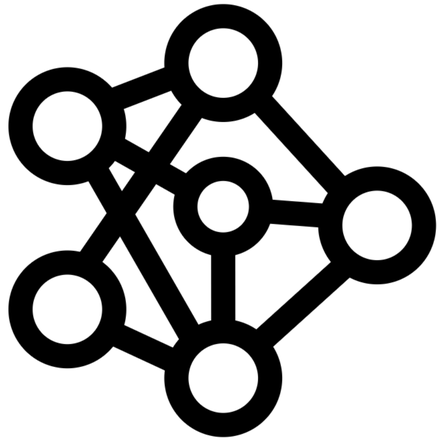


Why your brain is 3 milion more times efficient than GPT-4 !

dead simple introduction to Embeddings, HNSW, ANNS, Vector Databases and their comparison based on experience from production project.

Learn more about Vector Databases

Olaf Górski

Intro

How it all came to be

Recently I had to go on journey into the Vector Database world and pick one for a particular project. And oh boy, was it a ride. Recently Vector Databases have gained a newfound attraction and spotlight thanks to the rise of LLMs. Such situation is both a blessing and a little bit of a curse, bringing some of the things attributed to LLMs to Vector Databases too - the perception they are 'a new thing', untested technology or some shiny crap good only for LLM-based stuff. All of which is wrong.

In case of Vector Databases I was, and still am, actually, completely green. So I had to go around, gather facts, check stuff out and form my opinion regarding each particular database, as to select one for my particular use case.

Before we begin, let me put a disclaimer here. I do not title myself as an expert in AI. There are all my just personal ramblings, opinions, certainly biased. Take everything you read with a grain of salt and do your own due dilligence.

Vector Databases. Before?

Vector Databases have been with us for waaay longer than most know. There's legit engineering, science and rigorous testing behind most of them over the span of multiple years. And there's a lot of them, they all have their own quirks, just a little bit like RDBMSes do, except the situation there is quite clear, at least for me, in most cases postgres FTW, I mean it's not 2005 anymore, we won't go with mysql or LAMP stack, right? Do most of you even remember what it is or am I exaggerating things?

Before we actually begin I think we should dig a bit deeper into what exactly we are playing around. If you are not interested in understanding how this all works in detail and how it relates to generative AI, feel free to skip this part and go over to the next section.

Introduction to Vectors, Similarity and our amazing brains

First we must lay down certain axioms (smart word for the common sense/ground rules we all agree upon and accept as true).

One of such would be the fact that currently computers do not really understand words. They operate in what is called binary language, so a string of 0s and 1s. It's related to how electric charge, electrons and atoms work. Physics stuff. Very basics of CS and electronics.

Computers do not understand words, they operate on binary language, which is just 1s and 0s, so numbers. Computers only understand numbers.

1.Computers do not understand words, they operate on binary language, which is just 1s and 0s, so numbers. Computers only understand numbers.

No words?

How to live with only numbers

Knowing that, it becomes obvious, that we may face some problems. First of all if it's all just 1 or 0, how do we express different numbers?

Well, that part is easy, because thanks to the decimal counting system.

Long story short, we have 10 digits to express stuff, in binary you have 2. That means that you still can represent other numbers with base 2, the resulting number will be just a bit longer and calculated differently. I won't to into too much of a detail here, you can look it up yourself -> binary system, decimal system and how to convert from one to another. Or ask chatgpt for an explanation.

It does a fine job. Just know that any natural number can be easily converted from base ten to binary e.g. "4" in decimal i s "100" in binary. They hold equivalent value, but are expressed differently. The case is a bit more complicated for floating point numbers, but let's not get into that now.

So first layer of abstraction is down.
We have computers. Electric. Electrons. High current/low current, current/no current. 1s and 0s.

These 1s and 0s of ours now can magically become numbers.

This is what our computers operate on - in every case.

When you dig deep enough, everything you interact with in the virtual world is in fact a number,

so a particular configuration of these tiny little something's in the computer that either have electricity running through them, or they don't and that is interpreted as a number. Imagine 3 light bulbs.

1st is on, 2nd and 3rd are off. 1-0-0. We magically agreed that this means 4.

Now make the size smaller probably by a thousand, a milion times or something,
and you get an idea what's going on in that shiny macbook of yours. Back to the topic.

Numbers. Yes, we have them and are able to somehow interpret the current state of your computer, or it's part as it being a particular number.

Now that's not very useful, right?
We, humans, mostly operate on text and language, right? Yup.

Imagine a world where you have to decode numbers. in some magic dictionary to a word.

The computers wouldn't be so useful now, would they? But that's how it looks in the background.

Long time ago some smart dudes have gathered around that agreed that from now on, if we know we are dealing with text, we should interpret the number 65 as "A", "B" as 66 and so on.

Of course, it changed, different people had different standards and agreements but who cares, let's simplify. We come to understand the second axiom:

So the second layer of abstraction is down.
We, as society, have assigned certain roles/meaning to particular numbers, in this case it's letters.

We, as socity, have assigned certain roles/meaning to particular numbers, in this case it's letters.

Now that we have our computer, that understands numbers, and we know which numbers in which case are which letters, we kinda can start forming words for example. And if you have words you can have sentences, code, whatnot. That's more useful. All of your software builds on top of this base truth.

Now that we know that all words in fact, are nothing but numbers for your computer, we can go further than that.

For the computer, a given word, regardless of the context, based on the above, holds the same 'meaning' or 'value'. So even though you can say 'dust the furniture' as in clean or 'dust on the furniture' as in there's a mess, the representation (and the meaning) for the computer would stay the same, even though there's a mess and clean are two totally different things. So, even though we can convert a word for a distinctive number for the computer so it 'knows' kind of, what's up, there's missing 'context' to judge the meaning.

Words simply represented as binary numbers lack context.

Well, humans can be ingenious in the ways they do wicked things, but also in the ways the do great things. One such example here would be Contextualised word embeddings. What does that stand for? Well, let me tell you.

So we have this now simple case of turning a series of 1s and 0s into numbers, numbers into letters and then words.

