Chapter 3: Embeddings and Vector Search Questions Interviewers Ask

Cracking the RAG Interview

Embeddings and vector search are central to modern RAG systems, but interview questions usually go beyond basic definitions. You should be able to explain what embeddings mean, how similarity search works, where dense retrieval works well, where it fails and what changes when the embedding model changes.

1. What is an embedding?

Answer
An embedding is a way to represent text, images or other data as numbers in a vector space . In RAG , an embedding model receives a user question or document chunk and outputs a list of numbers , called a vector. The idea is that text with similar meaning should have similar vectors. For example: *"Glue job failed because the source schema changed."* and "ETL job failed because of a schema mismatch."

These sentences use different words, but they mean almost the same thing. Their embeddings will usually be close to each other.

During retrieval, the user question is also converted into an embedding. The system compares it with the embeddings of document chunks and finds the chunks with the closest meaning.

The embedding itself is not readable text. It is simply a numerical representation that helps the system compare meaning.

Possible follow-up questions

- What is an embedding ? What is a token ?
- What is the dimensionality of an embedding determined by?
- Can two unrelated pieces of text ever have similar embeddings?

2. Why do similar meanings appear close in vector space?

Answer
Embedding models are trained so that semantically related inputs produce similar vector representations. In training, the model learns relationships between words, phrases, sentences and concepts from massive amounts of data. For example, a user may search for: *"employee vacation policy"* But the document may contain: *"annual paid time off accrual"* The words are different, but the meaning is related. An embedding model can get this relationship and retrieve the correct chunk.

This is one of the main benefits of embedding based retrieval. The user does not need to use the exact same words that appear in the document. But embeddings are not perfect. They might fail for very specific technical terms, IDs, short queries, or words which can have multiple meanings.

Possible follow-up questions

- How is semantic similarity different from exact lexical matching?
- Why can domain-specific language cause problems?

- Would you use the same embedding model for every domain?

3. What is cosine similarity?

Answer
Cosine similarity is a way to measure how similar two embeddings are. In RAG, the user question is converted into an embedding and compared with the embeddings of document chunks. For example, suppose the user asks: *"Why did my Glue job fail?"* A document chunk says: *"ETL processing stopped because the source schema changed."*

If their embeddings are very similar, the cosine similarity score will be closer to 1. This tells the system that the two pieces of text are related in meaning.

A higher cosine similarity usually means the query and document are more closely related. A lower score means they are less related.

Many vector databases use cosine similarity to find the most relevant chunks. One important point is that a high similarity score does not mean the chunk definitely contains the correct answer. It only means the embedding model considers the query and the chunk similar in meaning.

