📚 Lecture 08 – Retrieval Augmented Generation (RAG) Complete Guide

---

🔷 RAG Ki Basic Samajh

🚀 RAG Kya Hai?

Full Form: Retrieval Augmented Generation

🧠 **Simple Definition:**
RAG ek aisa system hai jo **user ke question ka answer** dene ke liye pehle **apne documents se relevant information retrieve** karta hai,
phir **us context ko LLM (Large Language Model)** ko deta hai, jisse woh **accurate aur relevant answer generate** kar sake.

🎬 Example (from YouTube Video)

> Agar aap LLM ko bolo: “Meri girlfriend ki tarah behave karo”
> aur phir pucho: “Tumhe kahan ghumna pasand hai?”
> Toh LLM bolega — “Chicago” 😅
> Par asli girlfriend Uttarakhand se hai!
> Yeh galat output hai → **Hallucination**

💡 RAG Kyu Zaroori Hai?

- ❌ **Hallucination Problem:** LLM ko answer nahi pata hota, fir bhi kuch bhi bata deta hai
- 🩹 **Limited Context:** LLM ke paas aapke personal documents ka knowledge nahi hota
- 💰 **Cost Effective:** Fine-tuning se kaafi sasta hota hai
- 🔄 **Dynamic:** Naye documents easily add kar sakte hain

🔷 RAG vs Fine-Tuning – Complete Comparison

📌 Fine-Tuning

Definition: LLM ko naye data par dobara train karna.

Example: WhatsApp chat dekar LLM ko bolna — “Is data par train ho jao”.

Disadvantages:

- 💸 Bahut expensive (GPU cost high)
- 🕒 Time-consuming
- 🔁 Har baar naya data aane par retraining karni padti hai

📌 RAG

Definition: Documents ko context ke roop mein dena.
Example: WhatsApp chat ko context ke roop mein LLM ko dena — LLM wahi se answer karega.

Advantages:

- ⚡ Fast & Cheap
- 🔄 Real-time document updates possible
- 🧠 No training required

🎯 Comparison Table

Aspect	Fine-Tuning	RAG
💸 **Cost**	High (GPU, Training)	Low (API Calls)
⏱️ **Time**	Days/Weeks	Minutes/Hours
🔧 **Flexibility**	Low	High
🧹 **Maintenance**	Difficult	Easy

🧠 **Analogy:**
Fine-Tuning = Student ko poori book ratta marwana 📖
RAG = Student ko exam hall mein book le jaane dena 📚

🔷 RAG Ka Complete Flow (2 Steps)

🚀 Step 1: Indexing Phase (Ek Baar Ka Kaam)

🎯 **Goal:** Documents ko Vector Database mein store karna

1. 📄 Document Loading

```
const loader = new PDFLoader("dsa.pdf");

const rawDocs = await loader.load();
```

📒 Example: 112 pages ki PDF load hoti hai.

---

2. ✂️ Chunking (Tukdon Mein Katna)

Kyu? – Taki baad mein sirf relevant parts retrieve ho sake.
Chunk Size: 1000 words
Overlap: 200 words

💡 **Overlap Importance:**
Agar “Linked List” ki definition do chunks mein split ho jaye,
to overlap se dono chunks mein kuch context bacha rahega.

```
const splitter = new RecursiveCharacterTextSplitter({

    chunkSize: 1000,

    chunkOverlap: 200

});

const chunkDocs = await splitter.splitDocuments(rawDocs);
```

📊 Example: 112 pages → 227 chunks

---

3. 🔢 Vector Conversion

Definition: Text → Numbers (numerical representation)
Model: Google Generative AI – `text-embedding-004`
Dimensions: 768

```
const embeddings = new GoogleGenerativeAIEmbeddings({

    model: "text-embedding-004"

});
```

---

4. 💾 Vector Database Storage

Database: Pinecone

🧩 Setup Steps:

1. **pinecone.io** par sign up karo
2. Index create karo (768 dimensions)
3. API key & environment le lo

```
const pinecone = new Pinecone();

const index = pinecone.Index("rohit-neg");


await PineconeStore.fromDocuments(chunkDocs, embeddings, {

    pineconeIndex: index,

    maxConcurrency: 5

});
```

⚙️ 5 chunks ek saath process honge.

🚀 Step 2: Query Phase (User Interaction)

🎯 **Goal:** User ke question ka accurate answer dena

1. ❓ User Input Lena

```
const userQuestion = "What is AVL tree?";
```

2. 🔢 Query Ko Vector Mein Convert Karna

```
const queryVector = await embeddings.embedQuery(userQuestion);
```

3. 🔍 Vector Database Mein Search Karna

```
const results = await index.query({

    vector: queryVector,

    topK: 10,

    includeMetadata: true

});
```

4. 📝 Context Create Karna

```
const context = results.matches.map(match =>

    match.metadata.text

).join("\n\n");
```

5. 👹 LLM Ko Context + Question Dena

```
const model = new GoogleGenerativeAI({ model: "gemini-pro" });

const response = await model.generateContent({

    contents: [

        {

            role: "user",

            parts: [{

                text: `You are a DSA expert. Answer based ONLY on this
context:


Context: ${context}

Question: ${userQuestion}


If answer is not in context, say "I couldn't find the answer in provided
documents".`

            }]

        }

    ]

});
```

🔷 Advanced RAG Concepts

🎯 Query Transformation Problem

❌ **Problem:**
User: “What is Quick Sort?” → LLM correct answer deta hai
User: “Explain it in detail” → ❌ “I couldn’t find the answer”

Reason:
“Explain it in detail” ka vector different hai,
toh DB ko relevant documents nahi milte.

