Data science Interview Questions

Q1. What is difference between Predictive/ Discriminative AI and generative AI?

Discriminative model focuses on predicting the likelihood & probability of a particular class from the user's input.

Generative models → generate new instance of data similar to training data.

→ maximizes the conditional probability $P(y|x)$

→ maximizes the joint probability $P(x, y)$.

→ if you have less data then go for GenAI.

→ this is creative tool. Can generate new data from training data.

Q2. What is LLM and how LLMs are trained?

LLM → pretrained on large corpus & then finetuned for specific tasks.

Step 1: pretraining

Step 2: finetuning

Step 3: RLHF & reward model.

Q2. What is token in language model?

When preprocessing text, the text is divided into the smaller units, & these smaller units are called tokens.

1 token = 4 characters or 0.75 words

Text Segmentation methods:

(1) Space-based tokenization:
"I am Abhisek" → ["I", "am", "Abhisek"]

(2) Dictionary-Based Tokenization:
Split based on predefined dictionary.

(3) Byte Pair Encoding (BPE) Tokenization:
Split based on the number of bytes, often using 2-byte or 3-byte.

Types of tokens:

(1) word tokens
(2) Sub-word tokens.
(3) Special tokens
[CLS] → classification token
[SEP] → separator token
[MASK] → mask token.
(4) Punctuation Tokens:
[".", ","]

⑤ Out-of-Vocabulary Tokens:

"[UNK]" → Unknown

⑥ Padding Tokens:

"[PAD]" → used for text alignment

⑦ [SOS] → Sentence start tokens.

⑧ [EOS] → End of sentence tokens.

Q4. Explain temperature parameter & how to set it?

It influences the randomness, creativity & quality of generated text.

It adjust the probabilities of the predicted words in the Softmax output layer of the model.

It scales the logit values before Softmax fn applies.

It has range between 0.1 to 2.

temp < 1.0 ⇒ output more deterministic & repetative.

temp > 1.0 ⇒ likely to select less probable words as next word. more creativity & variability.

temp = 1.0 ⇒ default settings. balances between randomness & determinism

Q5. what are some decoding strategies for picking output token?

primarily methods:—

(1) Greedy decoding method:

max probability is chosen as the next output token.

(2) Random sampling methods:

⇒ top K

⇒ top P

⇒ temperature

(3) Beam search:

Q6. what are different kind of stopping criteria in LLM?

```
import torch
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load model and tokenizer
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Encode input text
input_text = "Once upon a time"
input_ids = tokenizer.encode(input_text, return_tensors='pt')

# Set stopping criteria
max_length = 100
eos_token_id = tokenizer.eos_token_id

# Generate text
output_ids = model.generate(input_ids, max_length=max_length, eos_token_id=eos_token_id)

# Decode and print the output
output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
print(output_text)
```

max_length ensures the text generation stops after 100 tokens.

eos_token_id stops the generation when the end-of-sequence token is encountered.

Huggingface Stopping criteria API.

```
from transformers import StoppingCriteria
from torch import LongTensor, FloatTensor, eq, device

device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

stop_list = [" \n\nQuestion:", " \nHuman:", " \n\n", ]
stop_token_ids = [tokenizer(x, return_tensors='pt',
add_special_tokens=False)['input_ids'] for x in stop_list]
stop_token_ids = [LongTensor(x).to(device) for x in stop_token_ids]

class StopOnTokens(StoppingCriteria):
    def __call__(self, input_ids: LongTensor, scores: FloatTensor,
**kwargs) -> bool:
        for stop_ids in stop_token_ids:
            if (input_ids[0][-len(stop_ids[0])+1:] == stop_ids[0]
[1:]).all():
                return True
        return False
```

Finetune with PEFT !!

If the model fails to generate the EOS token, why not consider instructing it to do so? The concept of enhancing the model's performance by fine-tuning it with a dataset that includes answers concluding with the EOS token is certainly a promising avenue to explore.

