AI Interview Prep

Simple Explanations for Non-AI Students

RAG • Agents • Search Algorithms • Knowledge Representation

How to Use This Guide
Each topic starts with a simple real-life analogy so you can explain it in plain English.
Then you get the technical definition for completeness.
Look for the interview boxes — those show exactly what to say when asked!
✓Green boxes = Key tips to remember. Orange boxes = Interview Q&A.

PART 1: RAG — Retrieval-Augmented Generation

1. What is RAG?

Think of RAG like an open-book exam.

A regular AI model (like ChatGPT) tries to answer from memory — like a closed-book exam. It can only use what it learned during training. If you ask about something recent or something niche, it might guess or get it wrong.

RAG gives the AI a textbook to look at while answering. Before answering your question, it searches its own library of documents, finds the most relevant pages, and then reads those pages to give you a better answer.

Interviewer: What is RAG and why is it useful?

✓**You say:** RAG stands for Retrieval-Augmented Generation. It combines a search system with an AI model. Instead of relying on what the AI already knows, RAG first searches a knowledge base for relevant documents, then passes those documents to the AI to generate an answer. This makes answers more accurate and up-to-date — like giving the AI an open book during an exam.

RAG Pipeline — Step by Step

Here is the complete journey of a user's question through a RAG system:

1. **User asks a question** — e.g. "What is our refund policy?"
2. **Query is converted to a vector** — the question is turned into a list of numbers that captures its meaning.
3. **Vector store is searched** — the system finds documents with similar meaning.
4. **Top results are reranked** — the most relevant documents are moved to the top.
5. **LLM generates the answer** — the AI reads the retrieved documents and writes a response.

2. Chunking — Breaking Documents into Pieces

Think of chunking like splitting a book into index cards.

Imagine you want to store a 500-page book in a searchable database. You can't search the whole book at once — it's too big. So you cut it into small pieces (chunks) and store each piece separately. When someone searches, you find the most relevant chunk instead of the whole book.

Key Numbers to Remember
Chunk size: usually 256–1024 tokens (roughly 200–800 words)
Overlap: 10–20% overlap between chunks so you don't lose context at the edges
Metadata: attach info like page number and document title to each chunk

Types of Chunking Strategies

Strategy	What it does	Best for
Fixed-size	Splits at every N words/characters, regardless of meaning	Simple documents
Recursive	Tries to split at paragraphs → sentences → words (most natural)	Most documents (default)
Sentence-based	Splits at sentence boundaries	Articles, prose
Semantic	Groups sentences by topic — splits when meaning changes	Research papers
Document-based	Splits at headings, page breaks, or HTML sections	Structured documents

Interviewer: What chunking strategy would you use and why?

✅**You say:** I would start with recursive chunking — it's the most natural because it tries to split at paragraph breaks first, then sentence breaks. I'd use a chunk size of around 512 tokens with 10% overlap to avoid losing context at boundaries. For structured documents like PDFs with headers, I'd use document-based chunking to keep sections intact.

3. Embeddings — Turning Text into Numbers

Think of embeddings like GPS coordinates for meaning.

Just like GPS turns a location into two numbers (latitude, longitude), an embedding model turns a sentence into hundreds of numbers. Sentences with similar meaning end up with "coordinates" that are close together. This lets us search by meaning, not just keywords.

Example: "What is your return policy?" and "How do I get a refund?" would have very similar embeddings even though they use different words.

Popular Embedding Models

Model	Provider	Simple description
text-embedding-3-small	OpenAI	Good quality, affordable — great starting point
text-embedding-3-large	OpenAI	Higher quality, more expensive — for production apps
all-MiniLM-L6-v2	Open source	Fast and lightweight — good for local/offline use
BGE-large-en	Open source (BAAI)	Top-performing free model for English text
Cohere embed-v3	Cohere	Has special modes for search vs. classification

Interviewer: What is an embedding and how does it help retrieval?