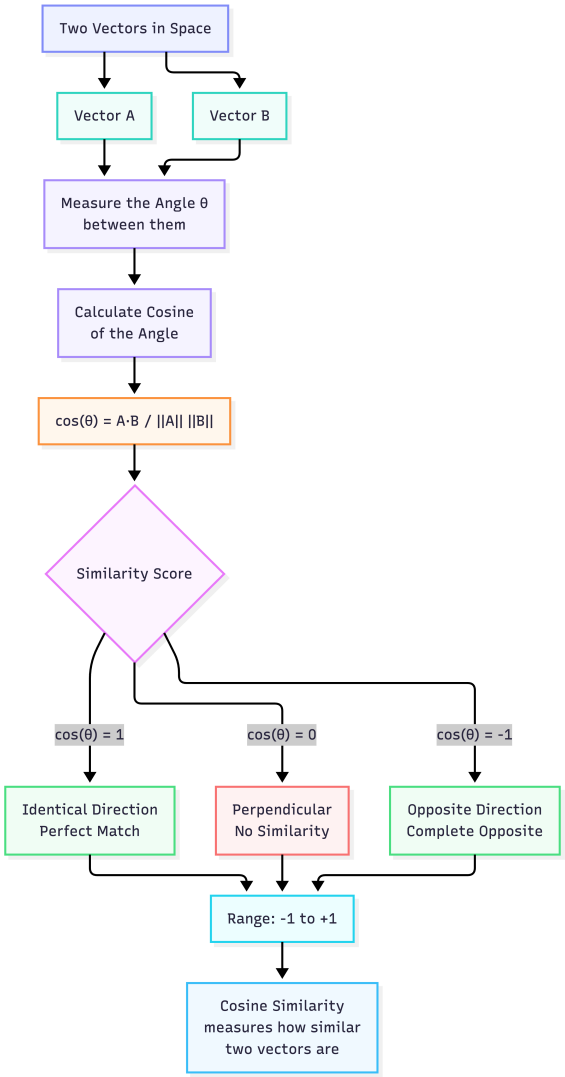

Fig: *Cosine Similarity*

Possible follow-up questions

- Cosine similarity vs dot product: what is the difference?
- Does a higher similarity score always mean a better answer?
- Can similarity scores be compared across different embedding models?

4. Why can dense retrieval miss exact error codes or identifiers?

Answer
Dense retrieval is capable of retrieving text with similar semantics, but it tends not to capture exact matching of words or IDs. For instance, when searching using: GLUE_ETL_ERROR_2027.

The embedding will tend to look for keywords like “Glue” and “error” and will be unable to return the exact document which has the error code.

This issue can also occur with ticket IDs, invoice numbers, product codes, or SKUs. For that reason, many RAG systems combine dense retrieval and BM25. Dense retrieval is used for semantically similar information. BM25 is used to precisely match words, codes, and rare terms.

The combination is known as hybrid retrieval. It provides the system with both semantic search and precise keyword search capabilities.

Possible follow-up questions

- Would increasing Top-K solve this problem?
- Why does BM25 work well for rare identifiers?

Answer:

BM25 gives more weight to words that are rare in the corpus . **This weighting is known as IDF or inverse document frequency** . Words like " error " or " job " occur in thousands of documents so BM25 treats these as low-signal .

A rare word like GLUE_ETL_ERROR_2027 may appear in only one document, so BM25 treats it as very high signal. When the query contains this rare word, its high IDF dominates the score. That one document rises straight to the top. There is nothing to dilute it.

Dense retrieval works differently. It averages the meaning of the whole query, so a rare identifier gets mixed in with the surrounding common words and loses its edge. This is why BM25 and dense retrieval are often used together. Dense retrieval handles meaning and paraphrases. BM25, through IDF, catches the exact rare word that dense retrieval would miss.