Q7. How to use stop sequence in LLM?

Stop Sequences

Stop sequences tell the model when to cease output generation, which allows you to control content length and structure. If you are prompting the AI to write an email, setting "Best regards," or "Sincerely," as the stop sequence ensures the model stops before the closing salutation, which keeps the email short and to the point. Stop sequences are useful for output that you expect to come out in a structured format such as an email, a numbered list, or dialogue.

Q8. Explain Basic Structure of prompt Engineering.

Prompt Engineering is the practice of designing & refining specific text prompts to guide transformer based language models such as LLM to generate desired output.

Task	Example Prompt	Possible Output
Text Summarization	Explain antibiotics.	Antibiotics are medications used to treat bacterial infections...
Information Extraction	Mention the large language model based product mentioned in the paragraph.	The large language model based product mentioned in the paragraph is ChatGPT.
Question Answering	Answer the question based on the context below.	It was approved to help prevent organ rejection after kidney transplants.
Text Classification	Classify the text into neutral, negative, or positive.	Neutral
Conversation	The following is a conversation with an AI research assistant.	Black holes are regions of spacetime where gravity is extremely strong...
Code Generation	Ask the user for their name and say "Hello."	let name = prompt("What is your name?"); console.log(`Hello, ${name}!`);
Reasoning	The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.	No, the odd numbers in this group add up to an odd number: 119.

techniques:

1. Zero-shot
2. one-shot
3. few-shot
4. COT.

Q9. Explain in-context learning?

method of prompt engineering where the model is shown task demonstration as a part of the prompt in natural language. Using ICL, you can utilize pre-trained LLM to solve tasks without finetuning.

How to teach LLM without retraining it from scratch? ⟶ In context learning.

It's like giving cheat sheet that helps it understand what you want it to do.

→ Zero-shot learning.
→ one-shot learning.
→ few shot learning.

Q10. What are some of the aspect to keep in mind while using few-shots prompting?

(1) Quality of examples:
ensures highly relevent to the task.
use a diverse set of examples that covers various aspects of the task.

(2) Number of examples:
typically 3-5.
too small is not sufficient.
too many are leads to longer processing time.

Q11. What are certain strategies to write good prompt?

→ clearly define what you want to achieve with your prompt. is it information, creativity or problem-solving.
→ avoid complex prompt.
→ avoid unnecessary information rather provide enough background for AI to understand the scenario.

→ refine your prompt based on the responses.

Q12. What is hallucination & how can it be controlled using prompt engineering?

"hallucination" → also called "make things up"

Common failure for LLM.

reason behind hallucination:

→ the model try to fabricate information when it's unsure of real answer.

→ It is not trained to avoid false information.

following methods are useful to reduce hallucination.

(1) Retrival Augmented Generation (RAG).

(2) ReAct prompting.

(3) Data filtering & cleaning.

(4) fine tune on domain specific data.