✅ **Solution: Query Transformation Layer**

```
async function transformQuery(question, history) {

    const model = new GoogleGenerativeAI({ model: "gemini-pro" });

    const response = await model.generateContent({

        contents: [

            {

                role: "user",

                parts: [{

                    text: `You are a query rewriting expert.

Based on chat history, rewrite this follow-up question into a standalone
question.


Chat History: ${history}

Latest Question: ${question}`

                }]

            }

        ]

    });

    return response.text();

}
```

Example:

> “Explain it in detail” → “What is Quick Sort in depth?”

🎯 Token Cost Optimization

Problem: Poora document dene se token usage zyada hota hai.
Solution: Sirf **relevant chunks** bhejo.

💡 Example: Agar “Course Revenue” puchha jaaye, toh sirf revenue related chunks bhejo, poori PDF nahi.

🔷 Complete Project Implementation

📂 File Structure

```
project/
├── index.js        # Indexing phase
├── query.js        # Query phase
├── .env            # API keys
└── dsa.pdf         # Input document
```

🔗 Dependencies

```
npm install @langchain/community @langchain/google-genai pinecone-client
pdf-parse
```

📝 Indexing Code (index.js)

```
// Load PDF

const loader = new PDFLoader("dsa.pdf");

const rawDocs = await loader.load();


// Split into Chunks

const splitter = new RecursiveCharacterTextSplitter({
```

```
  chunkSize: 1000,
  chunkOverlap: 200
});
const chunkDocs = await splitter.splitDocuments(rawDocs);

// Create Embeddings
const embeddings = new GoogleGenerativeAIEmbeddings({
  model: "text-embedding-004",
  apiKey: process.env.GOOGLE_API_KEY
});

// Connect Pinecone
const pinecone = new Pinecone({
  apiKey: process.env.PINECONE_API_KEY
});

// Store in Vector DB
await PineconeStore.fromDocuments(chunkDocs, embeddings, {
  pineconeIndex: pinecone.Index("rohit-neg"),
  maxConcurrency: 5
});
```

📝 Querying Code (query.js)

```
const userQuestion = "What is AVL tree?";

const queryVector = await embeddings.embedQuery(userQuestion);

const results = await index.query({

  vector: queryVector,

  topK: 10,

  includeMetadata: true

});

const context = results.matches.map(m => m.metadata.text).join("\n\n");


const response = await model.generateContent({

  contents: [{

    role: "user",

    parts: [{

      text: `You are a DSA expert. Answer using ONLY this context:
${context}`

    }]

  }]

});
```

🔷 .env File Setup

```
GOOGLE_API_KEY=your_google_api_key

PINECONE_API_KEY=your_pinecone_api_key

PINECONE_ENVIRONMENT=your_pinecone_environment
```

🔷 Common Issues & Solutions

Problem	Cause	Solution
❌ Vague Questions	Context missing	Use query transformation
💸 High Token Cost	Full doc sent	Send only relevant chunks
🤔 Wrong Answers	LLM using own data	Restrict: “Use only provided context”
🐢 Slow Speed	Too many chunks	Adjust `maxConcurrency`

🔷 Testing & Validation

✅ Test Cases

- “What is AVL Tree?” → Correct Answer
- “Explain it in detail” → Works after transformation
- “Who is Rohit Negi?” → “I couldn’t find the answer”
- “Quick Sort time complexity?” → Exact numerical result

🎯 Validation

- Pinecone dashboard → vectors visible
- Context retrieval → relevant
- LLM output → only context-based

🔷 Production-Ready Tips

💡 Performance Optimization

- Chunk Size → 500–1000 words
- Top K → 5–10 results
- Concurrency → 3–5 (for free tier)

💡 Cost Optimization

- Use free tiers (Google AI, Pinecone)
- Monitor token usage
- Implement caching for frequent queries

💡 Error Handling

```
try {
  // RAG code
} catch (err) {
  console.error("RAG Error:", err);
}
```

🔷 Real-World Examples

🎯 1. Girlfriend Chatbot

- Problem: LLM doesn't know personal info
- Solution: WhatsApp chat as context
- Result: Personalized accurate responses

🎯 2. Coder Army Revenue

- Problem: LLM doesn't know company revenue
- Solution: Give sales data as context
- Result: Exact revenue fetched

🎯 3. DSA PDF Q&A

- Problem: LLM lacks DSA knowledge
- Solution: DSA PDF in vector DB
- Result: Contextual, accurate DSA answers

🌟 RAG System – Final Summary

Concept	Explanation
🔍 **Retrieval**	Query se related documents dhoondhna
🧩 **Augmentation**	Un documents ko context ke roop me dena
🧠 **Generation**	Context ke basis par answer banana

📌 Why RAG?

- Hallucination reduction 🧱
- Cost-effective 💰
- Real-time document updates 🔄
- Domain-specific knowledge injection 🧭

⚙️ How It Works

1. **Indexing:** Documents → Chunks → Vectors → Vector DB
2. **Querying:** Question → Vector → Search → Context + Question → LLM → Answer

🚀 Advanced Features

- Query Transformation
- Multi-Document Handling

- Token Cost Optimization
- History-Aware Querying

🧰 Tools Used

- 🧮 Vector DB → Pinecone
- 🔤 Embeddings → Google Generative AI
- 🧠 Framework → LangChain
- 🤖 LLM → Gemini Pro

🎯 Final Outcome

RAG system banakar:

- LLM hallucination door hoti hai
- Context-based, accurate, real-world responses milte hain
- System scalable, dynamic aur production-ready hota hai 💪

✨ **Conclusion:**

RAG = LLM ka “Exam Hall Companion” 🧠📚
Jo use sirf “sahi notes” dekar perfect answer likhne mein madad karta hai! 🎯