That you already understood, right? We kinda explained the problem with that - lack of context. Well. We have developed methods that allow us to generate different and unique numbers for words, depending on their context or semantic meaning.

So if we used Contextualised Word Embeddings in our example above, the 'dust' from 'dust the furniture' would get totally different number than the word dust in 'dust on the floor'. Now the computer knows that these two have different meaning! How does that exactly work? It's a piece of work, let me tell you, but we won't cover that in this presentation.

First of all I'd have to understand that myself haha.

Long story short, we have a magic function that does this:

```
number_representation_of_a_word_and_meaning("dust", context="dust on the floor") == 42
number_representation_of_a_word_and_meaning("dust", context="dust the floor") == 24
```

The values are not random - words that within certain context have different meaning probably will have very different number assigned to them. A number that is far apart from the other one. If the meanings were similar, but just slightly different depending on the context, we'd have numbers that are closer to each other.

```
number_representation_of_a_word_and_meaning("bark", context="bark is a noun") == 12
number_representation_of_a_word_and_meaning("bark", context="bark is a verb") == 19
```

Why is that? "Bark" and "bark" both relate to something that is associated with outer protection or covering. If a dog barks usually it is a sign of danger, something related to safety, protection. The tree has a bark in order to protect itself from the environment. It's different, still, but this time we share a certain common theme at least in some sense, hence the numbers we got are closer together than in the previous case, because they are more similar.

Similar meaning of words, in a particular context, converted to numbers, are closer to each other than radically different meanings even if the words look the same.

By the way, what are embeddings you might wonder? Why did I not describe it earlier? Because we needed to understand the above to understand embeddings.

More or less it's what we described, so the way of representing a number based on particular set of criteria (in our case the semantic meaning) as similar numbers, even if they are not similar raw value wise.

So in the embeddings world, 'dust' would be, as a number, closer to 'clean' than 'dust' would be to 'dirt'. At least that's how I understand it. And 'dust' as in 'dust the floor' would be far away from both 'dust on the floor' or 'dirt'.
Alright, this one was a bit trickier, but we got there eventually.

Again, let me reiterate.

We have electrons and then electricity.

Because of electricity (or to be more exact high/low current) we have 1s and 0s, so binary language.

Next are numbers, binary numbers, so just numbers but expressed only with 1s and 0s, base 2.

That gets converted to base 10, so the decimal system that we humans mostly use (not everywhere and not always lol - hi friends from India or the ancients). These numbers then get simply mapped to words with the same values regardless of the context, of the meaning, in the base case.

But nowadays, we iterated over that and can associate words with their meaning and assign different numbers, based on the context of the text or a sentence (sentence transformers sounds familiar?) and their semantic meaning.

The groundbreaking publication about Transformer model. Go look it up.

By the way look at the year it was published and how quickly we went from theoretical stuff to something as amazing as GPT-4. INSANE. Either way.

Semantic meaning, embeddings and transformer model. This is the reason that the LLMs or generative AI can sound so convincing, as if it really knew the stuff it generated. Nope, it doesn't. It just processes numbers, they do not intrinsically understand the meaning, they understand that 1 is closer to 2 than 24 is to 42, which is the answer to everything.

See what I did there? This does not imply it 'understands' the meaning in the human sense and why I'm quite sure that AGI (Artificial General Intelligence), so something that 'really' thinks in the most human way, is so far away. The current technology and models, simply do not work like we assume them to.

Okay champ. We have these words mapped to different numbers, which are either closer or further away, depending on the semantic meaning in a particular context. What now?

Well, math magic comes in. I ain't no maths guy, I do enjoy the thought exercises or elementary and truthfully representation of stuff it can provide, nowadays I'd struggle to solve quadratic equation probably, being just a simple highschool dropout, but let me share with you my understanding of this absolutely marvellous phenomenon that happens.

So, numbers. In maths we have this thing called Algebra.

It's a study of relationships between things that vary over time, which are represented by symbols, usually letters, which also can represent things, usually numbers. What? ... Yeah. That would be my reaction to this statement too, at least was in the past.

Let's make it a bit more digestible. To put it in simple terms, for us numbers are things that allow us to measure reality, count things, they are building blocks of reality you could say. Almost everything (or as some would argue - everything), given a proper formula, could be represented as a number.

Be it something so physical as the length of your arm, number of apples in your kitchen or something as abstract as human language or even the meaning of a word, which we have demonstrated above.

That however relates to this particular branch of mathemathics and 'real' stuff. In here numbers are immanent, simple, natural, the best. What about branches of maths that operate on a bit different set of rules and so on? Different universes of sorts? Well, in there numbers might not be the best base unit of operation, in different universes we might need either different base building blocks or we might want to take numbers and extend them a bit, if you will, to make life easier.

One such example is the study of Linear Algebra. Linear Algebra has 'vectors' (which in fact are numbers with certain additional stuff to abstract ideas and make life easier) the same way basic Algebra has numbers/symbols/variables. I won't go into the details here too much as my understanding is also not so perfect, and probably I'd spout some gibberish, but I hope you get the analogy.

Space can have many 'dimensions', similarly with vectors. If you have one dimension, it's just [1].
If we have two, e.g. X & Y axis, we can position a point in place by having, well, two numbers for each dimension, eg [1,1], this should ring a bell if you remember anything from your middle school math class. What if we have 3D space? Well, similar stuff.

Notice how each additional dimension multiplies the number of all the possible vectors.

So let's assume we are working on a limited range of natural numbers, 1 to 10. In 1D case, we have 10 options. In 2D space we have 10 possibilities for first one, 10 for the second one, which totals to 10 * 10 = 100 possible different combinations. In case of 3d? You get the idea.

So now imagine how insanely big the possibilities are, if we are working with a bit of a broader range (e.g. 65536 or 256) and not with 3 dimensions, but with 1536! That's INSANE. Absolutely insane, but also what allows our LLMs to seemingly perceive so many nuances in the meanings of the words and what nots.

I know it was a long digression, but bear with me. Again.