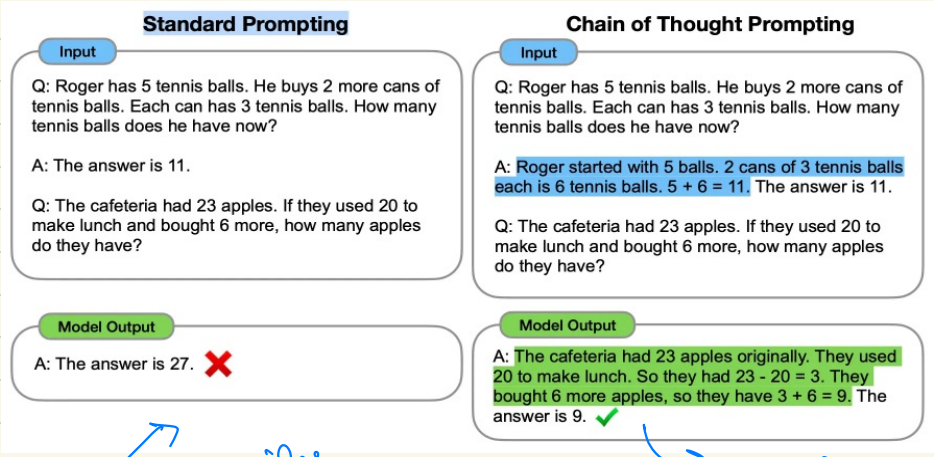
two types of hallucinations

1. Intentional → when someone inject data to your vector (cyber security)
2. Unintentional → models are trained with unlabeled data.

It could be because of no new data, model is not trained with updated data.

Q13. How to improve reasoning ability of my LLM through prompt engineering?

Prompting method that encourage LLMs to explain their reasoning.



this only provides response

this gives response with reasoning.

Limitation of CoT.

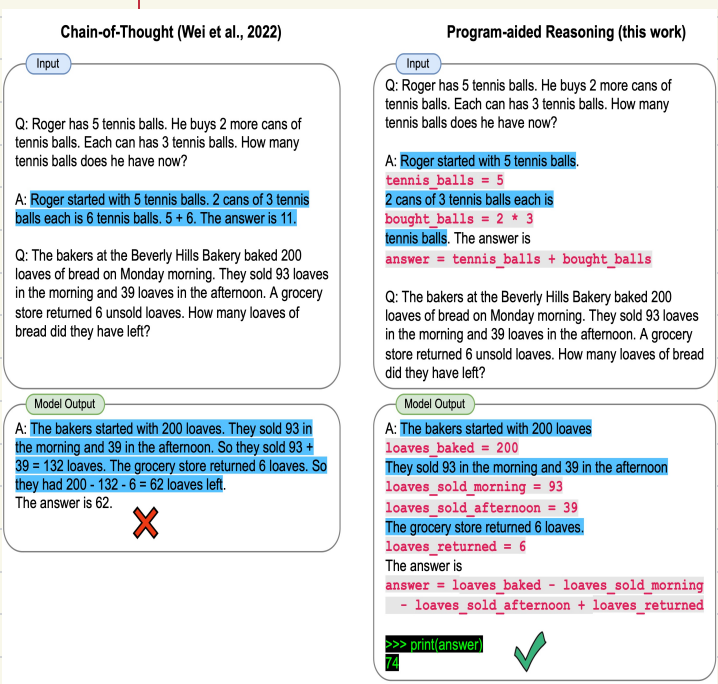
"CoT only yields good performance gain with models ~100B parameters. But small model wrote illogical chain of thought which is worser than standard prompting".

Q14. How to improve LLM reasoning if your CoT fails.

Due to limited math skill LLM may fail to do math correctly.

If you need accurate calculation then we would go for —

PAL (program-aided language models)



Program-Aided Language models (PAL): a new method that uses the LLM to read natural language problems and generate programs as the intermediate reasoning steps, but offloads the solution step to a programmatic runtime such as a Python interpreter.

With PAL, decomposing the natural language problem into runnable steps remains the only learning task for the LLM, while solving is delegated to the interpreter.