✅**You say:** An embedding converts text into a vector — a list of numbers that represents the meaning of the text. Similar sentences produce similar vectors. In a RAG system, we embed both the documents and the user's query, then find documents whose vectors are closest to the query vector. This allows us to search by semantic meaning rather than exact keyword matching.

4. Vector Store — A Smart Search Database

Think of a vector store like a library that organizes books by topic, not by title.

A normal database searches by exact text match. A vector store searches by meaning. It stores millions of embeddings and can instantly find the ones closest to your query — even if the words are completely different.

Popular Vector Databases

Name	Type	Best used when...
FAISS	Library (local)	Prototyping — fast, runs in memory, free
ChromaDB	Library (local)	Small projects, easy to set up locally
Pinecone	Cloud service	You want a fully managed solution with no setup
Weaviate	Self-hosted/Cloud	You need keyword + vector hybrid search
Qdrant	Self-hosted/Cloud	Production apps needing fast, rich filtering
pgvector	PostgreSQL add-on	You already use PostgreSQL and want to keep one DB

Good Interview Tip

If asked which vector DB to pick, say: 'For prototyping I'd use FAISS or ChromaDB because they're simple to set up. For production I'd use Pinecone or Qdrant based on scale and filtering needs. If the team already uses PostgreSQL, pgvector is a great option to avoid a new service.'

5. Retriever — Finding the Right Documents

Think of the retriever as the librarian who goes and finds the books.

The retriever is the part of the RAG system that actually performs the search. It takes the user's query, and returns the most relevant chunks from the vector store. There are several types:

Type	How it works	When to use it
Dense (vector)	Embeds query → searches by meaning similarity	General semantic search
Sparse (BM25)	Classic keyword matching (like Google search)	Exact terms, legal/medical docs
Hybrid	Combines vector + keyword search	Best of both — most robust

Type	How it works	When to use it
Multi-query	Generates multiple versions of the query, searches all	When users phrase questions differently
Self-query	AI extracts filters from the question itself (e.g. 'after 2023')	Structured data with metadata

Interviewer: What type of retriever would you use?

✅**You say:** I'd use a Hybrid Retriever that combines dense (vector) and sparse (keyword) search. Dense search handles semantic understanding — finding relevant documents even when different words are used. Sparse search (like BM25) is better for exact matches like product names or legal terms. Combining both using Reciprocal Rank Fusion gives the best recall.

6. Prompt Engineering — Talking to AI Effectively

Think of prompts like instructions you give to a very literal assistant.

The quality of the AI's answer depends hugely on how you ask the question. Prompt engineering is the skill of crafting those instructions carefully.

Core Techniques

Technique	What it is	Example
Zero-shot	Just ask without examples	"Summarize this document in 3 bullet points."
Few-shot	Give 2-3 examples before asking	"Q: What is AI? A: ... Now answer: Q: What is ML?"
Chain-of-Thought	Ask the AI to think step by step	"Let's think through this step by step..."
Role prompting	Give the AI a persona	"You are a senior software engineer. Review this code."
Output formatting	Tell AI what format to respond in	"Respond in JSON with keys: name, score, reason."

RAG Prompt Template — Use this in interviews!

System: "You are a helpful assistant. Answer ONLY based on the provided context below. If the answer is not found in the context, say I don't know. Do not make up information."

Context: [retrieved documents go here]
User question: [user question goes here]

7. Evaluating a RAG System

How do you know if your RAG system is actually working well?

You evaluate it in two parts: How good is the retrieval? And how good is the generated answer?

Retrieval Metrics

Metric	Simple meaning
Context Relevance	Did we retrieve documents that are actually about the question?
Context Recall	Did we retrieve ALL the important documents, or miss some?
Context Precision	Did we avoid retrieving useless/irrelevant documents?