We have electrons and then electricity. Yeah, I'll spare you that repetition again. Number representation of a word based on semantic meaning. So we have a number per meaning in a given context let's say.

This number gets then turned into a vector which lives inside vector space. To accommodate for the many meanings and contexts and how contextual our actions, words and speech are, this vector space should have appropriate number of dimensions, each dimension related to different part of true meaning/context.

It can't have too much as it'd get too huge to process. OpenAI settled for 1536 dimensions. Remember each 'dimension' can be assigned a different 'value', so the total number of possible meanings is THE_MAX_NUMBER_WE_OPERATE_ON to the power of 1536. THAT"S A LOT. Anyhow.

We went from having 1s and 0s, to our words being evaluated in 1536 dimensions or 'meanings'. The more true to a particular meaning is that word, the higher the value it'll get in that dimension.

As you can see we have a problem now. How do we actually process such a humongous amount of data? Traditional methods don't really work well or would be too expensive computationally or economically.

This is what we call "curse of dimensionality". To deal with this, stuff like Approximate Nearest Neighbour Search problem show up. Meaning: how do we find similar vectors (numbers) in very high dimensional spaces?

To resolve that puzzle we came up with Hierarchical Navigable Small World or HNSW. What is that? Long story short, even though the space is so HUGE, usually, in large networks, eg. human ones or social ones, despite the size, most nodes can be reached from any place in the network with surprisingly few steps. It's often reffered to as "six phones rule", that states that you are at maximum six phone call aways from anyone in the world.

For most people it’s within 2-4 range.

Imagine this, you potentially are 2-6 phone calls away from Donald Knuth.

You know someone who knows someone and so on, then boom. Your message gets delivered to Obama. It's quite interesting actually. Important to note is that not all nodes in the network are connected equally. There are these things called as hyperconnectors (there’s cool research/book about it recommended), which connect to A LOT of nodes, and the contrary so, the nodes that are a bit lonely. I think you get the idea and seen an example in your life - almost everybody knows that someone who seems to know everybody everywhere.

Either way, as I digress. If we take this Small World thing and add one more thing on top of it - hierarchy, managing even such a huge amount of data becomes doable. What does it look like? Simple.

Think in terms of enterprise org chart. It's as if, let's say, you were a CEO and wanted to know something about if your company does use type hinting in python like human beings do.

In HNSW the approach would be to first select a candidate, who might know something (be similar to the topic/value you search for) from a very few selected people in the company (mby the hyperconnectors should be at the top? ;)).

Let's say the are Lead/Director level people. So you e.g. have Director of PeopleOPs, Director of Product, Director of Engineering and so on.

Out of these, the semantic value of Director of Engineering will be most similar to "do we use type hinting in python". So we go to him and there we repeat the process.

However our guys here are lazy for whatever reason. He doesn't want to check or write the answer, so he delegates. -> who does the director know, who in the hierarchy is the level below him (these are the guys he knows best, he doesn't know the ones lower too well as he doesn't interact with them often) that should know the answer?

So he repeats the process: Engineering Manager in Frontend team, EM in DevOps, EM in Backend team. Backend Team it is.

Now this process gets repeated till we hit the bottom most layer of hierarchy, how many there are is up to the organisation to decide.

This way instead of the CEO asking maximum number of N people in the worst case, assuming that only the last person knew, the question gets asked only M maximum number of times, where M is the number of layers in the hierarchy we have. Or something like that.

Of course if he asked the question to everyone he would have better data and could select the best answer.

The one he will get in the case of using HNSW will be good enough (how good we want will require us to define a certain metric, but the more accurate the answer needs to be, the tricker it is computationally) but will cost SIGNIFICANTLY less to get.

Think sth like rolling out a feature that covers 80% of the needs in 2 weeks vs perfecting it to 99.99% in 2 years. 80% (or 50 or 95) is usually good enough when you weight the pros and cons, cost/benefit.

WOAH. Long story long, this is what enables us to process such huge amounts of data that is so high in dimensionality, that can accommodate the various context, nuances and meanings that is associated with for example human speech or thinking process.

Now imagine our brains do this at a higher and more profund level than this, they do it constantly, all the time, they fit into something small like our skull and run on the equivalent of 24 Watts of power per hour. In comparison GPT-4 hardware requires SWATHES of data-centre space and an estimated 7.5 MW per hour. So around **312 500x** less while doing stuff that is thousand times more sophisticated. WHAT THE.

We are a wonder of Nature. An amazing mess of a wonder but a wonder nonetheless. Also, keep in mind that our brains don't dedicate 100% of the horsepower to conscious thinking processes.

According to Auburn University paper I found, cognitive neuroscientists believe that only 5% of our cognitive activity is conscious. So multiply this number by anything from 20 to a 100.

Let's be conservative and say 10 for some reason. **That's still over 3 million times the efficiency than GPT-4**, while also having at least a **magnitude** (or at least N) greater complexity of processes to handle. I got carried away in the amazement.

TLDR: Electrons -> Electricity -> Binary -> Binary Numbers -> "Human" or base 10 numbers -> Letters -> Words -> Embeddings -> (Semantically) Contextual Word Embeddings -> Algebra -> Linear Algebra -> Vectors -> Vector Spaces -> High Dimension Vector Spaces -> Approximate Nearest Neighbour Search or just Nearest Neighbour Search (ANNS/NNS) -> Hierarchical Navigable Small World (HNSW)

Soak up all these terms and write them down. From here we are almost done. If you've been following closely you might start to piece the puzzles together.

LLMs are in fact just algorithms, very complex ones, that have indexed or ingested A LOT of data/words, vectorised and embedded them and their meanings in particular context. You can think of it as a VERY highly compressed library in fact.

Then, based on that data, they just 'guess', based on context, that if we have this particular thing or meaning in a semantic context, so embeddings, so numbers or vectors in high dimensional space, just right here, we probably have in mind this particular response which is nothing but a set of other embeddings.

So for 1 + 1 the answer is probably “= 2”, but can be “= 4” if the training data is bad or something. For LLM, even tho the answer is wrong, it’s equally right.

