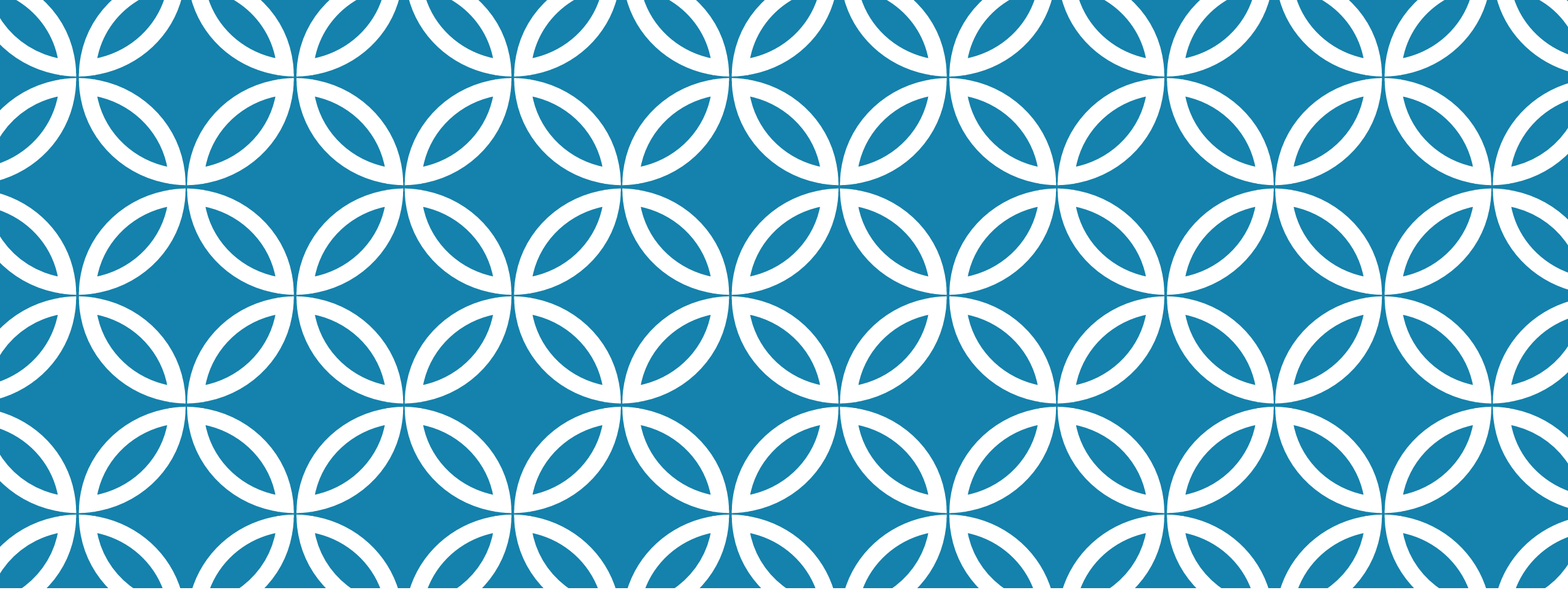


VECTOR DATABASES

CS4414/CS5416
Lecture 24

IDEA MAP FOR TODAY

Review the steps in creating a "knowledge database"

Filtering documents to exclude irrelevant ones

Chunking to identify important sections

Embedding in high-dimensional space

Building a vector index

Vector search

There exist about 170B unique books, 75M unique research papers, 4B web pages (but nearly 50T if we count past versions)…

Vector databases are a tool for fast search in **enormous collections of documents.**

ROLE OF A VECTOR DATABASE

Used to store data, usually some form of "document id" (the document itself is often in a separate KVS or file system)

Enables *search for approximate matches*, which will be our main topic today. These searches could be issued by humans, as a form of query, but can also be initiated by an ML such as an LLM or LRM or VLM.

Vector databases are key to modern ML yet somewhat hidden and not widely discussed. Many people assume the LLM memorizes everything. In truth, *an LLM often depends on a vector database for background info, hints, etc.*

WHAT MAKES IT A VECTOR DATABASE?

In addition to managing the documents (in a conventional way) these databases include an additional *index:* a fast-lookup data structure

The idea is to be able to find documents that are closely related to a query rapidly, but this raises a puzzle:

- A document can be about many things, only some of which connect to the query
- Some documents are much more relevant than others

WHAT MAKES THIS HARD?

As we mentioned, we might be dealing with hundreds of millions or even billions of documents.

Think about the index to a book – it might have thousands of items, and if you were searching for that topic you might want the definition, but you could be looking for how that technique is used in some other section

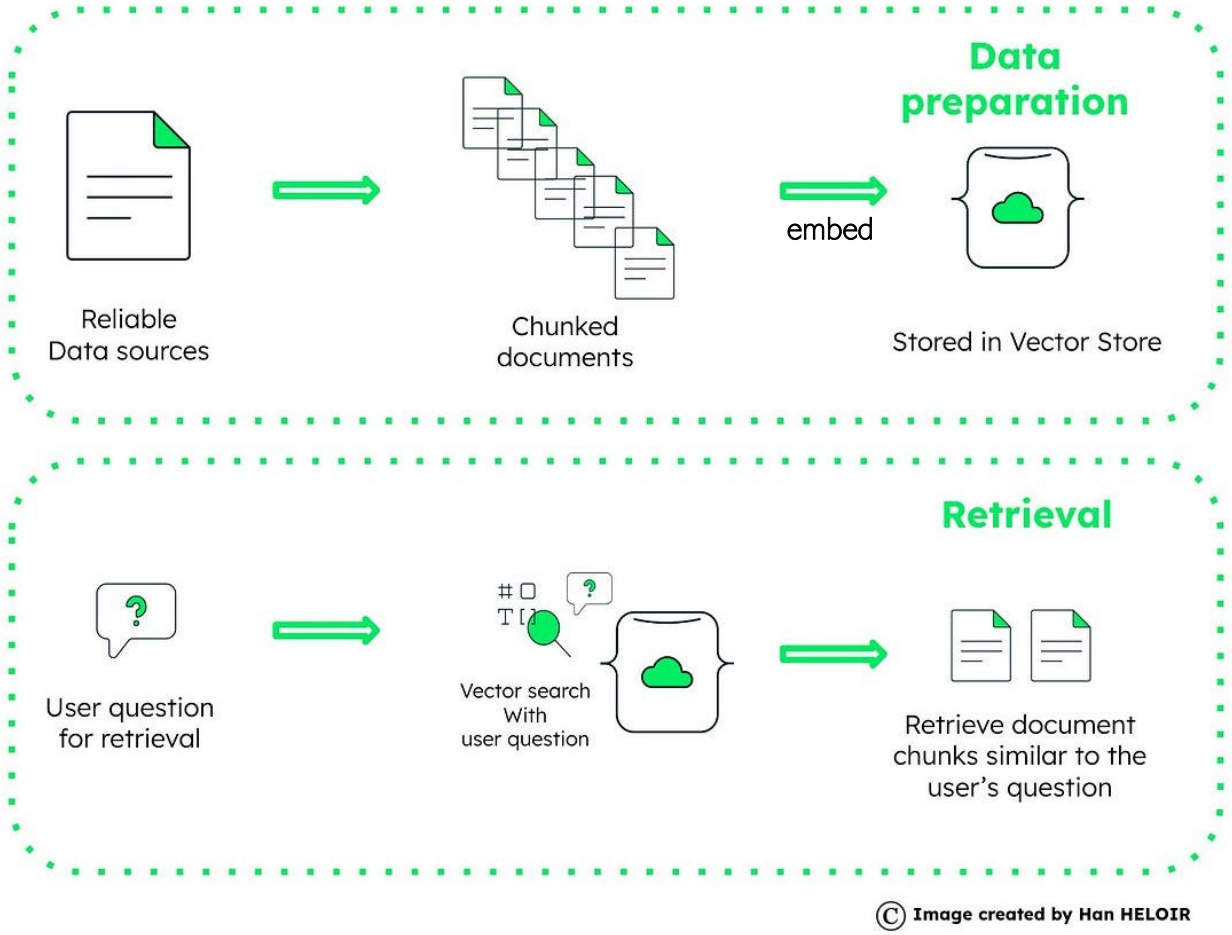
For an LLM to start with a query but then give you a useful quote from the book, it needs a way to find that book, page and quote!