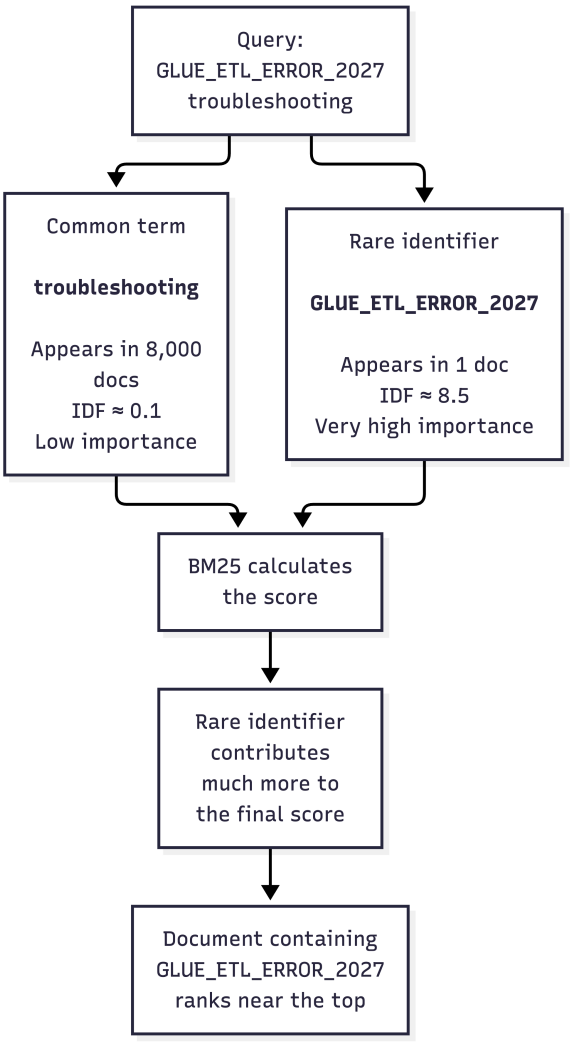

Fig: *IDF weighting in BM25*

- When would exact matching be more important than semantic matching?

5. What happens if you change the embedding model?

Answer

Changing the embedding model while keeping the input documents constant will result in changed output since each embedding model works differently in converting the text to vectors. Therefore, changing the embedding models will result in differences in vectors, similarity scores, and ranking. For instance, for the query: *"Glue job failed due to schema mismatch."*

With the old embedding model, the correct troubleshooting document may appear as result number 1. After changing the embedding model, the same document may appear as result number 4, while other documents move above it. Because of this, changing the embedding model usually means creating new embeddings for all existing document chunks.

We should carry out tests on the new model before adopting it. We should check whether the output document is relevant and how expensive it is in terms of performance, storage, and costs.

So, an embedding model should not be changed just because a newer model is available. It should be tested carefully to make sure it actually improves retrieval quality.

Possible follow-up questions

- Do you need to re-embed the entire corpus?
- How would you compare two embedding models?
- Can you perform a zero-downtime embedding-model migration?

6. Do query and document embeddings have to use the same model?

Answer

In a standard dense retrieval setup, query and document embeddings must live in a compatible vector space. The simplest and most common approach is to use the same embedding model for both.

Some retrieval models are explicitly trained with different encoders or different instructions for queries and documents, but those representations are still designed to be compatible with each other. You should not normally embed documents with one unrelated model and queries with another unrelated model because the resulting vectors are not directly comparable.

This is also why embedding-model versioning matters. If part of the corpus was indexed with one model and another part with an incompatible model, similarity search can become meaningless or inconsistent.

Possible follow-up questions

- What are asymmetric query and document encoders?
- What happens if only half of the corpus is re-embedded?
- How would you track embedding-model versions in metadata?

7. How would you migrate an embedding model in production?

Answer

A safe migration would not replace the existing index in place until the new model has been validated. A practical approach would be to create a second index with the new embedding model. Re-embed the corpus into that index, then run the same evaluation queries against both versions and compare retrieval quality, latency, cost and storage. If the new index performs well, route a small percentage of production traffic to it or run it in shadow mode before switching fully.

The migration flow can look like:

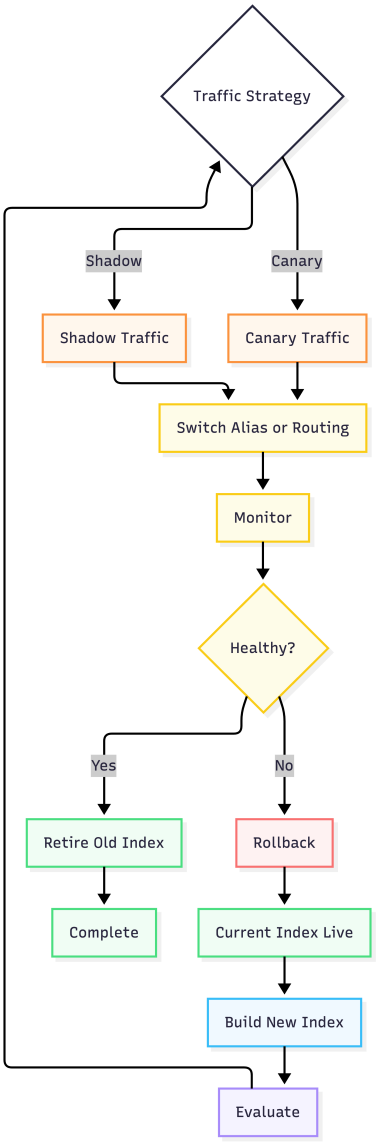

Fig: *Migration flow*

For large corpora, re-embedding may be incremental or distributed, but the principle remains the same: preserve a rollback path until the new retrieval behavior is proven.