Have in mind tho, they do not understand that all. They just guess. LLMs are nothing more than stochastic parrots.

So, NO, LLMs do not “think”. Regardless of whatever someone tells you. It’s all just probabilistic entropy.

Speaking of let’s touch up on the ‘hallucinations’.

Answers from LLMs that are totally wrong are usually termed hallucinations - LLMs writing out things that are not true.

The problem with that is, well, to an LLM, they are, in fact no different from the the proper responses.

Does it mean that from LLM perspective all text they generate are hallucinations? Both yes and no, it’s a bit more complicated here.

Also if it’s all about probability and randomness, you now kind of know why it’s hard to get deterministic outputs for LLMs. It’s just not in their nature, however with certain steps you can almost get there, almost.

So if based on the context and approximation, we know that this particular query seems like something that will be associated with this particular token, we will use it. Then if we have one token ahead, we can repeat the process till we start hitting lower levels of association, which in turn means that we can probably stop generating as the answer is complete.

So... Yup. There's no thinking happening. Just assosciating numbers and guessing. You read that right. GPT-4 does not 'think' at all. It's as far away from human consciousness as we are from Putin being a good guy. This simply does not come even close. So it's the reason I do not worry at night (at least yet) AGI will come and replace me or go on a conscious crusade against humanity.

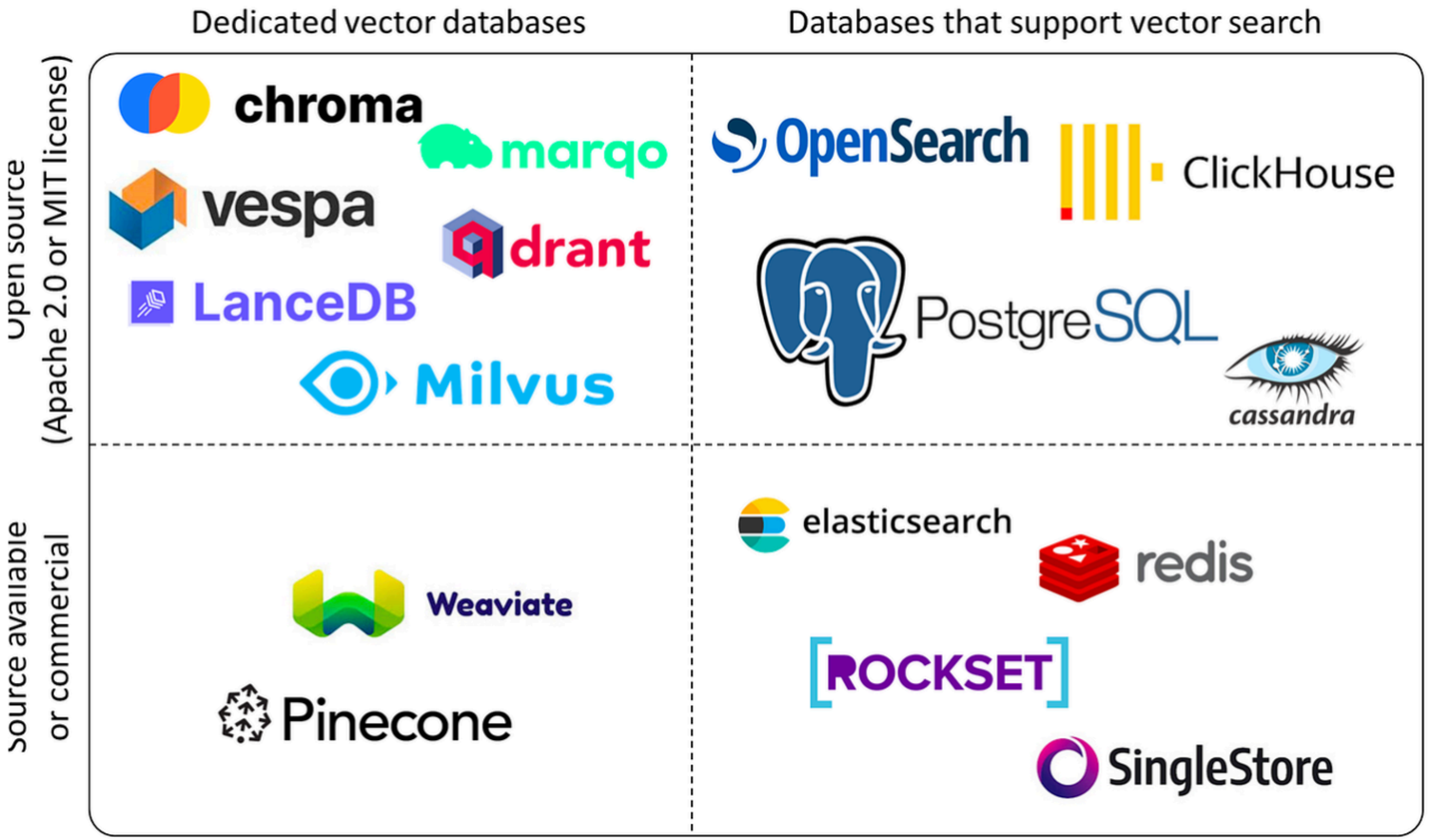
Bear in mind, my understanding of the stuff is VERY limited, I'm all new to this.

I've simplified A LOT not only for the reader, but also for myself. If any of this is unjustly inaccurate, let me know. I've googled most of the stuff along the way back. WHOAH. DAMN THAT WAS LONG.