Generation Metrics

Metric	Simple meaning
Faithfulness	Is the answer based on the documents — or did the AI make things up?
Answer Relevance	Does the answer actually address what was asked?
Answer Correctness	Is the answer factually correct?

Evaluation Frameworks to Name-Drop
RAGAS — most popular automated evaluation framework for RAG systems (faithfulness, relevance, context metrics)
DeepEval — unit testing framework for LLM applications
LangSmith — tracing and evaluation tool from LangChain
Human evaluation — still the gold standard for catching subtle issues

Interviewer: How would you evaluate your RAG system?

✅**You say:** I'd evaluate it in two parts. First, retrieval quality: are we fetching relevant, complete, and precise documents? I'd measure context relevance and recall. Second, generation quality: is the answer faithful to the retrieved context, relevant to the question, and factually correct? I'd use RAGAS as an automated framework, plus build a golden dataset of test questions with known correct answers to benchmark against.

PART 2: AI Agents

8. Types of AI Agents

An AI agent is a program that perceives its environment and takes actions to achieve a goal.

Think of it like different types of workers with increasing intelligence:

Agent Type	Simple analogy	How it works
Simple Reflex Agent	A light switch	If condition → do action. No memory, no thinking.
Model-Based Agent	A security guard with a map	Keeps an internal picture of the world to handle incomplete info.
Goal-Based Agent	Google Maps navigation	Has a destination and plans the best route to get there.
Utility-Based Agent	Self-driving car	Picks the action that maximizes a score (safety + speed + fuel).
Learning Agent	Netflix recommendations	Gets better over time by learning from experience.

Interviewer: Can you explain the types of AI agents?

✅**You say:** Sure — they go from simple to complex. A Simple Reflex Agent just responds to triggers with no memory, like a thermostat. A Goal-Based Agent has a destination and plans how to get there. A Utility-Based Agent picks the best action from multiple choices by scoring them. A Learning Agent improves over time from experience. In real AI apps today, most LLM-based agents are goal-based or utility-based.

9. How an Agent Formulates a Problem

Before an agent can solve anything, it needs to define the problem clearly.

Every AI problem has 5 components — remember them as IATGP:

Component	What it means	Navigation example
Initial State	Where do we start?	You are in City A
Actions	What can we do?	Drive to neighboring cities
Transition Model	What happens when we act?	Drive(A → B) puts us in City B
Goal Test	Are we done?	Have we reached City Z?
Path Cost	How expensive was this?	Total kilometers driven

10. BFS vs DFS — Search Algorithms

These are the two most fundamental ways to search through possibilities.

BFS — Breadth-First Search

Analogy: Spreading ripples in a pond — level by level.

BFS explores all options at the current depth before going deeper. It uses a Queue (first in, first out). Imagine looking for a person in a building: BFS checks every room on floor 1 before going to floor 2.

BFS Key Facts
Data structure: Queue (FIFO)
Finds shortest path: YES ✓
Will always find a solution if one exists: YES ✓
Memory usage: HIGH (stores all nodes at current level)
Best for: When you want the shortest solution

DFS — Depth-First Search

Analogy: Exploring a maze by always going as deep as possible before backtracking.

DFS goes as deep as it can down one path before backing up and trying another. It uses a Stack (last in, first out). Imagine reading a book: DFS reads Chapter 1 all the way to the end before starting Chapter 2.

DFS Key Facts
Data structure: Stack (LIFO) or recursion
Finds shortest path: NO ✕
Will always find a solution: NO ✕(can get stuck in loops)
Memory usage: LOW (only stores one path at a time)
Best for: When memory is limited or solution is deep

Interviewer: When would you use BFS vs DFS?

✓**You say:** BFS is my choice when I need the shortest path — like finding the fewest hops between nodes in a network. It guarantees optimality when all edges have equal cost. DFS is better when memory is limited or when I expect the solution to be deep in the search tree. In practice, A* search is often preferred over both because it uses a heuristic to guide the search more efficiently.

