1. Architecture Patterns for Information Retrieval & Semantic Search

There are three dominant architectural patterns for building search systems today:

- **Pattern A: Bi-Encoder (Dense Retrieval)**
  - *Mechanism:* Documents and Queries are independently converted into vectors (embeddings) using a model like BERT. Similarity is calculated using Cosine/Dot Product.
  - *Architecture:* Query $\rightarrow$ Embedding Model $\rightarrow$ Vector DB (ANN Search) $\rightarrow$ Results.
  - *Pros:* Extremely fast retrieval (milliseconds).
- **Pattern B: Cross-Encoder (Re-Ranking)**
  - *Mechanism:* The Query and Document are fed into the model *together* as a pair. The model outputs a relevance score (0 to 1).
  - *Architecture:* Initial Retrieval (Bi-Encoder/BM25) $\rightarrow$ Top 50 Docs $\rightarrow$ Cross-Encoder $\rightarrow$ Top 5 Docs.
  - *Pros:* Higher accuracy than Bi-Encoders because the model sees the interaction between query and document terms.
- **Pattern C: Hybrid Search (The Gold Standard)**
  - *Mechanism:* Run **Sparse Retrieval** (BM25 for keywords) AND **Dense Retrieval** (Vectors for meaning) in parallel.
  - *Architecture:* Merge the results using Reciprocal Rank Fusion (RRF) to get the best of both worlds.

---

2. Why is it important to have very good search?

- **Conversion/Retention:** In e-commerce, if a user can't find a product, they can't buy it. Bad search directly correlates to lost revenue.
- **Knowledge Worker Productivity:** Employees spend ~20% of their time just looking for information. Good search drastically reduces this wasted time.
- **Trust (RAG):** In LLM applications, the "Generation" is only as good as the "Retrieval." If the search retrieves irrelevant documents, the LLM will hallucinate or fail.

---

3. How can you achieve efficient and accurate search results in large-scale datasets?

- **Hierarchical Indexing:** Use **HNSW** (Hierarchical Navigable Small Worlds) graphs. This allows logarithmic time complexity $O(\log N)$ search instead of linear scan.
- **Two-Stage Retrieval:**
  1. **Stage 1 (Recall):** Retrieve 1,000 candidates using a fast, approximate index (HNSW/IVF).
  2. **Stage 2 (Precision):** Re-rank the top 50 using a heavy Cross-Encoder model.

- **Quantization:** Compress vectors (e.g., from `float32` to `int8`). This reduces memory usage by 4x and speeds up distance calculations, allowing you to fit billions of vectors in RAM.

---

4. Scenario: Improving a Failing RAG Retrieval System

Problem: The RAG system is retrieving irrelevant chunks.

Steps to Improve:

1. **Analyze Failures:** Look at the bad queries. Is it missing keywords (Product ID "X-500") or missing semantic meaning?
2. **Hybrid Search:** If it misses keywords, add **BM25** (keyword search) alongside the vector search.
3. **Chunking Strategy:** Is the chunk size too small (missing context) or too big (too much noise)? Experiment with **Parent-Child Retrieval** (search small chunks, retrieve the parent document).
4. **Query Expansion:** Use an LLM to rewrite the user's query.
   - *User:* "It's slow." $\rightarrow$ *Rewrite:* "Why is the dashboard loading latency high?"
5. **Re-Ranking:** Implement a Cross-Encoder (like `bge-reranker`) to filter the top 50 retrieved results before sending them to the LLM.

---

5. Explain the keyword-based retrieval method.

- **Mechanism:** It matches exact words in the query to words in the documents. The industry standard is **BM25** (Best Matching 25).
- **How BM25 works:** It isn't just counting words (TF). It penalizes common words (IDF) and normalizes for document length.
  - *TF (Term Frequency):* How often does "apple" appear in this doc?
  - *IDF (Inverse Document Frequency):* Is "apple" rare across the whole database? (If yes, it's more important).
- **Pros:** Exact matching (great for names, IDs, error codes). Zero hallucinations.
- **Cons:** Fails at synonyms. "Car" will not match "Automobile."

---

6. How to fine-tune re-ranking models?

You don't fine-tune the vector database; you fine-tune the **Cross-Encoder**.

1. **Dataset:** Create pairs of `(Query, Document, Label)`.
   - *Label:* 1 (Relevant) or 0 (Irrelevant).
2. **Hard Negatives:** Mine "Hard Negatives"—documents that the base model *thought* were relevant (high score) but are actually wrong.
3. **Training:** Use a **Contrastive Loss** or **Cross-Entropy Loss** function. The model learns to output a score closer to 1 for the positive pair and 0 for the negative pair.
4. **Library:** Use `sentence-transformers` `CrossEncoder` class for easy implementation.