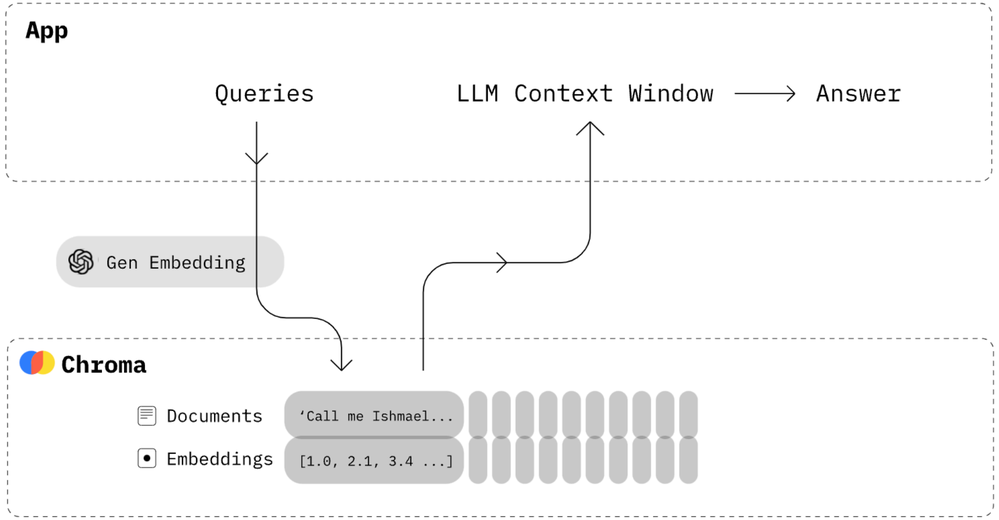
Let's answer the initial question then. How do vector databases work then? Well, they take the text. Vectorise it. Embed them in high-dimension space, then if you query it, they search for approximate similar tokens/pieces/texts. Simple, right?

Now with that over, let's get to the stuff I had in mind when starting the article, so comparison of vector databases and a raport from using some of the more popular ones.

Popular Vector Database providers



Starting point - chromaDB



My starting point was chromaDB. It allowed me get up & running quickly and seamlessly. It was dead simple and it got the job done. However it wasn't the best approach TBH. For a PoC? Yup, it did get the job done. But for more serious, potentially production-grade stuff? Not given how I implemented it.

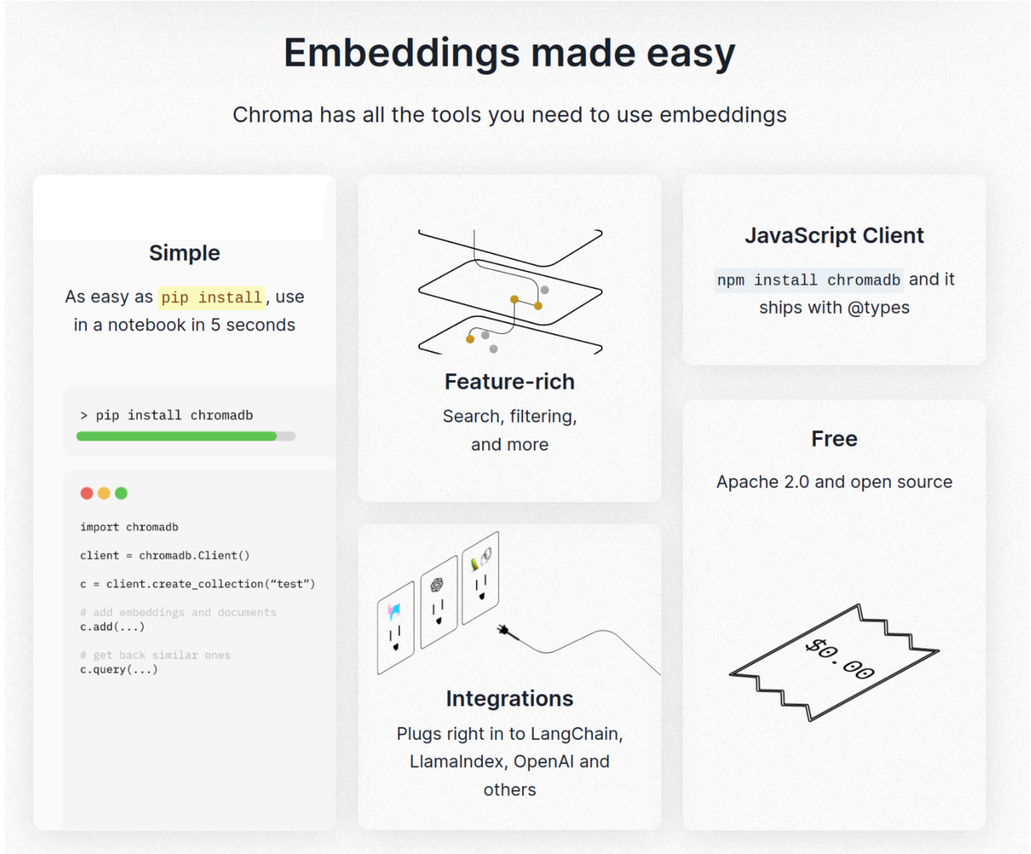
Long story short, for a PoC, I created this Q&A over a scraped knowledge centre app let's say. I needed a VectorDB deseprately. Pinecone was off-record for obious reasons that I'll list later on in the article, even though it was the easiest to implement I'd say. There was stuff like FAISS and what not, but the choice, over initial exploration, fell on chromaDB.

I used a very dirty approach, where I packaged it together in one container with the API backend part of the application. ChromaDB persisted the vectors in their internal format in a certain directory in my project's repo. I simply commited it to the repo. Not the best, I know.

Now, the whole VDB (Vector Database) was a part of our repo. I locally ingested the data that I needed in the VDB, then commited the state of the VDB to the repo.

chromadb got added as a project dependency (installing it is quite trivial - it's just a python package) to pyproject.toml and boom. Initialise it somewhere in the code during application startup and boom, you done. Again, this is very dirty approach.

API is packaged in the same container as the DB. In this setup changes in the VDB are not really persisted unless you commit them to the repo - only the commited state matters (so read-only apps or go home champ). They share resources. If one fails, the other does too. It was somehow scalable given that we had read-only app, so it was self-contained but...



The git repo got big, fast. The docker image build time got very long for my standards. I mean imagine 25-30m build time for a simple fastAPI app. Not exactly desirable. This came from the fact that the chromadb was inside the container and had to be rebuilt each time the code changed, and as you can imagine, rebuilding db package can be quite time consuming (looking at you, psycopg).