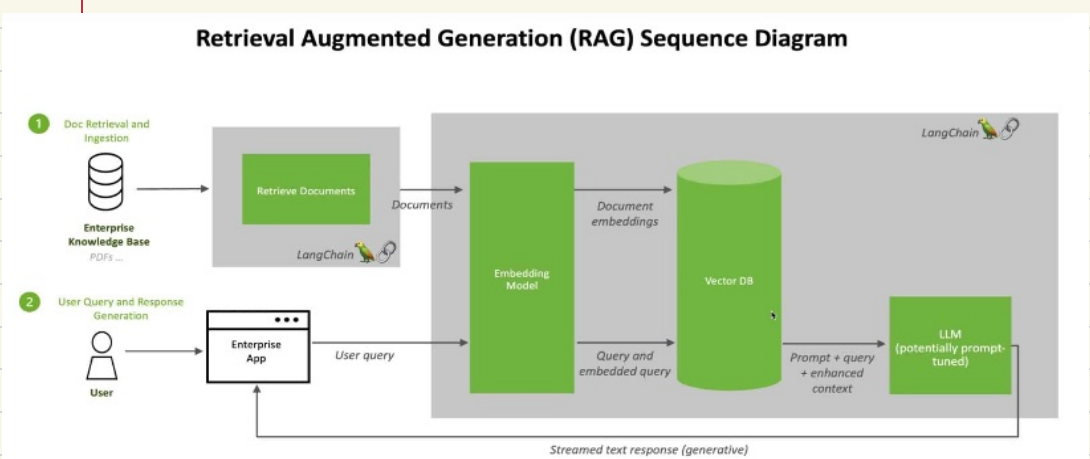
Q15. How to increase accuracy & reliability & make answer variable in LLM.

Use RAG.

Q16. What is RAG?

It is a technique for enhancing the accuracy & reliability of genAI models with facts fetched from external sources.

Q17. How RAG works?



https://blog.langchain.dev/tutorial-chatgpt-over-your-data/

RAG improves & enriches LLM's response.

criticism of LLMs →

- → Lack of transparency into the information source
- → reference to only static data sources.
- → limited context windows.
- → tendency towards hallucination.

RAG addresses all these concerns.

Q18. Benefits of implementing RAG:

1. Enhance accuracy & factual grounding
   relies on verifiable & external source of information that reduces hallucination.
2. Improve contextual understanding.
3. Real time information access & Integration.
4. Explain transparency & explainability.

Q19 When should i use Fine-tuning instead of RAG?

RAG feeds realtime data to LLM

fine tune further training a pretraining model to adapt a particular task,

so, fine tuning is costly.

RAG provides more transparency which provides source documents also.

Q20 What is chunking & Why do we do this?

Improves information retrival.

By splitting text effectively, RAG can generate a more accurate response that is contextually appropriate.

Strategies for text chunking & splitting:

(1) fixed length:

Character text splitter.

(2) Semantic chunking:

Recursive character text splitter. This allows overlap between chunks & preserve context. that might be lost at the boundary of a chunk.

SemanticChunker.

(3) Advance splitting (Markdown header text splitter)

useful for XML, HTML.