EMBEDDING A DOCUMENT

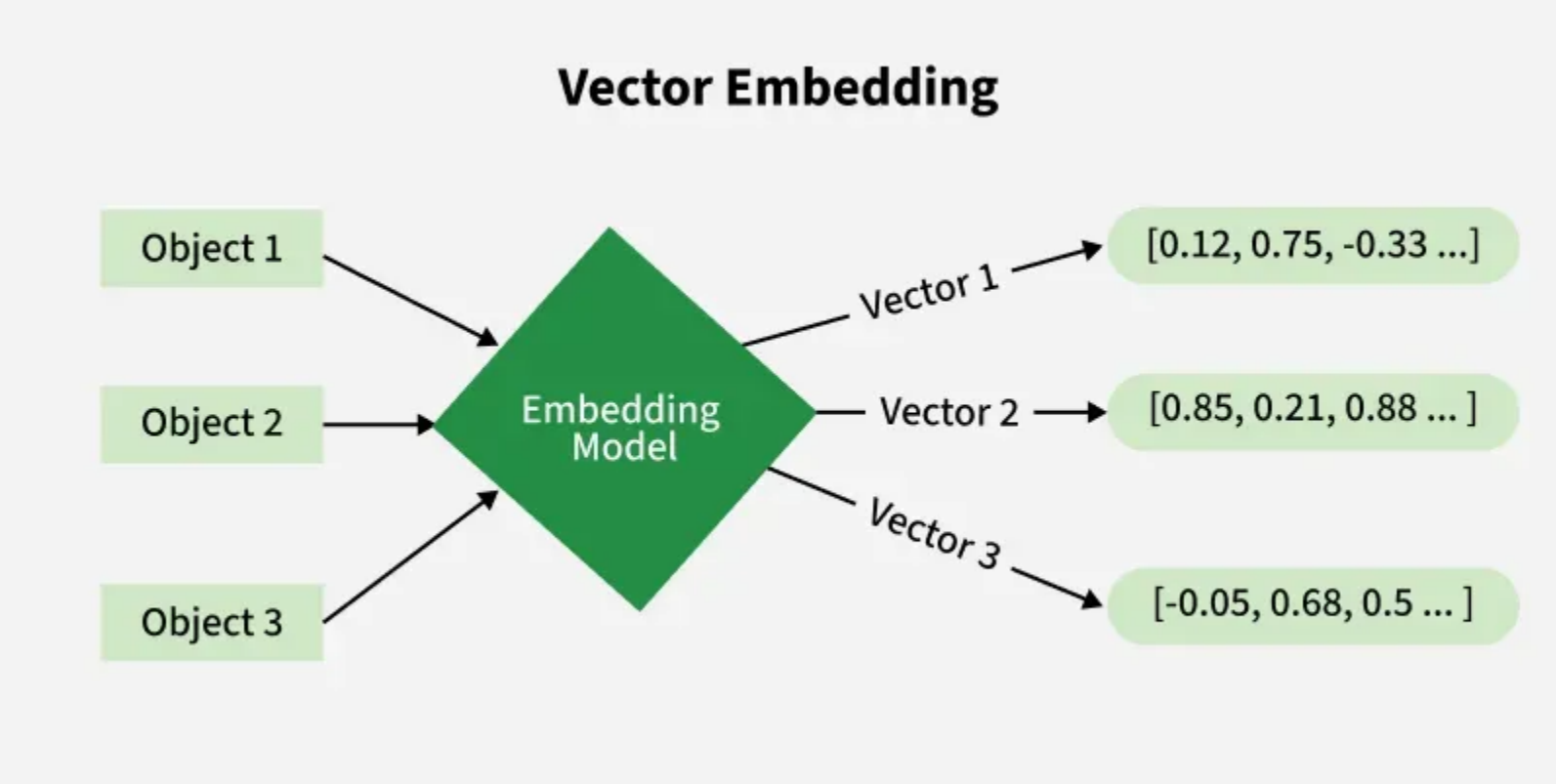
Over many years, the ML community arrived at a multi-step approach:

- First, **filter** the set of documents – if I'm asking for advice about stretching a strained muscle, the database itself shouldn't be full of media gossip about Prince Andrew and Fergie, or Corgie dogs.
- Next, "chunk" each document into high-quality snippets of text
- Then transform each chunk into an *embedding*, like in HW1 and HW3.
- … and these become the "vectors". They link to document ids or URLs

VISUALIZING THIS PROCESS



VISUALIZING EMBEDDING



ONE "DOCUMENT" COULD HAVE THOUSANDS OF EMBEDDINGS!

That medical textbook alone could have tens of thousands of important paragraphs spread over it!

And a medical vector database might have hundreds of books and tens of thousands of medical abstracts and research studies and drug information documents in it.

IMPORTANT PROPERTY

Embedding is a learned process, because it needs to put similar objects close together in the high dimensional "vector space"

Moreover, we can embed many kinds of objects into the identical space, such as text, images, categories, etc

This makes the embedding space fundamental (and very hard to design)! The big AI companies offer **chunking and embedding as a service**

THE EMBEDDING AND DNN GO TOGETHER

Embedding maps to a high dimensional representation space on which the DNN will later be trained

Even experts view the methodology as somewhat magical. It works and we do understand why it works, but at the scale of genuine LLMs, it all becomes very subtle and often, surprising.

