used Claude for grammar/formatting polish and formatting for better readability

---

Comments

> **Low_Acanthisitta7686** • 20 points •
>
> Here is the complete technical guide if anyone is interested to deep dive: https://www.reddit.com/r/LLMDevs/comments/1n98lsf/building_rag_systems_at_enterprise_scale_20k_docs/
>
> track me

> **wildyam** • 3 points •
>
> Thanks for taking the time to post this!
>
> > **Low_Acanthisitta7686** • 1 points •

❤️

dr_tardyhands • 3 points •

Appreciate the write-up! One question: tech-stack-wise: any hot tips? Or at least what to avoid? You mentioned an Andrew Ng course on LangChain (or graph?), but I hear tons of people warning about bringing those anywhere near production.

> **Low_Acanthisitta7686** • 1 points •
>
> yeah I guess I mentioned it above, the videos will give you the foundational skills only, from there you have to build and learn.

Sad_Perception_1685 • 4 points •

Solid write-up. +1 on metadata, doc QA, and evals — that's where toy → prod actually happens.

If you ever share a v2, I'd love hard numbers: retrieval recall@50 targets, rerank p95 budgets, faithfulness/citation thresholds, and your re-chunk/re-embed policy. That's the stuff most teams miss.

> **Low_Acanthisitta7686** • 3 points •
>
> Thank you! Yeah, I wanted to keep the post timely—maybe I'll share them in v2 :)
>
> > **Sad_Perception_1685** • 3 points •
> >
> > Kinda my lane too. I wrap RAG with a governance layer: cite-first prompts, drift gates, ACL checks at retrieval *and* synthesis, and a BLAKE3 chained audit so every answer has verifiable spans. Your post nails the production pain, metadata/evals are where it gets real lol

FakeTunaFromSubway • -2 points •

Why bother posting if you're just copying GPT-5? Why waste the time and tokens? Genuinely curious

> **Sad_Perception_1685** • -1 points •
>
> Genuinely curious as to why I am copying from chat gpt-5? Who's time? Whose tokens? I am genuinely curious.

Kong28 • 2 points •

Thanks man, I just got brought on to help a fix a support bot. We use a 3rd party solution for the chat bot, but I'm in charge of cleaning up the prompt instructions and the RAG context documents. The resources you shared will get a better handle of everything going on under the hood, very much appreciated!

I also really liked your Claude note! It's refreshing to see it disclosed instead of wonder if this was fully AI generated.

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

foobarrister • 2 points •

Have you tried using AWS Bedrock Knowledge Bases? I find them to be way easier to start with for small/medium size projects because these are pre-built tools, created by people who hopefully knew wtf they were doing.

So, leveraging these PaaS offerings was a huge help for me starting out because you can POC a full e2e pipeline in a matter of hours.

NOTE: still these things are no magic button, they require significant expertise in implementing correctly. Especially when it comes to Code RAG - that's a difficult problem to solve properly.

> **Low_Acanthisitta7686** • 1 points •
>
> whats the scale your dealing with? does the offerings even work properly for 1000 docs?
>
> > **foobarrister** • 1 points •
> >
> > Yeah there are no limits on the number of documents
> >
> > > **Low_Acanthisitta7686** • 1 points •
> > >
> > > I know there aren't really limits, but I'm asking if the system still works as expected once you're dealing with thousands of documents. Maybe for simple retrieval it's fine, but for more complex stuff—like analyzing across multiple docs, basically those 'needle in a haystack' type queries and analysis.
> > >
> > > > **cbusmatty** • 1 points •
> > > >
> > > > I'm not the guy but you can use any underlying vector or graph db in aws kbs. Pinecone for example. It generally defaults to using OpenSearch, which can be expensive to run there.
> > > >
> > > > Love your stuff. My white whale is graphrag or knowledge graphs and an effective "chat with your codebase". It doesn't feel like there is a single solution that solves that problem, am I missing something obvious ?

AlienNTechnology4550 • 2 points •

What is your go to tech stack?

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

Acrobatic_Ice886 • 2 points •

Interesting, very good

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

DrDiecast • 0 points •

Need some help and guidance 🙏🙏

complead • 1 points •

For edge cases and document quality issues, have you considered leveraging automated tools for OCR verification and format consistency checks? They might help streamline some of the grunt work and improve overall efficiency. It could free up time for tackling more complex aspects of RAG systems.

> **Low_Acanthisitta7686** • 1 points •
>
> have tried some, does not solve the problems I deal with usually.

> **Krunkworx** • 1 points •
>
> My biggest pain point is I can’t send sensitive chunks to an LLM api eg OpenAI. How did you solve that?
>
> > **GreetingsFellowBots** • 1 points •
> >
> > Run a local LLM on premise.

> **jsuvro** • 1 points •
>
> Thanks for the post. Can you tell me a little about how you handled compliance? Is guardrails a good way to implement compliance?

> **Felistoria** • 1 points •
>
> Thank you for taking the time to write this up!
>
> > **Low_Acanthisitta7686** • 1 points •
> >
> > welcome :)

> **Disastrous_Grass_376** • 1 points •
>
> thanks for sharing. I will need it soon
>
> > **Low_Acanthisitta7686** • 1 points •
> >
> > sure!

> **jcumb3r** • 1 points •
>
> Good post OP. Thank you.
>
> > **Low_Acanthisitta7686** • 1 points •
> >
> > welcome :)

> **lunied** • 1 points •
>
> How do you chunk your docs in a way to keep semantic meaning as much as possible?
>
> I've heard some methods like agentic RAG/Embedding where you feed your doc into an LLM to let it chunk for you, but that's added costs.