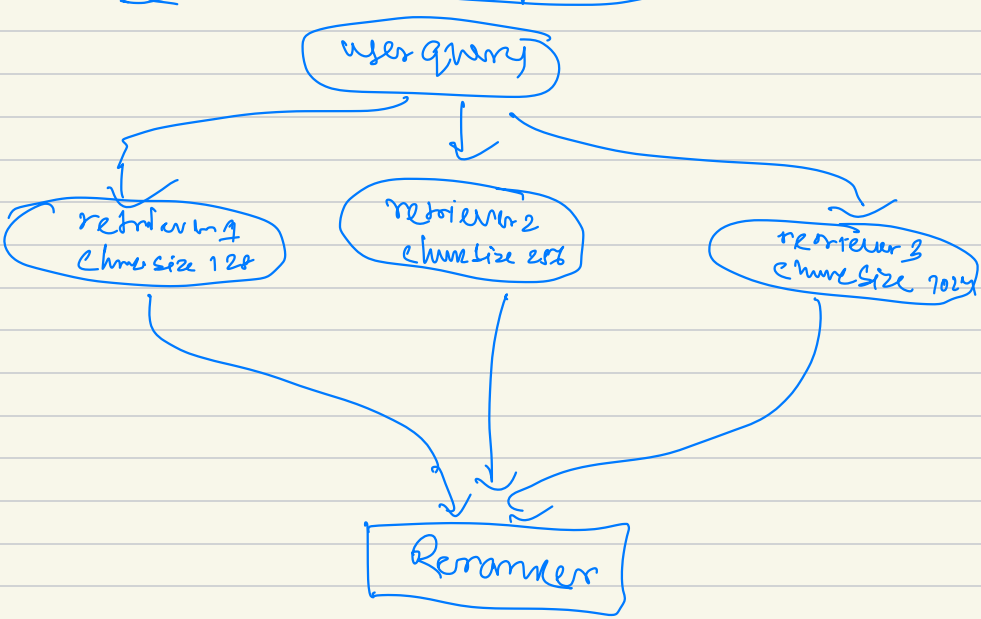
④ Split by token: tiktoken

Q21 What are the factors influences chunk size?

① Data characteristics:
for text doc → use average paragraph length.

② Memory & Computational resource:
Larger chunk may lead to higher memory usage. Balance the size with the available resources to ensure efficient processing.

** Retriver Ensambling & Reranking:



Q22 What is vector embeddings & what is embedding model?

Vector embeddings convert high dimension information like text or image into a structured vector space.

types of vector embeddings —

① Word embedding:
Word2vec, Glove

② Sentence & document embeddings:
represent larger pieces of text.
BERT, Doc2vec ✓

③ Semantic Embedding
ELMo

Which embedding model is good?

MTEB leaderboard on Huggingface.
↓
look at retrival average column.

Just embedding models:
- all-mpnet-base-v2
- all-MiniLM-L12-v2

Model Size
Max tokens
Embedding Dimensions

no of tokens compressed into single embeddings.
generally single paragraph

If input (no of tokens) → small dim → low inference latency.

Q23. What is difference in embeddings of short & long content?

Small Content:

→ limited content, captures local relationships within a small context window.

→ Word2vec, Glove

→ These embeddings focus on word meanings & syntactic relationships.

Long Content:

→ paragraphs/documents contain richer content.

→ Embeddings for long content consider global context & dependencies.

→ BERT, DOC2VEC

⇒ They handle co-reference, topic shifts & overall document structure.

Q24. How sentence transformer works? How to improve it?

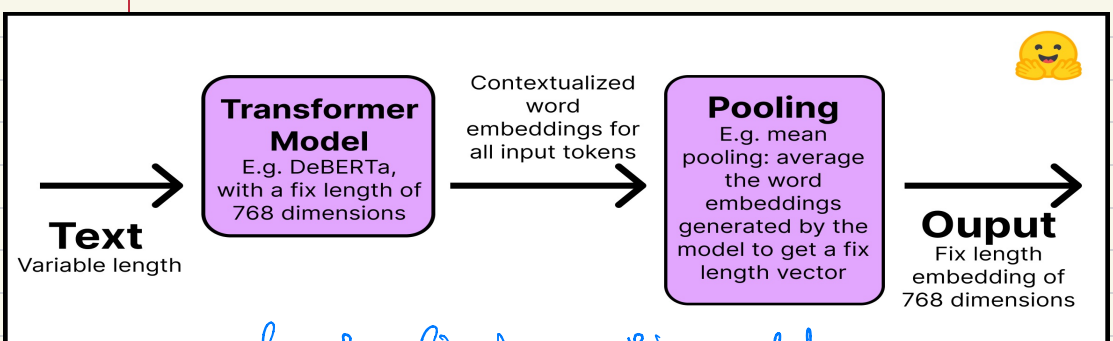
Sentence transformer went through layer

Layer 1 :

input text → Pretrained transformer model → embeddings for each word

Layers :

→ Pooling Layer → output fix length embedding of 768 dim.



