* What is a vector database? Well, they say a picture is worth a thousand words. So let's start with one. Now in case you can't tell, this is a picture of a sunset on a mountain vista. Beautiful. Now let's say this is a digital image and we want to store it. We want to put it into a database and we're going to use a traditional database here called a relational database.
* Now what can we store in that relational database of this picture? Well we can put the actual picture binary data into our database to start with, so this is the actual image file but we can also store some other information as well like some basic metadata about the picture so that would be. things like the file format and the date that it was created, stuff like that.
* And we can also add some manually added tags to this as well. So we could say, let's have tags for sunset and landscape and orange, and that sort of gives us a basic way to be able to retrieve this image, but it kind of largely misses the images overall semantic context. Like how would you query for images with similar color palettes for example using this information or images with landscapes of mountains in the background for example.
* Those concepts aren't really represented very well in these structured fields and that disconnect between how computers store data how humans understand it has a name. It's called the semantic gap. Now traditional database queries like select star where color equals orange, it kind of falls short because it doesn't really capture the nuanced multi-dimensional nature of unstructured data.
* Well, that's where vector databases come in by representing data as mathematical vector embeddings. and what vector embeddings are, it's essentially an array of numbers. Now these vectors, they capture the semantic essence of the data where similar items are positioned close together in vector space and dissimilar items are positioned far apart, and with vector databases, we can perform similarity searches as mathematical operations, looking for vector embeddings that are close to each other, and that kind of translates to finding semantically similar content.
* Now we can represent all sorts of unstructured data in a vector database. What could we put in here? Well image files of course like our mountain sunset. We could put in a text file as well or we could even store audio files as well in here. Well this is unstructured data and these complex objects They are actually transformed into vector embeddings, and those vector embeddings are then stored in the vector database.
* So what do these vector embeddings look like? Well, I said there are arrays of numbers and there are arrays of numbers where each position represents some kind of learned feature. So let's take a simplified example. So remember our mountain picture here? Yep, we can represent that as a vector embedding. Now, let's say that the vector embedding for the mountain has a first dimension of say 0.
* 91, then let's say the next one is 0.15, and then there's a third dimension of 0.83 and kind of so forth. What does all that mean? Well, the 0.91 in the first dimension, that indicates significant elevation changes because, hey, this is the mountains. Then 0.
* 15 The second dimension here, that shows few urban elements, don't see many buildings here, so that's why that score is quite low. 0.83 in the third dimension, that represents strong warm colors like a sunset and so on. All sorts of other dimensions can be added as well. Now we could compare that to a different picture. What about this one, which is a sunset at the beach? So let's have a look at the vector embeddings for the beach example.
* So this would also have a series of dimensions. Let's say the first one is 0.12, then we have a 0.08, and then finally we have a 0.89 and then more dimensions to follow. Now, notice how there are some similarities here. The third dimension, 0.83 and 0.89, pretty similar. That's because they both have warm colors.
* They're both pictures of sunsets, but the first dimension that differs quite a lot here because a beach has minimal elevation changes compared to the mountains. Now this is a very simplified example. In real machine learning systems vector embeddings typically contain hundreds or even thousands of dimensions and I should also say that individual dimensions like this they rarely correspond to such clearly interpretable features, but you get the idea.
* And this all brings up the question of how are these vector embeddings actually created? Well, the answer is through embedding models that have been trained on massive data sets. So each type of data has its own specialized type of embedding model that we can use. So I'm gonna give you some examples of those.
* For example, Clip. You might use Clip for images. if you're working with text, you might use GloVe, and if you're working with audio, you might use Wav2vec These processes are all kind of pretty similar. Basically, you have data that passes through multiple layers. And as it goes through the layers of the embedding model, each layer is extracting progressively more abstract features.
* So for images, the early layers might detect some pretty basic stuff, like let's say edges, and then as we get to deeper layers, we would recognize more complex stuff, like maybe entire objects. perhaps for text these early layers would figure out the words that we're looking at, individual words, but then later deeper layers would be able to figure out context and meaning, and how this essentially works is we take the high dimensional vectors from this deeper layer here, and those high dimensional vectors often have hundreds
* or maybe even thousands of dimensions that capture the essential characteristics of the input. Now we have vector embeddings created. We can perform all sorts of powerful operations that just weren't possible with those traditional relational databases, things like similarity search, where we can find items that are similar to a query item by finding the closest vectors in the space.
* But when you have millions of vectors in your database and those vectors are made up of hundred or maybe even thousands of dimensions, you can't effectively and efficiently compare your query vector to every single vector in the database. It would just be too slow. So there is a process to do that and it's called vector indexing.
* Now this is where vector indexing uses something called approximate nearest neighbor or ANN algorithms and instead of finding the exact closest match these algorithms quickly find vectors that are very likely to be among the closest matches. Now there are a bunch of approaches for this. For example, HNSW, that is Hierarchical Navigable Small World that creates multi-layered graphs connecting similar vectors, and there's also IVF, that's Inverted File Index, which divides the vector space into clusters and only searches the most relevant of those clusters.
* These indexing methods, they basically are trading a small amount of accuracy for pretty big improvements in search speed. Now, vector databases are a core feature of something called RAG, retrieval augmented generation, where vector databases store chunks of documents and articles and knowledge bases as embeddings and then when a user asks a question, the system finds the relevant text chunks by comparing vector similarity? and feeds those to a large language model to generate responses using the retrieved information.
* So that's vector databases. They are both a place to store unstructured data and a place to retrieve it quickly and semantically.