Remember: This is your only AI project — master it deeply. You should be able to explain every design decision, technology choice, and trade-off. Good luck.

Priority Study Plan for RAG System Mastery-----Week 1: Core AI/RAG (Most Important)

1. RAG Fundamentals ⭐⭐⭐⭐⭐

- **Concepts:** What RAG is and why it matters.
- **Pipeline:** Ingestion → Retrieval → Generation.
- **Key Paper:** RAG: Retrieval-Augmented Generation (Lewis et al.)

2. Embeddings ⭐⭐⭐⭐⭐

- **Technology:** `sentence-transformers`, model `all-mpnet-base-v2` (768 dimensions).
- **Concepts:** What embeddings are, semantic similarity, cosine similarity.
- **Insight:** Why the same model is used for documents and queries.
- **Resource:** Sentence Transformers documentation

3. Vector Databases & pgvector ⭐⭐⭐⭐⭐

- **Technology:** PostgreSQL + `pgvector` extension.
- **Indexing:** HNSW indexing (understand its role in scaling).
- **Usage:** Vector similarity search queries.
- **Resource:** pgvector GitHub

4. Hybrid Search & Retrieval ⭐⭐⭐⭐⭐

- **Technologies:** Vector search + BM25 (`rank-bm25`).
- **Concepts:** Hybrid search, weighted sum (0.7/0.3 weighting), BM25 algorithm.
- **Re-ranking:** Cross-encoder re-ranking (`sentence-transformers` CrossEncoder).
- **Process:** Two-stage ranking (bi-encoder → cross-encoder).

5. LangChain ⭐⭐⭐⭐

- **Technologies:** `langchain`, `langchain-google-genai`.
- **Usage:** Text splitting (`RecursiveCharacterTextSplitter`), LLM integration, agents.
- **Resource:** LangChain documentation

6. LLMs & Prompt Engineering ⭐⭐⭐⭐

- **Technology:** Google Gemini (`gemini-1.5-flash`).
- **Concepts:** System prompts, temperature, token limits, streaming.

-----Week 2: Backend Technologies7. FastAPI ⭐⭐⭐⭐

- **Technology:** `fastapi`, `uvicorn` (async ASGI server).
- **Concepts:** Async/await, dependency injection, Pydantic validation, SSE streaming.

- **Resource:** FastAPI documentation

8. SQLAlchemy (Async) ⭐⭐⭐⭐

- **Technology:** `sqlalchemy[asyncio]`, `alembic` (migrations).
- **Concepts:** Async ORM, `AsyncSession`, models, relationships, migrations.

9. PostgreSQL ⭐⭐⭐⭐

- **Concepts:** SQL, indexes (B-tree, HNSW), foreign keys, query optimization.
- **Usage:** `pgvector`, multi-tenant queries, JSON columns.

10. Authentication & Security ⭐⭐⭐

- **Technologies:** `python-jose` (JWT), `passlib[bcrypt]`, `cryptography` (AES).
- **Concepts:** JWT tokens, password hashing, encryption vs. hashing.

-----Week 3: Frontend & Additional Tech11. Next.js 14 (App Router) ⭐⭐⭐

- **Technology:** `next` with App Router.
- **Concepts:** Server Components, Client Components, routing, data fetching.

12. React & TypeScript ⭐⭐⭐

- **Technologies:** `react`, `typescript`.
- **Concepts:** Hooks, component patterns, type safety.

13. Document Processing ⭐⭐⭐

- **Technologies:** `pdfplumber`, `pypdf2`, `python-docx`, `pandas`.
- **Concepts:** PDF extraction challenges, text preprocessing.

14. AWS S3 ⭐⭐

- **Technology:** `boto3`.
- **Concepts:** S3 basics, file storage strategies.

-----Week 4: System Design & Architecture15. Multi-tenancy ⭐⭐⭐⭐⭐

- **Pattern:** Shared database with `tenant_id` filtering.
- **Concepts:** Data isolation, security, row-level filtering.

16. Scalability & Performance ⭐⭐⭐⭐

- **Concepts:** Horizontal scaling, indexing, caching, query optimization.
- **Optimizations:** HNSW index, top-k retrieval, batch processing.

-----Must-Know Interview QuestionsRAG-Specific

1. Explain your RAG pipeline end-to-end.

2. Why hybrid search over pure vector search?
3. How do you handle chunking? Why 1500 chars?
4. Difference between bi-encoder and cross-encoder?
5. How do you evaluate RAG performance?

System Design

1. How to scale to 10 million documents?
2. How do you ensure multi-tenant data isolation?
3. What are bottlenecks in RAG systems?
4. How does pgvector/HNSW work internally?

Technical Deep Dives

1. Walk through a query from API → DB → LLM.
2. How does HNSW improve search speed?
3. Why Sentence Transformers vs. OpenAI embeddings?
4. How do you handle embedding model updates?

-----Quick Reference Card

Technology	Your Usage	Key Concept
Sentence Transformers	`all-mpnet-base-v2` (768d)	Local embeddings, no API costs
pgvector	HNSW indexing	Vector similarity search in PostgreSQL
Hybrid Search	Vector (0.7) + BM25 (0.3)	Combines semantic + keyword
Re-ranking	Cross-encoder	Two-stage ranking for accuracy
Chunking	1500 chars, 300 overlap	`RecursiveCharacterTextSplitter`
FastAPI	Async endpoints, SSE	Modern Python web framework

SQLAlchemy	Async ORM	Database abstraction
Next.js	App Router	React framework
Multi-tenancy	Tenant ID filtering	Data isolation strategy

-----Study ResourcesPapers

- RAG: Retrieval-Augmented Generation
- HNSW Paper

Documentation

- Sentence Transformers
- pgvector
- LangChain
- FastAPI

Your Project Guides

- Read `RAG_COMPREHENSIVE_GUIDE.md` (detailed RAG explanation).
- Review your codebase (understand every file).

-----Pro Tips

1. **Know your numbers:** chunk size (1500), overlap (300), embedding dims (768), hybrid weights (0.7/0.3).
2. **Understand trade-offs:** For each decision, know what you sacrificed (e.g., weighted sum vs. RRF).
3. **Practice explaining:** Record yourself explaining the RAG pipeline in 2 minutes.
4. **Code review ready:** Be able to explain any file in your project.
5. **Draw diagrams:** Practice architecture diagrams for system design questions.

-----Final Checklist

Before the interview, you should be able to:

- Explain RAG pipeline in 2 minutes
- Draw your system architecture
- Explain why you chose each technology
- Discuss trade-offs (hybrid search, chunking, embeddings)
- Answer "How would you scale to 10M documents?"

- Explain multi-tenancy implementation
- Walk through code in `retriever.py`, `ingestion.py`, `answer_engine.py`

Remember: This is your only AI project — master it deeply. You should be able to explain every design decision, technology choice, and trade-off. Good luck.