SPECIALIZED TASKS OFTEN REQUIRE SPECIALIZED TERMINOLOGY OR PHRASING

Some applications use highly specialized terminology. For this reason we often need to *fine tune* a document chunking or embedding solution.

- It is much easier to fine tune a document chunking tool. We would use a technique like a low rank adapter (LoRA) to teach it how to recognize significant phrases or contextual important terms.
- Embedder solutions are strongly tied to the DNN we will later use for the LLM or LRM, and any changes could reduce accuracy for existing models, forcing whole model training: a hugely costly task!

INTERNAL STRUCTURE OF MANY LLMS

Often an LLM would produce a nonsense result if it doesn't understand the context of a query

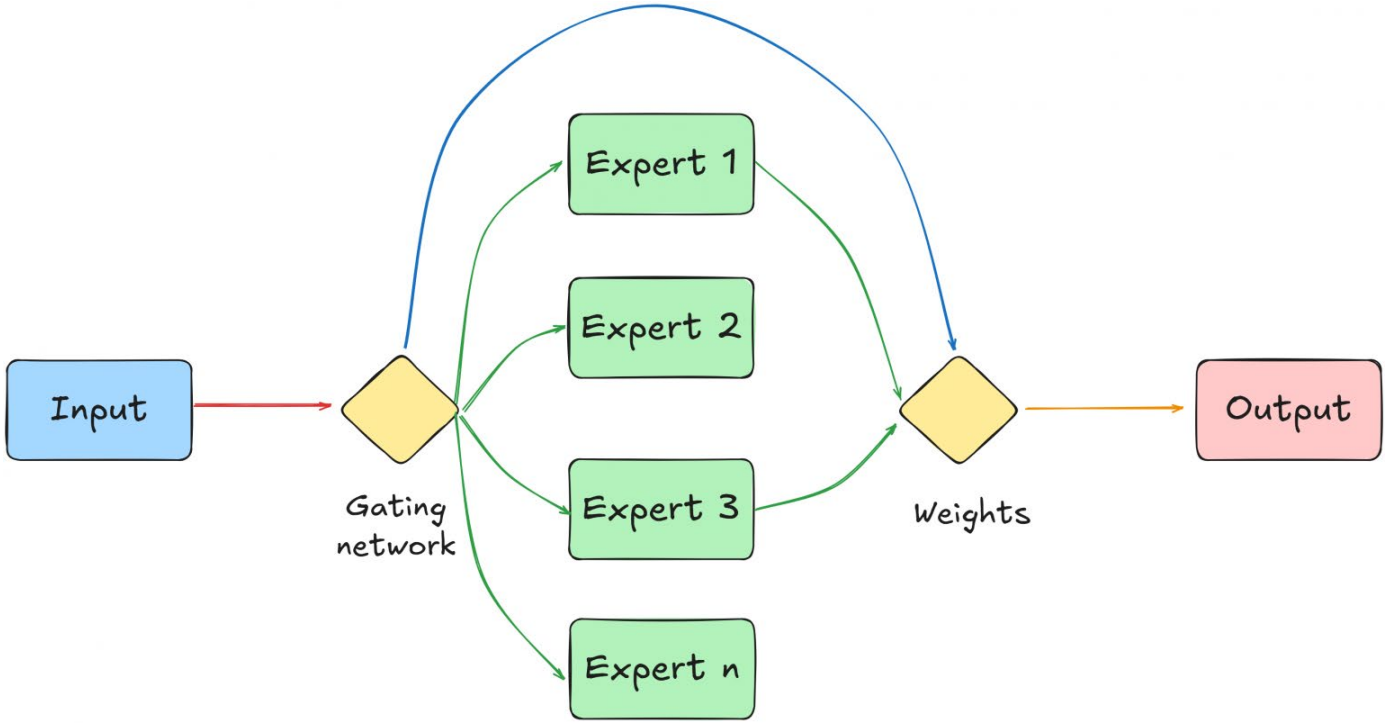
To guide it towards the desired result, we take the prompt or query and augment it with (1) history of the dialog, and (2) additional related content

This could be a context, a history of prior dialog with the client, and it could also be information pertinent to the query drawn from precollected data that has been indexed for convenient search.

MIXTURE OF EXPERTS

A mixture of experts (MoE) takes a further step and breaks the LLM task down into subtasks that can focus on different specialities

VISUALIZING A MIXTURE OF EXPERTS



WHAT ABOUT STEP BY STEP REASONING?

We might also design the LLM to tackle complex problems step by step

This yields an MoE that supports a **large reasoning model**: an *LRM*.

So, if your boss next year says you'll work on a new RAG MoE LRM, you'll know you should ask "So cool! Which vector database are we using?"

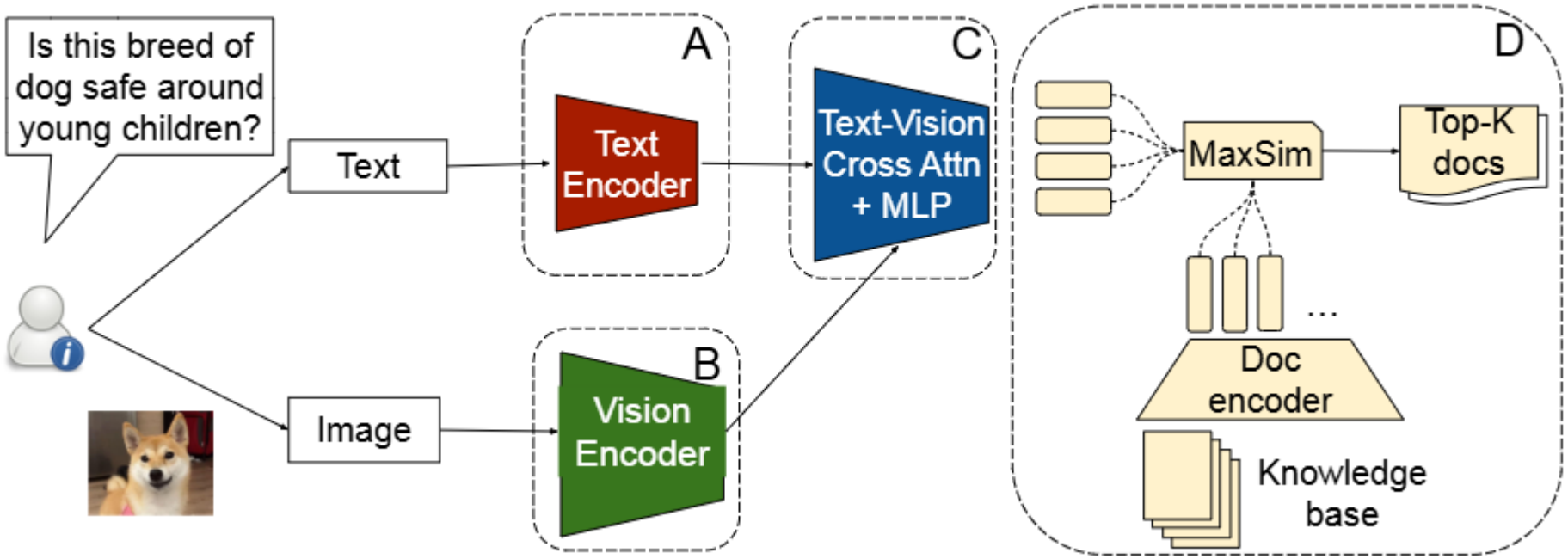
KNOWLEDGE RETRIEVAL

Consider this:

- Memory: "Client name = Anita Singh". Spouse = "Ranjan Singh". …
- Documents Anita authored (text, articles, powerpoint slides, etc)
- Photos and video she posted, often tagged with meta-data
- Home city, university transcripts, course syllabi…

… data about people, places and things needs to be searchable too!