We couldn't go on for long with this approach. Ofc you can run chromadb as a separate container, self-hosted, but that added overhead I didn't need at that time for a PoC. At that time, from what I remember, there was also no option to do proper clustering that'd allow you to get prod-level scalability, survavibality and so on. TLDR: you could run with a single instance and that's it. I also didn't see helm/k8s OOB support that'd make it easy to deploy in a cluster somewhere. On top of that other projects were in the pipeline, then what? Each of them should have their own instance? How would we share data? All of this meant, it had to go away.

But it allowed me to get off the ground, integrate fast, LEARN a lot and so on. For this purpose it did an AMAZING job.

Chroma raises $18M seed

building the AI-native open-source embedding database

Also, what needs to be said is that some of the features mentioned weren't supported at the time I was creating the project, which was couple of months ago. Since then Chroma has closed $18M seed round, developed a solid roadmap and started addressing some of what was mentioned here. We had a nice talk with Jeff Huber, chromadb's Founder, about all of this. I really loved a lot how he was humble and helpful enough to admit, that given the preconditions for the project & productionisation, chroma at that time wasn't exactly the best choice, and choosing something else for our needs was a smart move! He is a very intelligent, passionate and brilliant guy, so I'm more than eager to see how they develop, however till we see the features shipped, in the use cases we had in mind, it was a no-go. However to get started, have one-self-contained app with db inside and be done in 4h instead of 4 days (I know i exaggurate)? Sure, why not. If I was to make a comparison to something people are more familiar with,RDMBSes, I'd say the current chroma is like sqlite. Easier to comprehend now where it can shine and where it won't? Great. Also remember, my approach was simple. If you do it properly, set up all the proper stuff, configure it nicely, adhere to best practices, you probably won't run into any problems. I did not do that, lack of time & expertise.

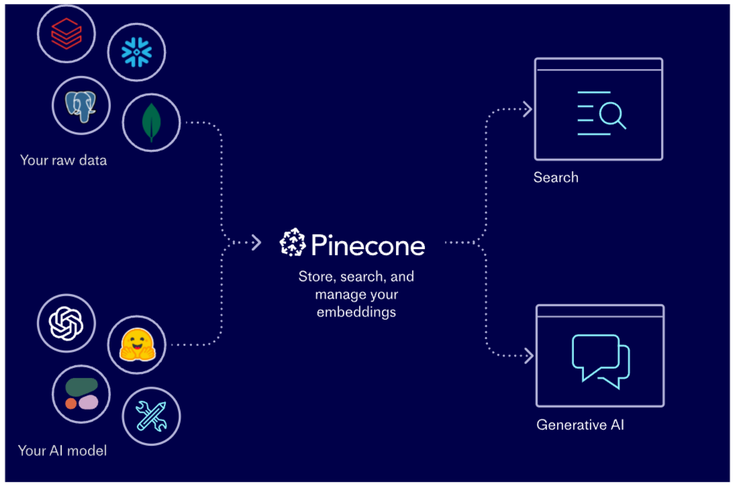
A short edit here needed to happen. I've reviewed this together with Jeff (THANKS) and he shed light on how most of these issues got addresed in v0.4.

They made an architectural shift in their design, changed the storage engine, the build is now at least 20x faster (which now makes my point invalid), requires way less memory (that was not an issue though, but is good to see) and is far more performand and durable (even in April I did not run into any issues with durability). Overall, as I've predicted before actually getting the message from Jeff, it's been a short time, yet they've made amazing progress. This made me reshuffle the recomendations part a bit and this + the coming changes, make chroma a strong candidate to evaluate. I still need to play around with it more to see the exact progress but seems promising.

So now, most of the stuff that I mentioned in this section, that were relevant at the time of making the choice - April, are not a concern now. Keep this in mind.

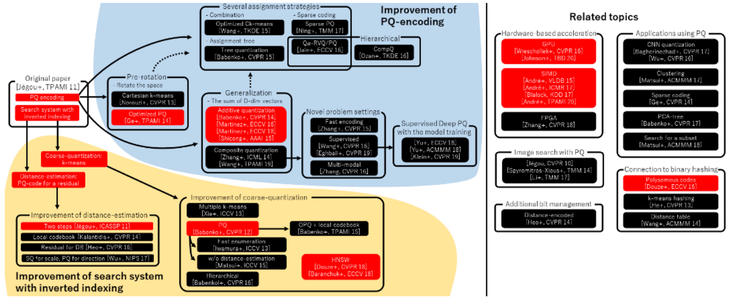
Pinecone

Long story short. It's a black box hosted somewhere, VDB as a Service. Proprietary. No-go. I want control over my data, where my db is deployed, where it lives and so on. Sorry. Privacy & Compliance would not be happy, same with myself.

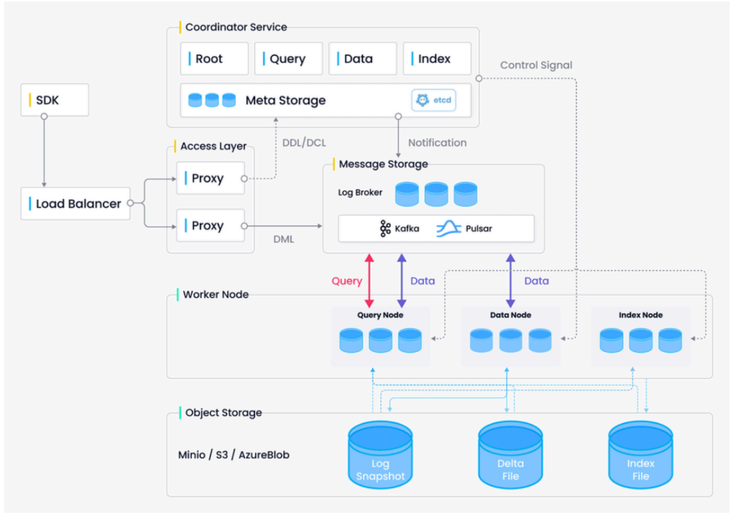


Faiss

It's just a library. not a full fledged VDB. It'd be a step back compared to chroma, so no. But for tiny PoCs? Possible. Evalute it. I didn't do that in detail.



