RAG System Explained - Simple Version

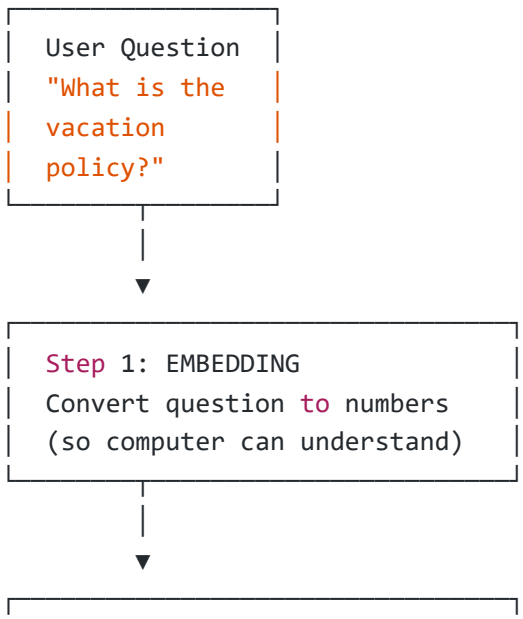
What is this system?

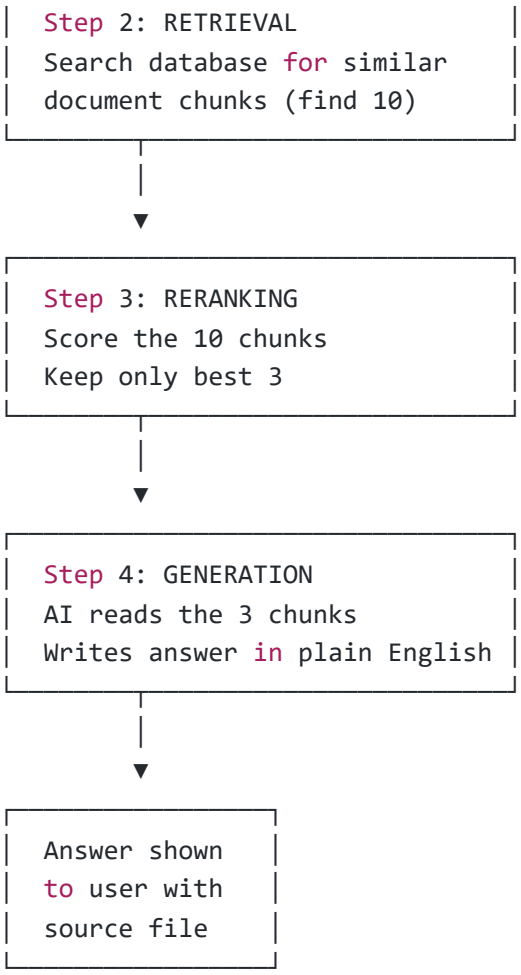
This is a smart chatbot that helps employees find information from company documents. Instead of searching through hundreds of pages, you just ask a question and get an answer with the source.

Overall Architecture Diagram

```
USER asks question
    ↓
SYSTEM searches documents
    ↓
SYSTEM finds best matches
    ↓
SYSTEM ranks them by relevance
    ↓
AI generates answer from matches
    ↓
USER gets answer + source
```

Visual Flow:





End-to-End Data Flow

1. Load Documents
   - System reads PDF, Word, HTML files
   - Example: HR_Policies/vacation.pdf

2. Break into Chunks
   - Split each document into small pieces (1500 characters each)
   - Why? So we can find exact relevant parts
   - Example: One chunk = "Vacation Policy: Employees get 15 days..."

3. Convert to Numbers (Embeddings)

   - Similar chunks get similar numbers

4. Store in Database
   - Save all chunks with their numbers in FAISS database
   - Also save: which file, which category

Part 2: Answering Questions

What happens: System finds relevant info and generates answer.

```
USER: "What is the vacation policy?"
    ↓
STEP 1: Convert question to numbers (same way as documents)
    ↓
STEP 2: Search database
    - Find 10 chunks with most similar numbers
    - Filter by category if detected (e.g., HR)
    ↓
STEP 3: Rerank the 10 chunks
    - Score each one more carefully
    - Keep only top 3 most relevant
    ↓
STEP 4: Generate answer
    - AI reads the 3 chunks
    - Writes answer using ONLY info from those chunks
    - Adds source citation
    ↓
ANSWER: "Employees receive 15 days of vacation per year..."
SOURCE: HR_Policies/vacation.pdf
```

How Embedding Works

Simple Explanation: Embedding = Converting text into numbers so computers can compare them.

Example:

- Text: "vacation policy"
- Embedding: [0.23, -0.45, 0.67, ... 1536 numbers total]

Why it's useful:

- Similar meanings → Similar numbers
- "vacation policy" and "time off rules" get similar numbers
- System can find related info even if exact words don't match

In our system:

1. We use AWS Titan Embeddings model
2. Every document chunk gets converted numbers

3. Every user question also gets converted numbers
4. We compare these numbers to find matches

Visual:

```
Document: "Employees get 15 vacation days"
    ↓ Titan Embeddings
Numbers: [0.12, 0.34, -0.56, ...]

Question: "How many vacation days?"
    ↓ Titan Embeddings
Numbers: [0.15, 0.32, -0.54, ...]

Compare → Very similar! → This document matches the question
```

How Retrieval Works

Simple Explanation: Retrieval = Finding the most relevant document chunks for a question.