---

7. Most common metric in Information Retrieval and when it fails?

- **Metric: Recall@K** (Did the correct answer appear in the top K results?).
- **Failure Case:** Recall only cares *if* the answer is there, not *where* it is.
    - *Scenario:* If the correct answer is at Rank #10 (bottom of page 1), Recall@10 is 100%. But the user likely clicked the first result and left. Recall fails to measure the **ranking quality**.

---

8. Metric for Quora-like System (Pertinent Answers Quickly)

- **Choice: MRR (Mean Reciprocal Rank).**
- **Why:** You want the *best* answer to be at the very top (Rank 1).
    - If the answer is at Rank 1, Score = 1.
    - If at Rank 2, Score = 1/2 (0.5).
    - If at Rank 10, Score = 1/10 (0.1).
- MRR heavily penalizes the system if the "pertinent" answer drops even slightly down the list. This aligns with the goal of "finding answers quickly."

---

9. Metric for a Recommendation System?

- **Choice: NDCG (Normalized Discounted Cumulative Gain).**
- **Why:** Unlike search (where there is usually *one* right answer), recommendations have *multiple* good items with varying degrees of relevance (Perfect match, Good match, Okay match).
- **How it works:** NDCG gives credit for retrieving relevant items but "discounts" the credit if they are lower down the list. It handles graded relevance better than MRR or Recall.

---

10. Comparing IR Metrics

Metric	Focus	Use When...
Recall@K	"Is the answer in the list?"	You just need to find the document *somewhere* (e.g., Legal Discovery).
Precision@K	"Is the list mostly junk?"	You want to avoid showing bad results (e.g., Google Search first page).

MRR	"Is the answer at the top?"	There is **one** correct answer (Factoid QA, Quora).
NDCG	"Is the ranking order perfect?"	There are **multiple** relevant items with different quality levels (Netflix, E-commerce).

---

11. How does Hybrid Search work?

Hybrid search combines the strengths of **Keyword Search (BM25)** and **Semantic Search (Vectors)**.

1. **Run Sparse Search (BM25):** Finds documents with exact keywords (e.g., "Error 404").
2. **Run Dense Search (Vectors):** Finds documents with similar meaning (e.g., "Page not found").
3. **Normalization:** The scores from BM25 (e.g., 0 to 15) and Vectors (e.g., 0.6 to 0.9) are on different scales. You must normalize them (usually 0 to 1).
4. **Fusion:** Combine the two lists using a weighted sum `(0.7 * Vector) + (0.3 * BM25)` or RRF.

---

12. How would you merge and homogenize rankings from multiple methods?

- **Technique: Reciprocal Rank Fusion (RRF).**
- **Why:** It doesn't rely on the raw scores (which might be incompatible between models). It relies only on the **Rank**.
- Formula: For each document, calculate a score:
  $$\text{Score} = \sum \frac{1}{k + \text{rank}}$$
- **Logic:** If Doc A is Rank 1 in Keyword Search and Rank 1 in Vector Search, it gets a massive score. If Doc B is Rank 1 in Vector but Rank 100 in Keyword, its score drops significantly. This effectively "homogenizes" the lists without needing to calibrate the model scores.

---

13. How to handle multi-hop/multifaceted queries?

- **Problem:** Query: "Who was the CEO of the company that created the iPhone?"
  1. *Hop 1:* Who created the iPhone? $\rightarrow$ Apple.
  2. *Hop 2:* Who was the CEO of Apple? $\rightarrow$ Steve Jobs.
- **Technique: Query Decomposition.**
  1. Use an LLM to break the complex query into sub-questions.
  2. **Step 1:** Retrieve docs for "Who created iPhone?". Result: "Apple".

3. **Step 2:** Use the answer from Step 1 to generate a new query: "Who was the CEO of Apple?".
4. **Final:** Retrieve docs for Step 2 and synthesize the answer.

---

14. What are different techniques to be used to improved retrieval?

1. **Metadata Filtering:** Filter by `Year > 2023` *before* searching vectors.
2. **HyDE (Hypothetical Document Embeddings):** Generate a fake answer, embed *that*, and search.
3. **Query Expansion:** Add synonyms to the query (e.g., "Car" $\rightarrow$ "Car, Auto, Vehicle").
4. **Contextual Re-Ranking:** Use a Cross-Encoder to re-score the top results.
5. **Sentence Window Retrieval:** Retrieve a single sentence for the search match, but give the LLM the 5 sentences before and after it for context.