Milvus



This one seemed promising. Initially I thought we'd roll with them. Promised all the new shiny nice stuff, but then I actually started using it. Had troubles to even index the data. Started running into bugs (or documentation not explaining stuff dumb enough for me to get it or not explaining at all), weird edge cases and complications that I was not in for. In case of chroma I just plugged it in, ran my Langchain-based data-ingestion service and I was done. Here? Forget it. However it was farily decent and pleasant to run locally in docker, a bit less so in k8s, but still passable. Bonus points for OS, self-hosted version and on-prem + cloud possibilty. If I was to make a similar comaprison as I did in case of chromadb, I'd say Milvus is the old Oracle DB of Vector Database world. Plus I've heard some similar opinions and unconfirmed rumours regarding ceratin dubious possible origins/practices and policies. It did seem promising but ended up as a failure last minute.

Eventually I did succeed in ingesting the data and so on, but the results were not satisfactory. Performance was okay, but I was done with frustration.

pgvector

pgvector/**pgvector**

Open-source vector similarity search for Postgres

13 Contributors · 10 Issues · 7k Stars · 268 Forks

This one disappointed me greatly.

Bad performance. Unusual selection of the underlying algorithm. Some problems with accuracy, especially when concurrency comes into play. Think carefully if the ease of integration is worth it. In my opinion this + the performance weren't. Take a look at the benchmarks. It's not satisfactory to say the least.

It also turned out that it's not supported everywhere as it's relatively new extension, for postgres standards.

And it's me who says this, guy who is team postgres. But postgres is a great RDBMS. Again, it does not specialise in vectors.

The benefit here is it being a standard piece of infrastructure, having your data in one place, all the other benefits postgres brings. It's still useful piece of software.

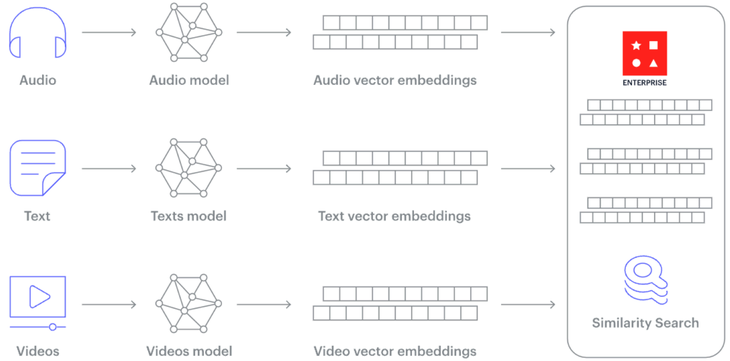
Bear in mind though, that in AWS world, aurora does not support it yet. Meaning if you are in aurora world, you'll need to deploy a separate base postgres instance with pgvector enabled. This eliminates the main benefit.

However if you do not care about that, you are running your own pgvector instance, and just want to get started? Hm, maybe. So there might be some relevant and valid use cases, but not in my case.

The performance was not there, accuracy also, especially when concurrency is considered and throughput. We couldn't deploy it in our Aurora cluster, we'd need to go for separate RDS. I think I'll pass, however it was indeed tempting because of the postgres brand, it being so reliable and common, but let's consider the borader picture - this was 2nd iteration where I expected things to be done in a proper, future-proof manner.

For RDBMS just go postgres. For VDB? Think twice about using pgvector.

Redis



Redis was a strong possible candidate. Good performance. Standard piece of infrastructure. Everyone knows redis, right? Features on the thin side though. I wasn't sure if, given some time, it'd be enough. But I kept it in place. It more or less worked. However It lingered in my mind that Redis is not specialised in Vectors. It's key value store. This is not the core of their business. Sure it'd be nice to be able to just deploy redis and have it done with. Everyone knows redis. However doubts sprouted in my mind, rightfully so eg. given their recent sunsetting of Redis Graph.

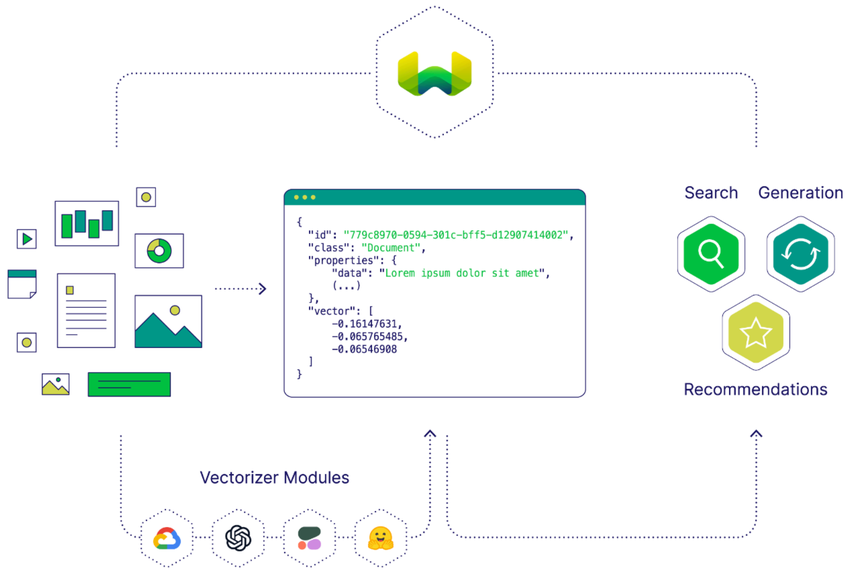
However if you already have a Redis cluster running somewhere, for starting it might be enough.

There is the question of perfromance, sure, but it's acceptable.

So, yeah, viable choice, but make a conscious decision.

Weaviate

Heard some good stuff here, especially regarding the feature set. However did not try it out personally. Compared to qdrant tho, they do have to improve on performance probably, but it might be a valid potential choice. Plus their community seems nice. Or I'm being biased only because @Philip Vollet, so their head of Dev Growth laughed at my joke about pgvector, so he seems salty in similar way to myself, which implicates good stuff. Eh, Olaf and his shenanigans again. Either way. Check it out, should be ok.