Possible follow-up questions

- How would you migrate 100 million vectors?

Answer:

We should not replace the live index in place. The safe pattern is to build a second index alongside the first one. Start by creating a new empty index using the new embedding model. Re-embed the corpus in batches, usually 10,000 to 100,000 vectors at a time. Use checkpointing so a failed batch can be retried without redoing the work already done.

Once the new index is built, run it in shadow mode. Every production query still goes to the old index, and the user sees that result. In parallel, the same query also hits the new index and its result is logged. Comparing the two lets us measure whether the new index is actually better on Recall@K, MRR, and latency.

If the new index performs well, shift traffic gradually. A common pattern is 1 percent, 10 percent, 50 percent, and then 100 percent. Keep the old index warm for some time so we can roll back if something goes wrong.

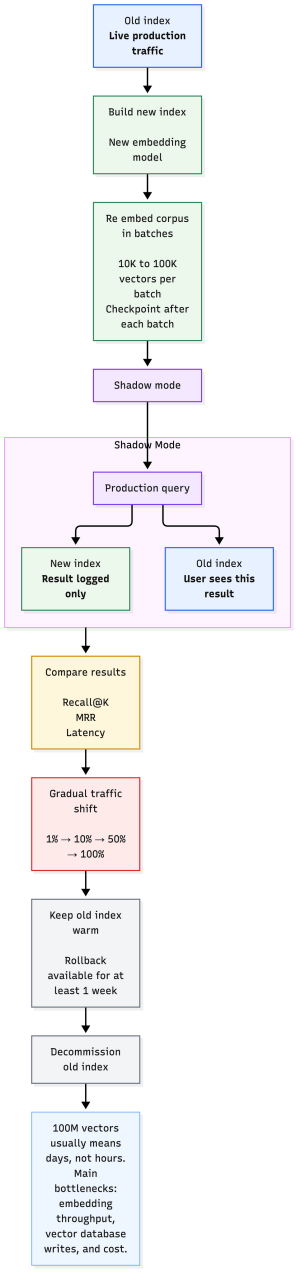

Fig: *Shadow-mode migration for a vector index*

- What metrics would you compare before cutover?
- How would you roll back if retrieval quality drops?

8. When is vector search the wrong retrieval strategy?

Answer

Vector search is useful when we want to find information with similar meaning, even when the user and the

document use different words. For example, a user may search for: *“employee vacation rules”* while the document says: *“paid time off policy”*

Vector search can understand that these are related. But vector search is not always the best option. It can struggle when the query contains exact values such as error codes, product IDs, policy numbers, names, or version numbers. In those cases, keyword search such as BM25 may work better.

Vector search may also be unnecessary for structured data. For example, if the user asks:

“What is the status of order 12345?”

it is better to look up order 12345 directly in a database or API instead of using vector search.

So, a good RAG system does not rely only on vector search. It uses the right retrieval method for the type of question, such as vector search, BM25, metadata filters, databases, APIs, or reranking.

Possible follow-up questions

- When would you use BM25 only?
- When should a RAG system call a database or API instead of a vector store?
- How would you route different query types to different retrievers?

Chapter 3 Recap

Embeddings make semantic retrieval possible by representing queries and document chunks in a comparable vector space. They are powerful when the same idea is expressed using different words, but they are not a universal search mechanism. Exact identifiers, structured lookups, model migrations, and domain-specific language all require additional design choices. In an interview, the strongest answer is rarely 'use a vector database.' It is to explain when vector search helps, where it can fail, and what retrieval strategy you would use instead or alongside it.