A KNOWLEDGE RETRIEVAL ML SERVICE



PreFMLR knowledge retrieval, finds documents relevant to a query.

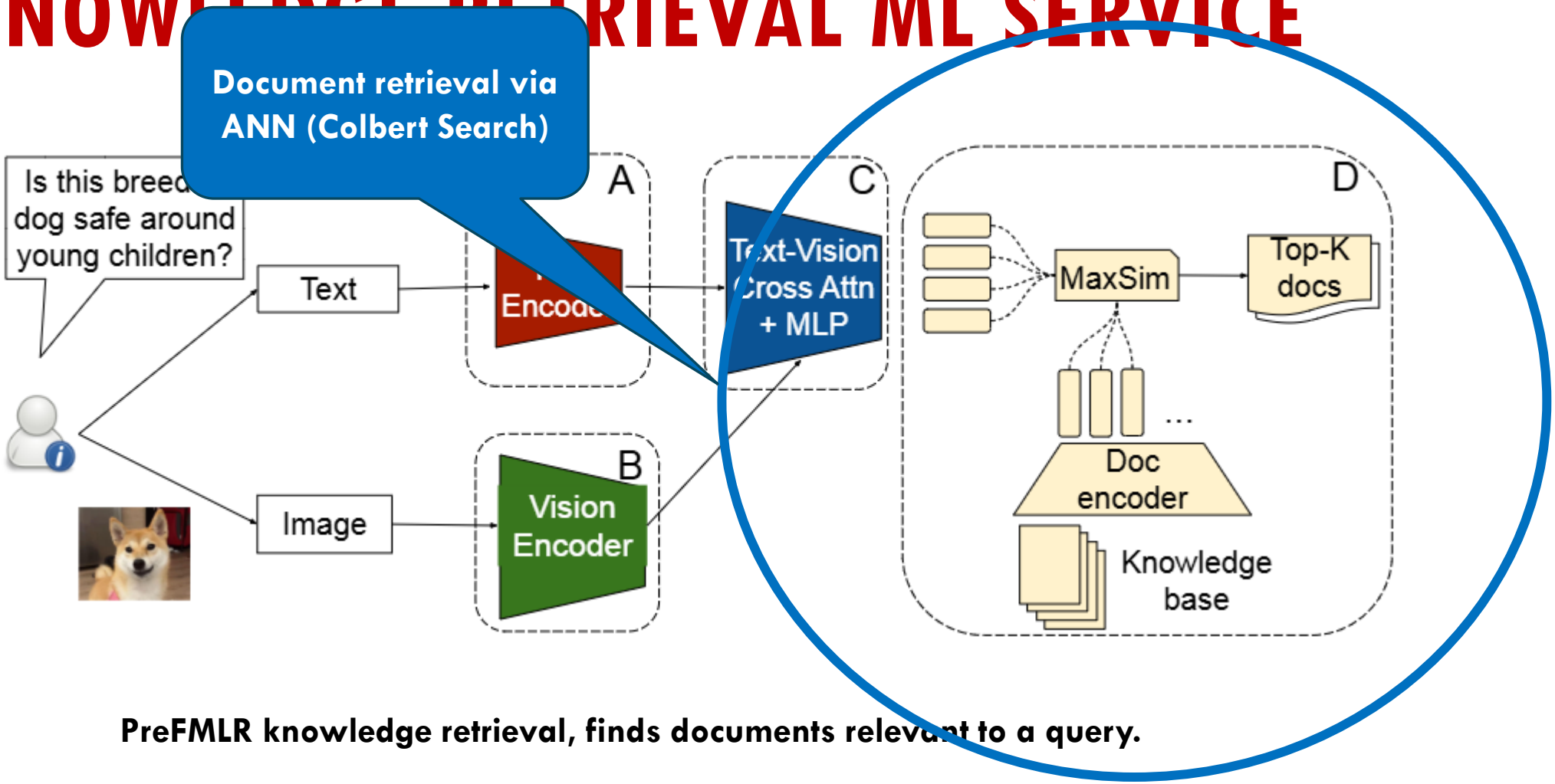
WHAT ARE THE ENCODERS AND CROSS-ATTENTION LAYERS DOING?

The encoders preprocess the input

But in this case the query is multimodal: it is a text query about the image

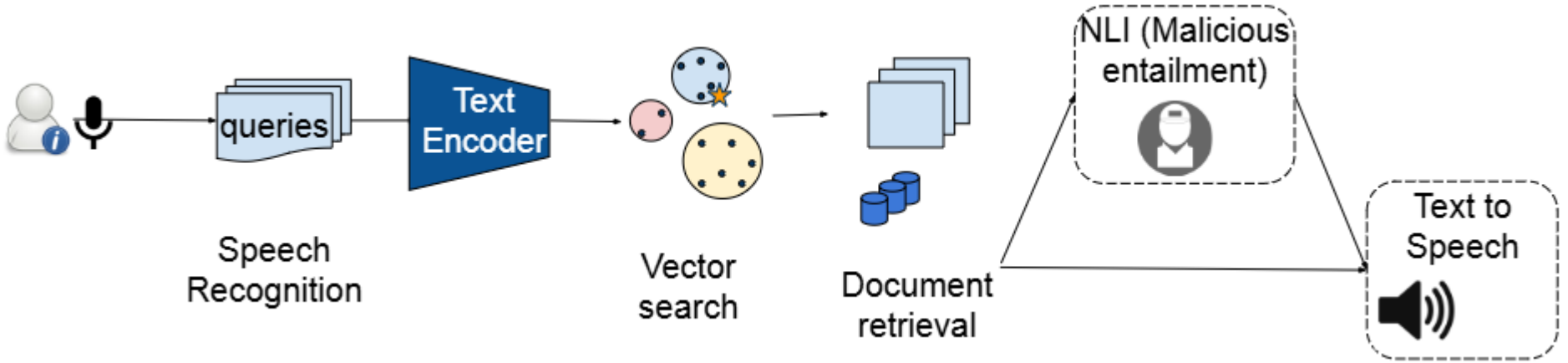
The cross-attention layer "focuses" the query on the relevant aspect of the image, after which embedding will produce a vector that captures the essential nature of the query