11. Greedy Search and A* Search

Greedy search and A* are smarter alternatives to BFS/DFS.

Greedy Search

Analogy: Always driving toward the city that looks closest on a map, even if there's a mountain in the way.

Greedy search uses a heuristic — an estimate of how far you are from the goal — and always picks the option that looks closest. It's fast but not always correct.

- Not optimal — might find a path but not the shortest one
- Not complete — can get stuck if there's no visited-node tracking
- Fast — makes fewer comparisons than BFS/DFS

A* Search — The Best of Both Worlds

A fixes Greedy search by also considering how far you've already traveled.*

Formula: f(n) = g(n) + h(n)

- **g(n)** = actual cost from start to current node (how far we've come)
- **h(n)** = estimated cost from current node to goal (how far we have to go)
- **f(n)** = total estimated cost of the path through this node

A* is both complete AND optimal — use it when accuracy matters

As long as h(n) never overestimates the real cost (this is called 'admissible'), A* is guaranteed to find the shortest path.

Common heuristics: straight-line distance for map problems, number of misplaced tiles for puzzles.

Interviewer: What is A* and how is it better than Greedy?

✅**You say:** A* improves on greedy search by combining two costs: g(n), the actual distance traveled so far, and h(n), the heuristic estimate to the goal. Greedy search only looks at h(n) — how close something looks — which can lead it down the wrong path. A* considers the full picture, so it finds the optimal path as long as the heuristic never overestimates. It's the go-to algorithm for pathfinding in games and maps.

12. Knowledge Representation

How do AI systems store and use knowledge about the world?

Knowledge Representation (KR) is how an AI organizes information so it can reason and make decisions. Think of it as the AI's memory structure.

Type	Simple analogy	Example
Logical Rules	Math equations	All humans are mortal → Socrates is human → Socrates is mortal
Semantic Networks	Mind map	Dog → IS-A → Animal; Dog → HAS → Legs
Frames	A class in programming	Car { color: red, wheels: 4, engine: V8 }
Production Rules	IF-THEN rules	IF fever AND cough THEN possible flu
Ontologies	Official taxonomy	Medical concept hierarchy (SNOMED, ICD codes)

Modern AI Knowledge Representation

In modern AI systems, knowledge is represented in two main ways:

- **Parametric knowledge** — stored inside the neural network weights (what the model learned during training)
- **Non-parametric knowledge** — stored externally in databases, documents, or knowledge graphs (what RAG retrieves at query time)

Interviewer: What is knowledge representation and why does it matter?

✓**You say:** Knowledge representation is how an AI system stores, organizes, and accesses information. Without good knowledge representation, an AI can't reason or make decisions effectively. Traditional AI used logical rules, semantic networks, and ontologies. Modern LLMs encode knowledge implicitly in their weights — this is parametric knowledge. RAG systems complement this with non-parametric knowledge by retrieving from external databases at query time, which keeps the knowledge current and verifiable.

FINAL INTERVIEW TIPS

Formula for any AI Interview Answer
1. Start with a real-life analogy (shows you truly understand it)
2. Give the technical definition (shows you know the terminology)
3. Mention a practical consideration (trade-offs, when to use it)
4. Name a tool or framework if relevant (shows industry awareness)
Example: "RAG is like an open-book exam for AI. Technically, it's a pipeline that retrieves relevant documents from a vector store before generating an answer with an LLM. A key consideration is chunking strategy and retriever type. For evaluation, I'd use the RAGAS framework."

Quick Glossary
LLM = Large Language Model (e.g. GPT-4, Claude, Gemini)
Token = roughly one word or part of a word
Vector = a list of numbers representing meaning
Embedding = the process of converting text to vectors
Heuristic = an educated guess/estimate to guide search

Corpus = a collection of documents or text data
BM25 = classic keyword-based search algorithm
HNSW = a fast algorithm for approximate nearest-neighbor vector search
Hallucination = when an AI confidently states something false