Step-by-Step:

1. **Detect Category** (Optional)
    - System looks for keywords in question
    - "vacation" → HR_Policies category
    - "API" → Engineering_Docs category
    - Helps narrow down search
2. **Convert Question to Numbers**
    - Question: "What is vacation policy?"
    - Becomes: [0.15, 0.32, -0.54, ... 1536 numbers]
3. **Search Database**
    - Compare question numbers with all document chunk numbers
    - Use FAISS (fast search tool)
    - Find 10 chunks with closest numbers
4. **Return Results**
    - Get 10 most similar chunks

- Each has: text content, source file, category

Example:

```
Question: "What is the vacation policy?"

Search Results (Top 10):
1. "Employees receive 15 days vacation..." (HR_Policies/vacation.pdf)
2. "Vacation days can be carried over..." (HR_Policies/vacation.pdf)
3. "Time off requests must be submitted..." (HR_Policies/timeoff.pdf)
... (7 more)
```

Where Reranking Happens

Simple Explanation: Reranking = Double-checking which chunks are REALLY the most relevant.

Why we need it:

- First search is fast but not perfect
- Reranking is slower but more accurate
- Better to spend extra time and get the right answer

When it happens: After retrieval, before generation.

How it works:

1. **We have 10 chunks from retrieval**
2. **Re-embed each chunk**
   - Convert each chunk to numbers again (fresh calculation)
   - More accurate than stored embeddings
3. **Calculate similarity scores**
   - Compare question numbers with each chunk numbers
   - Use dot product (multiply and add)
   - Higher score = more relevant
4. **Sort by score**
   - Rank chunks from highest to lowest score

5. **Keep top 3**
    - Only send best 3 chunks to AI
    - Reduces noise, improves answer quality

Example:

```
After Retrieval (10 chunks):
Chunk 1: Score 0.85
Chunk 2: Score 0.92  ← Best
Chunk 3: Score 0.78
Chunk 4: Score 0.88  ← 2nd best
Chunk 5: Score 0.65
Chunk 6: Score 0.81  ← 3rd best
Chunk 7: Score 0.55
Chunk 8: Score 0.70
Chunk 9: Score 0.60
Chunk 10: Score 0.50

After Reranking (keep top 3):
✓ Chunk 2 (0.92)
✓ Chunk 4 (0.88)
✓ Chunk 6 (0.81)
```

How Generation Works

Simple Explanation: Generation = AI reads the relevant chunks and writes an answer in plain English.

Step-by-Step:

1. **Prepare Context**
    - Take the 3 best chunks from reranking
    - Format them nicely for AI
    - Add labels (category/filename)
2. **Create Prompt**
    - Combine: chunks + question + instructions
    - Instructions tell AI: "Only use info from these chunks"
3. **Send to AI (Claude 3 Haiku)**

- AI reads everything
- Writes answer based ONLY on the chunks
- Cannot make up information

4. **Add Citation**

- System adds source file at the end
- Shows which document the answer came from

5. **Return to User**

- User sees answer + source

Example:

Input to Chatbot:

```
Context:
[HR_Policies/vacation.pdf]
"Employees receive 15 days of vacation per year. Vacation days
can be carried over to the next year up to a maximum of 5 days."

[HR_Policies/vacation.pdf]
"Vacation requests must be submitted at least 2 weeks in advance
through the HR portal."

Question: "What is the vacation policy?"

Instructions: Answer using ONLY the context above.
```

Output from Chatbot:

```
Employees receive 15 days of vacation per year. Vacation days can
be carried over to the next year, but only up to 5 days maximum.
To request vacation, you must submit your request at least 2 weeks
in advance through the HR portal.

---
📚 Source Referenced: HR_Policies/vacation.pdf
```

Complete Example: User Question to Answer

User asks: "How many vacation days do I get?"

Step 1: Embedding (0.2 seconds)

- Convert question to numbers: [0.15, 0.32, -0.54, ...]

Step 2: Retrieval (0.05 seconds)

- Search database for similar chunks
- Find 10 candidates from HR_Policies

Step 3: Reranking (2 seconds)

- Score all 10 chunks carefully
- Keep best 3:
  i. "Employees receive 15 days vacation..."
  ii. "Vacation days can be carried over..."
  iii. "New employees receive prorated vacation..."

Step 4: Generation (2.5 seconds)

- AI reads the 3 chunks
- Writes: "Employees receive 15 days of vacation per year..."
- Adds source: HR_Policies/vacation.pdf

Total time: ~5 seconds

User sees:

```
Employees receive 15 days of vacation per year. Vacation days
can be carried over to the next year up to a maximum of 5 days.

---
📚 Source Referenced: HR_Policies/vacation.pdf
```

Summary

What the system does:

1. Stores all company documents as searchable numbers
2. When you ask a question, finds relevant parts

3. Ranks them by relevance
4. AI reads them and writes an answer
5. Shows you the source

Why it's useful:

- Fast: Get answers in 5 seconds
- Accurate: Only uses real documents
- Traceable: Always shows source
- Smart: Understands meaning, not just keywords

Technology used:

- AWS Titan: Converts text to numbers
- FAISS: Fast search database
- Claude AI: Writes answers
- Python: Connects everything