You can further finetune this model on your own data. / The dataset configuration can be any of below forms —

- Case 1: The example is a pair of sentences and a label indicating how similar they are. The label can be either an integer or a float. This case applies to datasets originally prepared for Natural Language Inference (NLI), since they contain pairs of sentences with a label indicating whether they infer each other or not.
- Case 2: The example is a pair of positive (similar) sentences without a label. For example, pairs of paraphrases, pairs of full texts and their summaries, pairs of duplicate questions, pairs of (query, response), or pairs of (source_language, target_language). Natural Language Inference datasets can also be formatted this way by pairing entailing sentences. Having your data in this format can be great since you can use the MultipleNegativesRankingLoss, one of the most used loss functions for Sentence Transformers models.
- Case 3: The example is a sentence with an integer label. This data format is easily converted by loss functions into three sentences (triplets) where the first is an "anchor", the second a "positive" of the same class as the anchor, and the third a "negative" of a different class. Each sentence has an integer label indicating the class to which it belongs.
- Case 4: The example is a triplet (anchor, positive, negative) without classes or labels for the sentences.

Q25 What is Vector DB?

Vector database is a collection of data stored as mathematical representations.

↳ Specialized for similarity searches and high dimensional data.

Q26 How it is different from relational data?

RDBMS commonly used for transaction data, business applications.

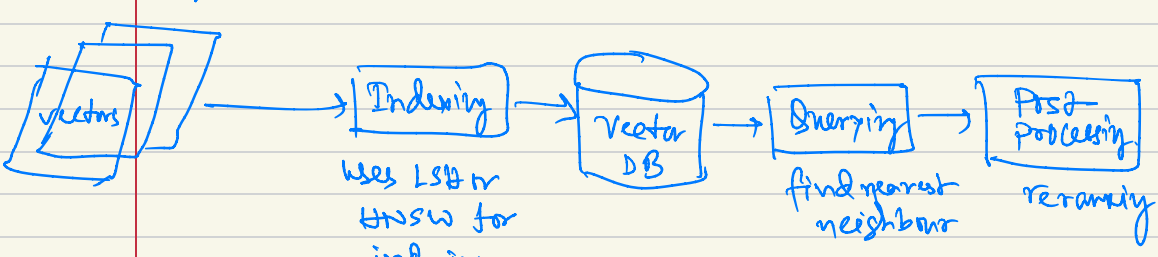
Indexing are different for both.

Q27. How does Vector DB works?

→ Basically does apply a similarity matric to find most similar vector matches our query.

→ It is a combination of different algorithms that all participates in Approximate Nearest Neighbor (ANN) search.

These algorithm optimizes the search using hashing, quantization or graph based search.



Indexing in vector DBs:

ANN offers two types of indexes (IVFFlat & HNSW)

IVFFLat: (Inverted File with Flat Compression)

https://www.timescale.com/blog/nearest-neighbor-indexes-what-are-ivfflat-indexes-in-pgvector-and-how-do-they-work/

1st step: apply K-means to vectors to find cluster centroids.

2nd step: map each vector to its nearest centroid.

```
inverted_index = {
  centroid_1: [vector_1, vector_2, ...],
  centroid_2: [vector_3, vector_4, ...],
  centroid_3: [vector_5, vector_6, ...],
  ...
}
```

pgvector offers three different methods to calculate the distance between vectors.

L2, inner product, cosine.

HNSW: (Hierarchical Navigable Small worlds)

Q28. what is different vector search strategies?

Lexical search → BM25
Semantic search → ANN.

Q29 Explain Locality-sensitivity hashing (LSH) index method?

most popular ANNs.

It's a hashing function that allows us to group similar items into same hash buckets.

** Unlike most hashing functions which aims to minimize hashing collisions — LSH algorithms aim to maximize hashing collisions.

Q30 what is Product Quantization (PQ)?

Moreover, in Postgres, indexes over large vectors are difficult to support. Indexes are stored in 8kB sized pages. The largest vector that can fit on one page is about 2000 dimensions. It is possible to support larger dimensions, but at the cost of increased complexity and reduced performance. As a result, Postgres vector search extensions like pgvector and Lantern currently only support up to 2000 dimensions for vector indexes.