A KNOWLEDGE RETRIEVAL ML SERVICE



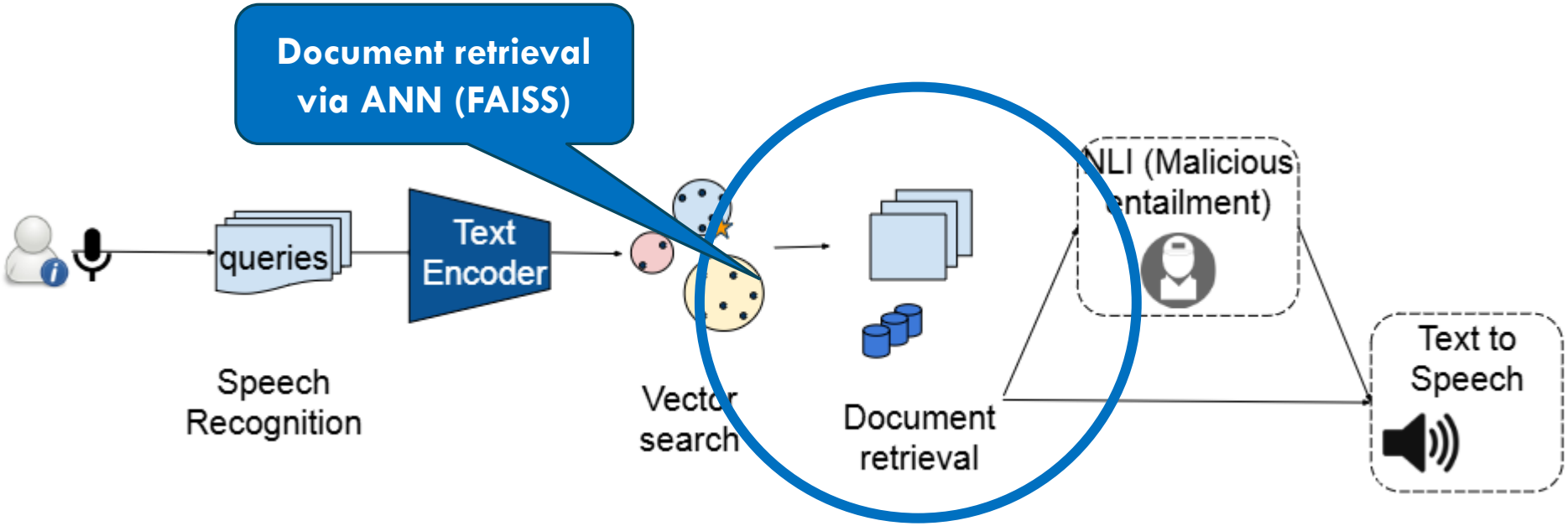
PreFMLR knowledge retrieval, finds documents relevant to a query.

A Q/A ML SERVICE



SpeechSense: audio Q/A for news feeds or document collections.

A Q/A ML SERVICE



SpeechSense: audio Q/A for news feeds or document collections.

WHY JUST ONE VECTOR DATABASE?

Today, one database often suffices: embedders give adequate separation between topics. But in the future, it may become common to manage **collections of vector-search indexed databases.**

The advantage is that with multiple databases we can *guarantee* that a search won't inappropriately pull up sensitive (private, proprietary, classified…) content.

So within a few years, a single Colbert Search or FAISS search could automatically classify the query to decide which database(s) to include, and then search separately in them, and then merge the results.

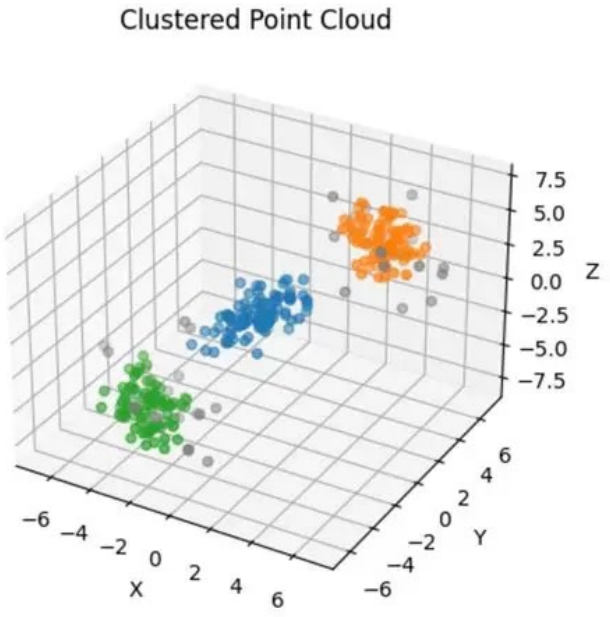
VECTOR DATABASES PROVIDE THE DOCUMENT RETRIEVAL FUNCTIONALITY

A vector database is a form of database designed specifically to deal with this category of somewhat unstructured data

It requires an *embedder:* A form of ML component that takes documents, chunks them and then identifies a data point in a high dimensional space for each chunk. One document could map to many points, hence an embedder actually outputs a tensor: a matrix where each row is one vector.

Even so, when we use the term *embedding* we are talking about **one** of those points: a vector that could have thousands of elements, normally floating point

HIGH DIMENSION VECTOR SPACES



All of us are familiar with 2D and 3D spaces

But it is impossible to visualize 200D spaces – at best we can project the data down to 2D but doing so loses a great deal of information.

Still, the basic idea is that with a high-quality embedder, these embedded points will "represent" knowledge in a very abstracted sense

OBSERVATIONS

If our embedding works well, related information will be close to one-another and unrelated information will be highly separated

For a particular ML task, like an LLM query or an LRM interaction that creates a multi-step plan, the query itself establishes the relative importance of certain concepts.

EXAMPLE

Suppose that after a long day, some climbers are unwinding and rehydrating.

One person complains that "My shoulder is really sore again!" A friend suggests asking the LLM for advice.

It suggests obvious things like ice, easing up on that shoulder for a few days, maybe seeing a PT or a doctor if the problem doesn't go away.

EXAMPLE

But now our climber is back home and decides to see a PT

"After a day of climbing, my right shoulder is sometimes really sore, even if I don't put a lot of stress on it or fall or anything."

The PT might consult an LLM. It would probably suggest getting a full history and assessing for possible tendon tears, tendonitis, rotator cuff injuries, etc. It might also suggest stretches and strengthening ideas if there is no sign of a more serious injury.

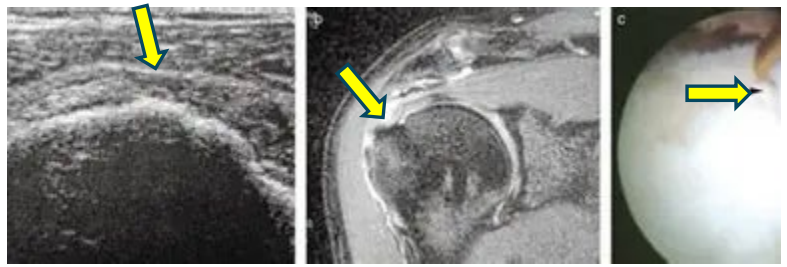
EXAMPLE

The PT recommended seeing an orthopedic surgeon.

The patient explains that for a few months now, each time she goes climbing her shoulder seems unreasonably sore.

The physician might consult an LLM to scan her medical records, look for recent MRIs, perhaps to prepare a lab order for a new MRI.

EXAMPLE



Back in the physician's office, the doctor might ask the LLM to pull up the sequence of MRIs dating back a few years and highlight new findings.

If the LLM includes a VLM expert, it might ask that expert to assess the series of images, or it might just sort through pathology reports

The summary suggests assessing for a possible partial bursal side tear in the rotator cuff, and the LLM highlights the location on a screen image

NOTICE THAT THE PROMPTS ARE SIMILAR!

The original prompt might look very similar in all of these cases!

The augmented prompting contextualizes and leads the LLM to generate very different advice and perhaps take different actions

And the augmentation is coming from information extracted from the database –a "vector" database. But probably the different specialists use very different MLs, each linked to distinct, specialized databases.

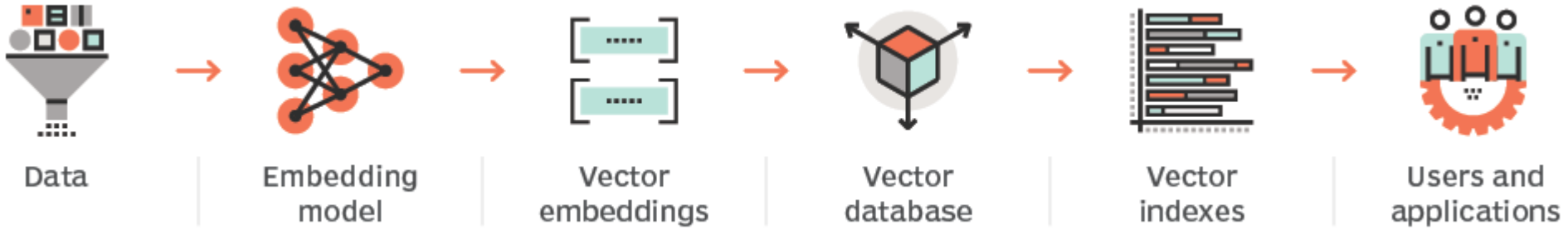
… THUS A VECTOR DATABASE

Starts with a more standard database, containing various types of data

Each data item has one or more embeddings into a single, shared, high dimensional space. Images, text, videos – all of them coexist in the single space. (This makes embedding a whole science, by itself!)

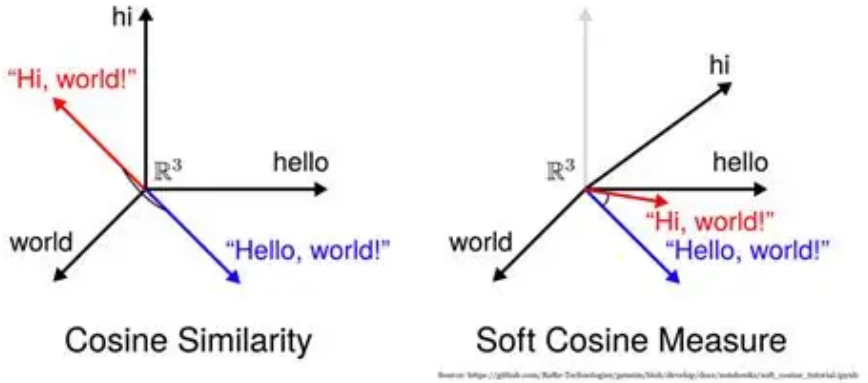
The fundamental operation is distance measurement: given an embedding representing our query, which objects are close in the vector database?

. . . LEADING TO THE SCHEMA FROM SLIDE 2



How do we search with a tool such as this?

THERE ARE MANY WAYS TO COMPUTE SIMILARITY



We can compute the Euclidean distance: length of the vector from the query embedding to the vector database object embedding.

We can use a slightly faster computation called Cosine Similarity. Same idea but it avoids the need to compute square roots. If you imagine an embedding to be a vector from the origin to the embedding coordinate, cosine similarity is the cosine of the angle between the query vector and the object embedding.

IMPLEMENTING A VECTOR DATABASE

Precompute an index to make it fast to carry out these kinds of searches. Then offer a search API: Given a query, it returns the documents that match

KNN requires exact results: the K closest embeddings. This can be too costly to compute if the collection is large, and so is rarely used.

ANN looks for a sampling among the documents close to our embedded query, and also aims for "diversity" – if the query is near a few clusters of data, it tries to include samples from each cluster.

FAMOUS VECTOR DATABASE PRODUCTS

Pinecone, Weviate, Chroma, Faiss, Colbert, Qdrant, Milvus, PgVector

There can be many criteria for deciding between the options.

Some products view vector database indexing as a feature, not the whole story. For example, Timescale DB mixes traditional SQL data retrieval with fast support operations over time ranges
It includes a pgvector indexing API for ANN search

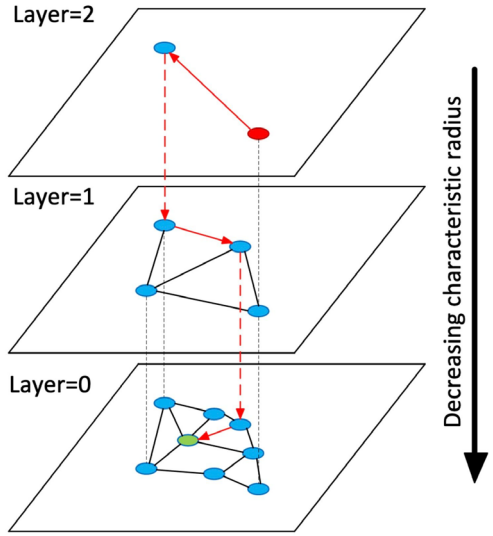
HOW DO THESE PRODUCTS IMPLEMENT ANN?