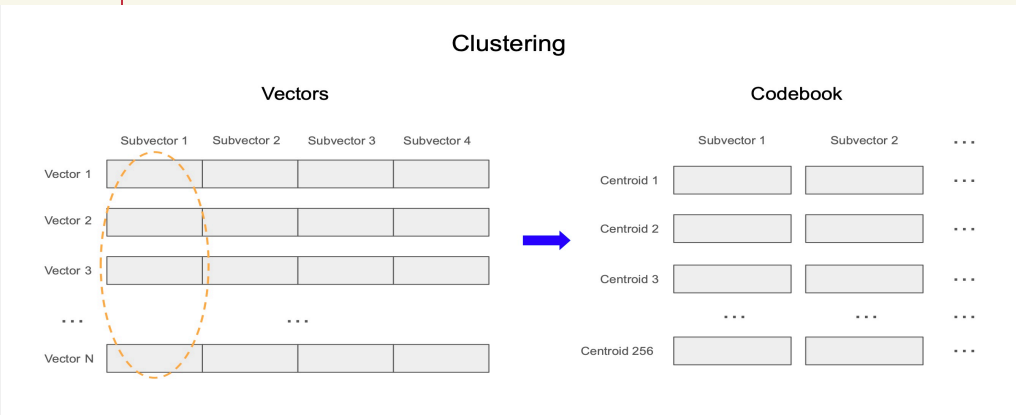
What is product quantization?

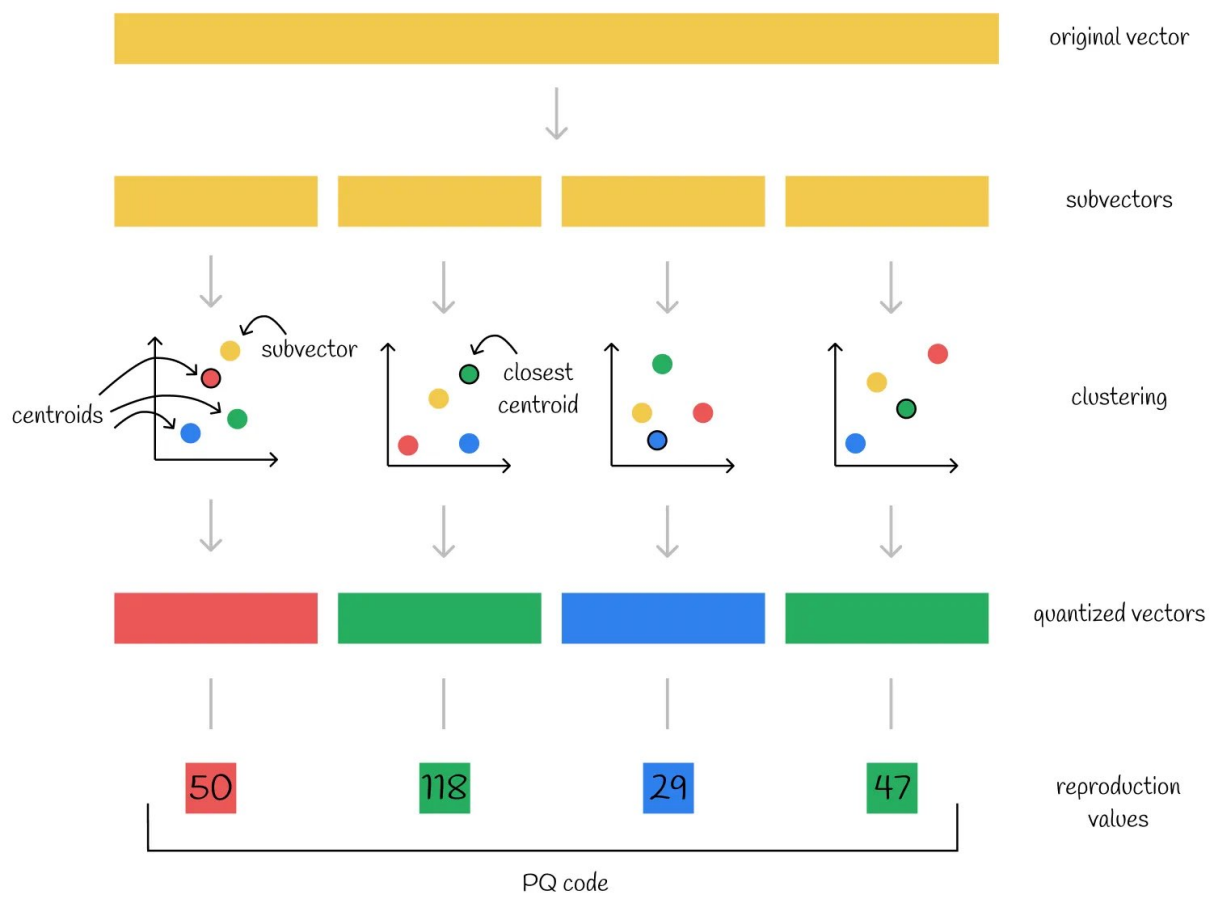
One way to address the index size problem is with compression. In this article, we'll focus on product quantization (PQ), which is a compression technique for high-dimensional vectors.