There are many famous data structures for approximate search

We typically build a graphical index, then search within it

Homework 1 looked at a simple form of balanced ANN. A widely used production quality index is the Hierarchical Navigatable Small Words index: HNSW. Open source, and used in many projects

HNSW INDEXING

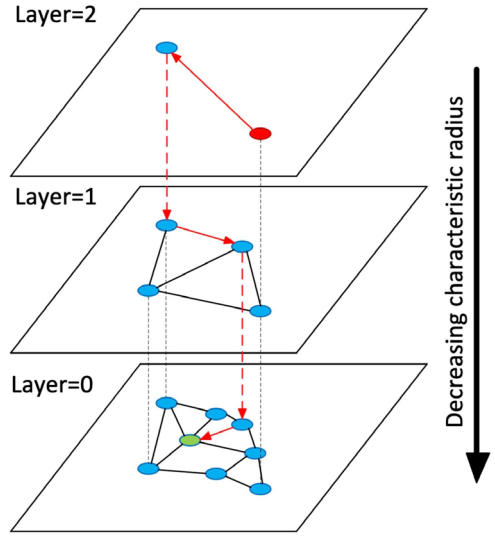


Hierarchical Navigable Small World, or HNSW is used for in-memory indexing. Nodes are embeddings and directed edges connect the "neighborhood"

HNSW creates an expander graph in which there is a path from any node to any other node that has shallow depth (usually log(size of the data set))

For example, how would you "pass a note" person to person to the pope? It has been shown that in human society, there is a path of people who know one-another no more than six hops in length!

WHAT DOES THE ILLUSTRATION SHOW?



We think of HNSW in layers

A layer is defined by "distance of paths to our target area." Layer k has nodes (embeddings) that are all roughly k hops from the neighborhood we are seeking.

HNSW scouts each layer (red search) until it finds a hop that will take it to a closer layer, then jumps to it. The first hop is roughly ½ the distance, then ¼, 1/8, etc. When we reach the neighborhood, we switch to random sampling.

HNSW IS POPULAR!

Easily half the products for RAG database search use it

But it has a limitation if a graph has a LOT of embeddings, like billions: it tends to require too many disk reads if the graph doesn't fit in memory

So HNSW is a great choice for graphs that fit entirely in memory, but we tend to switch to other options if the number of embeddings is huge.

… DISKANN

DiskANN

Almost any database product can be used as a vector database if it has explicit support for objects with embeddings and has indexing for ANN search!

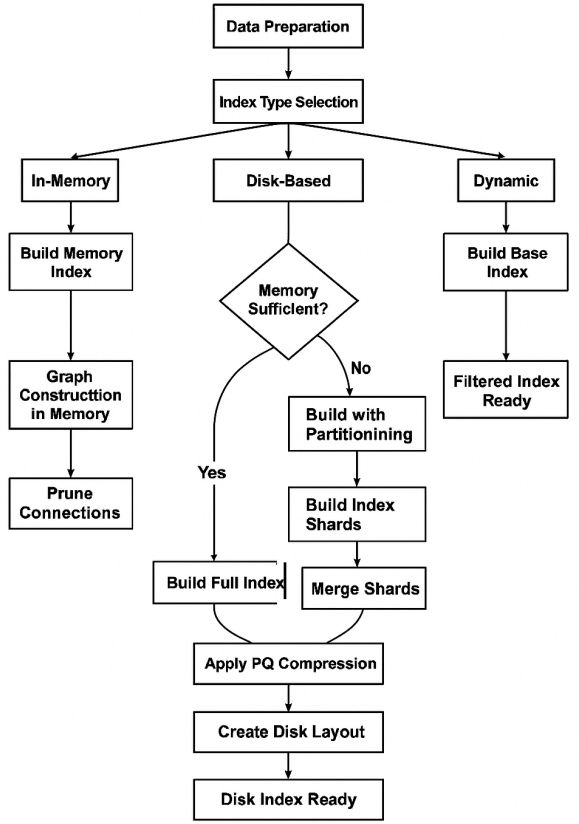
DiskANN is a Microsoft Research project that yielded an especially good solution. It is open and widely popular, and optimized to minimize the disk I/O required for doing the graph search. HNSW lacks this optimization.

Overview of the DiskANN Project (2018–present)

DISKANN DATA STRUCTURE IS ALSO A GRAPH

However, graph construction is slow.
The algorithm is very sophisticated

Idea is to start with a random graph
but improve it into a low-depth expander
graph while also grouping “clustered”
embeddings to form large “pages”



Flow Chart for DiskANN index builder

TO PERFORM ANN SEARCHES, DISKANN USES BEAM SEARCH

… A heuristic approach where only the most promising ß nodes (instead of all nodes) at each step of the search are retained for further branching.

ß is called Beam Width.
Here, ß = 5.

Beam search is an optimization of best-first search that reduces its memory requirements.

BEAM SEARCH ALGORITHM

OPEN = {initial state}

while OPEN is not empty do

1. Remove the best node from OPEN, call it n.
2. If n is the goal state, backtrace path to n (through recorded parents) and return path.
3. Create n's successors.
4. Evaluate each successor, add it to OPEN, and record its parent.
5. If |OPEN| > ß , take the best ß nodes (according to heuristic) and remove the others from the OPEN.

done

…DISKANN VERSUS HNSW

HNSW is more widely used in today's RAG database servers. But DiskANN is popular for ANN searches you could run on your own laptop.

Implemented as an open-source vector database index for use on a laptop or other small computer with SSD storage and limited memory

The pages are typically 1MB in size, and group embeddings of objects so that similar objects are on the same page

SEARCH CAN BE UNDERSTOOD AS HAVING THREE PHASES

Initially, we assume the user starts at a random node. Phase 0.

Phase 1 seeks to find the neighborhood(s) in which close matches will reside. The embeddings in any single neigborhood are ideally grouped on a small number of pages

Phase 2 searches the neighborhoods to solve the ANN task, with a heuristic to improve diversity: we don't want all our documents coming from just one cluster of embeddings if other clusters also have promising matches.

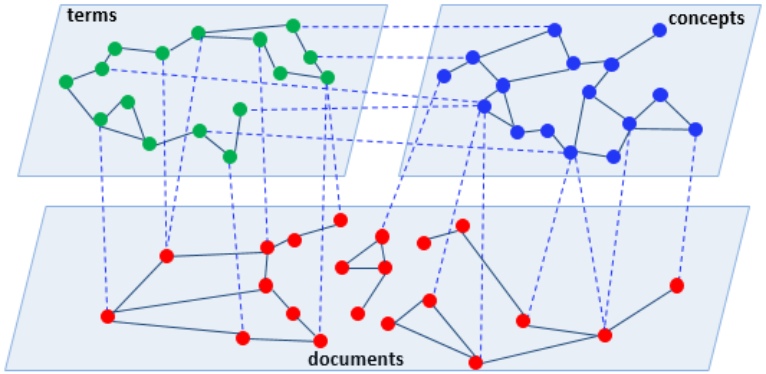
WHY THIS FOCUS ON PAGES?

The heuristic in DiskANN is designed so that after doing **a small number of disk reads,** ANN search can finish up using phase-3 search on in-memory data.

This makes DiskANN fast as well as efficient.

For a database with a billions item, the whole index can still fit on a laptop and searches run in fractions of a section

WHAT ABOUT THE DATA OBJECTS THEMSELVES?



When we use DiskANN, we save tuples: (embedding; object-id)

The objects themselves are saved in a KVS and can be fetched with a separate query

Thus the search returns a list of matching embeddings but we can fetch the objects themselves using a **get** operation in our KV store.

WHAT ABOUT FAISS?

FAISS is an open source graph search from Facebook

One benefit of this approach is that if a GPU is available, it can be used to accelerate the search.

FAISS is also very suitable for batched searches where we have several embeddings and want to compute ANN for all of them at the same time.

MODERN CHALLENGES?

RAG databases are growing larger and larger and over time could be as large as big-data collections in the cloud

As a result, we may soon see systems with ANN searches involving trillions of documents, not just billions. Indices would be 1000x larger

DiskANN must be deployed as a distributed index for such cases, spread over multiple machines and running as an “indexing service”

MODERN CHALLENGES?

Dynamic updates pose a number of challenges

- It is easy to add items to a DiskANN index: a quick algorithm for creating the index is to just search for the embedding we are adding, then link the new embedding to a set of approximate neighbors
- But deletion is difficult. Best is to leave a placeholder and just mark that this was once an embedding, but has been deleted
- Also, because this way of doing additions ignores the paging concept, over time the DiskANN paging layout becomes unbalanced and must be rebuilt, which is a very costly process.

MODERN CHALLENGES

We may need to tweak the LLM when we update the data!

To generate a high quality vector query, the LLM needs to be trained on the contents similar to those its RAG database to ensure that the LLM understands the terminology important in the target domains.

If key concepts evolve over time, even if you update the RAG database with the right content, the search might not find it because the query might not reflect these “new” but important considerations.

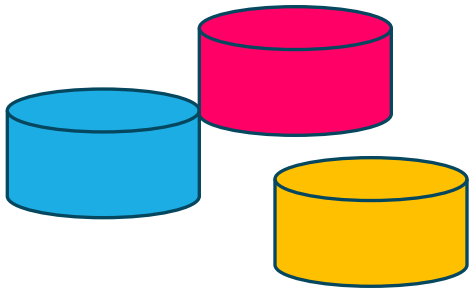
MODERN CHALLENGES

It is hard to integrate time into vector database indexing, yet there is growing need to search by time in many ML settings.

"Show me how crude oil pricing has changed in the past month, and compare this with gas-station pump prices seen by consumers"

Do a beam search first, then filter the results? Or should we somehow be searching in a time-filtered way, "directly"?

MODERN CHALLENGES



We may also see uses who need more than one database

For example, suppose that Hertz rental cars is creating LLM tools for customers, rental counter agents, people who inspect returned cars, management at various layers, travel agents, etc.

- Rather than one RAG LLM for everyone, it may be more effective (and safer in a security sense) to have one per use case.
- This way the LLM at the rental counter can't reveal information about a big merger that the company executives are negotiating and using *their* LLMs as planning tools.
- But would we literally want K entire indices? Or can we make better use of the limited memory for in-memory with a modified caching approach pages?

MULTITENANCY MAY BECOME THE NORM

In fact you can imagine a future where we might want a different RAG database for each user and in many cases, multiple databases

This leads to a question of how to take a normal RAG database and then extend it with "deltas" efficiently, so that the users could share the main RAG database.

Perhaps we can create one RAG index but combine it at query time with smaller "adaptors" to customize the main one.

WHAT ABOUT GENERATIVE QUERIES AND AGENTIC AI?

It is important to keep similar-sounding but unrelated concepts separate:

- Today we learned about **ANN search in vector databases.**
  Used in: prompt augmentation, generative result formation, knowledge retrieval.

WHAT ABOUT GENERATIVE QUERIES AND AGENTIC AI?

It is important to keep similar-sounding but unrelated concepts separate:

- Today we learned about **ANN search in vector databases**
- **Generative queries** arise when we use an LLM or LRM to generate database queries: SQL code to perform some operation on a standard relational or transactional database. SQL doesn't use approximate search. A person who likes vibe coding might call this "vibe coding for relational databases."

WHAT ABOUT GENERATIVE QUERIES AND AGENTIC AI?

It is important to keep similar-sounding but unrelated concepts separate:

- Today we learned about **ANN search in vector databases**
- **Generative queries**: SQL code to perform some operation on a standard relational or transactional database.
- **Agentic AI** systems are Ais that "send" requests to other kinds of systems. These could include queries to a database but more often would be cases where the AI is doing something on behalf of the user, such as sending email, trading stocks, or booking airplanes and hotels for a trip

WHAT ABOUT GENERATIVE QUERIES AND AGENTIC AI?

It is important to keep similar-sounding but unrelated concepts separate:

- Today we learned about **ANN search in vector databases**
- **Generative queries:** SQL code to perform some operation on a standard relational or transactional database.
- **Agentic AI** systems are AIs that "send" requests to other kinds of systems.

All are important, but ANN search today is mostly internal to ML services.

SUMMARY

Two weeks ago we saw two major cloud services for important kinds of data at Facebook: Its cache for resizable blob content, and its social network graph.

ML services need databases too – potentially, one or more per user!
Vector database indexing is an important tool for supporting RAG databases.

Key concepts include embeddings, indexing, similarity metrics, approximate nearest neighbor search (ANN) and fair sampling.

SELF-TEST

Your friend took a course and learned all about relational databases.

But now you are both working on an LLM system that requires using a vector database.

How would you explain to your friend the similarities and differences between a vector database index and a relational index, on the same field?

SELF-TEST

An LLM is being used at an appliance warehouse company that just hired you as their new expert hacker.

The company constantly changes prices, makes special offers, brings in new products, etc. So the vector database is continuously being updated

Would you expect that:

- The LLM will be able to fetch the right documents for customers with no retraining when these updates occur?
- The LLM *plus some adjustments to the prompting strategy* would work?
- The LLM might periodically need some retraining.

Why? Don't just guess yes or no – justify your answer. It might require research!

SELF-TEST

We never really talked about embedding, so this may require a little research (assisted by your favorite LLM!)

Suppose that you are using Meta's Llama embedding technology for documents but are creating a specialized database of documents and experimental data for quantum computing technologies

Would you expect that you might need to find a different embedding solution, or will the existing one probably be fine?

SELF-TEST

Could a vector database be useful in settings unrelated to ML?

Consider situations where we might consult a campus "app" for Cornell. Are any of them good candidates to leverage vector database ANN?

Historically, systems like Google Maps used indices to search for data by geographical location, combined with query terms. Would a vector database be a good "modern" option for this task?

SELF-TEST

We learned about dynamically updating data in KV stores and file systems. Sometimes consistency turned out to be an issue.

Would we need consistency when updating a vector database?

Suppose you own this task at a financial firm. Would all the updates be incremental "additions" or would some updates need to delete objects, or replace them with a newer version? How would this impact a vector database index that your firm is using to continuously query that database?