PQ works in a few steps:

1. Clustering: We divide each vector into equal-sized subvectors. For each set of subvectors, we run the k-means algorithm to identify centroids, or representative subvectors, for that subspace.
2. Quantization: For the stored vectors, we quantize each subvector by encoding it with the ID of the nearest centroid and store this quantized representation in a column.
3. Indexing: We use the quantized vectors to generate an HNSW index. Inside the index, we store the centroid IDs for each subvector.
4. Searching: When a query vector is submitted, it traverses the index structure by calculating the distance between the original query vector and the quantized database vectors.

The diagrams below depict an example of this process. In this example, we calculate 256 centroids for each subvector of length 32. This results in over 99% memory reduction.




original vector
subvectors
subvector
centroids
closest
centroid
clustering
quantized vectors
50
118
29
47
reproduction
values
PQ code

https://towardsdatascience.com/similarity-search-product-quantization-b2a1a6397701

Q31: Which VectorDB to choose ?

Choosing a vector database

The best vector database for you will depend on your specific needs and requirements. Consider the following factors when making your decision:

- The size and complexity of your dataset
- The performance requirements of your application
- The features that are important to you
- Your budget

If you are unsure which vector database to choose, you can try out a few different ones to see which one works best for you. There are a number of free trials available.

https://benchmark.vectorview.ai/vectordbs.html

Q32: Explain different types and challenges associated with filtering in vector DB?

```
vectorstore.similarity_search(
    "ducks",
    k=10,
    filter={
        "$and": [
            {"id": {"$in": [1, 5, 2, 9]}},
            {"location": {"$in": ["pond", "market"]}},
        ]
    },
)
```

```
vectorstore.similarity_search("kitty", k=10, filter={"id": {"$in": [1, 5, 2, 9]}})
```

Operator	Meaning/Category
\$eq	Equality (==)
\$ne	Inequality (!=)
\$lt	Less than (<)
\$lte	Less than or equal (<=)
\$gt	Greater than (>)
\$gte	Greater than or equal (>=)
\$in	Special Cased (in)
\$nin	Special Cased (not in)
\$between	Special Cased (between)
\$like	Text (like)
\$ilike	Text (case-insensitive like)
\$and	Logical (and)
\$or	Logical (or)

Q33. Why it is important to have very good search?

→ users looks for more up-to-date, accurate & efficient response.

→ Improves Customer satisfaction.

→ Contextual Relevance: Good search capabilities ensure that the retrieved documents are contextually relevant to the input query.