Tie it all together

So now, we understand some basic concepts regarding the vectors, embeddings, fundaments of CS, HNSW, ANNS. What Vector Databases do, is to tie this all together, provide more functionalities on top of that and some abstraction layer, so you don't have to reinvent the while, and on top of that, they also take care about the regular stuff that databases do.

What are they? Well, it's mostly stuff related to scalability (so you can easily handle millions of users), persistence (so stuff doesn't get lost), survivability (so it doesn't randomly die easily), data consistency and so on.

It's A LOT of work to do. We don't want to take care about all of these ourselves, do we?

So that's why we let Vector Databases take care of that. It's something they should excell at, other than performance and accuracy, obviously. It's not easy at all.

Also, I think that before you call your product a Database, it's required it meets at least some of these conditions as lately, in the hype driven world, it's not always the case.

Given that, all of the above and certain set of requirements I had for the project, I did the evaluation and research as to which product to actually choose.

Below you'll find a very brief and to the point summary of it. I didn't want to dig deep into the tech details to keep this article digestible, so these will come eventually in another article.

TLDR; Qdrant is the best

qdrant

Product Resources Community Cloud GitHub

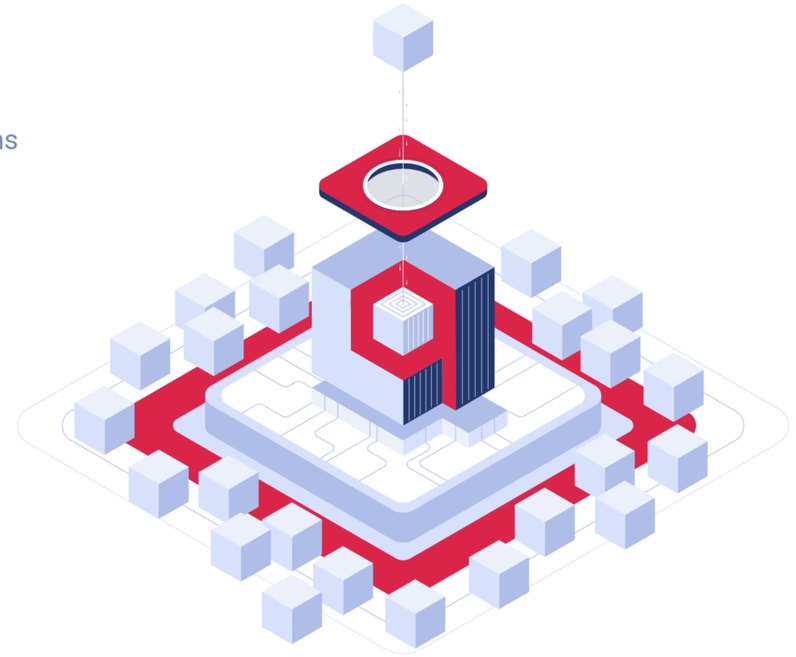
Vector Database

Powering the next generation of AI applications with advanced and high-performant vector similarity search technology.

```
docker pull qdrant/qdrant
docker run -p 6333:6333 qdrant/qdrant
```

Take a look at our Quick Start Guide or build your own neural search with our step-by-step Tutorial

Benchmarks Demos



Loved by everyone

Open source? Check. Self-hosted option? Check. Cloud if you are lazy? Check. K8s? Check. Clustering? Check. Survavibality? Check. Performance? Off the charts. Community? Great. Integration with langchain and other libs? Yup. One-line docker and you good to go locally? You bet. I was sold.

Founders care about enduser

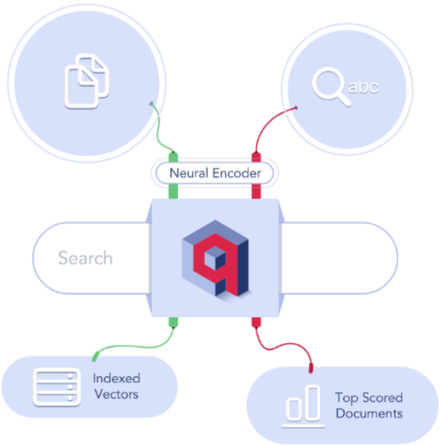
But then it got better. Within hours of signing up for their cloud offering to test things out I was approached by the founders to check in. Big shotout to @Andre Zayarni, @Fabrizio Schmidt and @Andrey Vasnetsov. Keep doing what you doing. We had a meeting set up right after. Tremendous knowledge. I really like the spirit they operate in or the values they highlight in some of their blog articles, especially the one related to closing the seed round. Andre's writing is sharp AF. Snarky remarks about the article not being written by chatGPT get bonus points (btw gonna steal that one).

They offered help (despite me not earning them a dime at that moment and stating that it'd be the case for the coming future or it'll be dime a dollar) with everything. Accommodated to our needs. Shared slack channel if problems arise? There you go. You wanna learn more? Sure, here are the resources. Workshops? Possible.

Similar Image Search

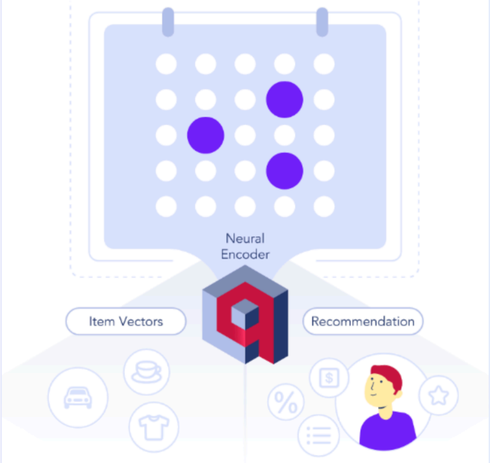
Find similar images, detect duplicates, or even find a picture by text description - all of that you can do with Qdrant vector database. Start with pre-trained models and fine-tune them for better accuracy. Check out our demo!

Semantic Text Search



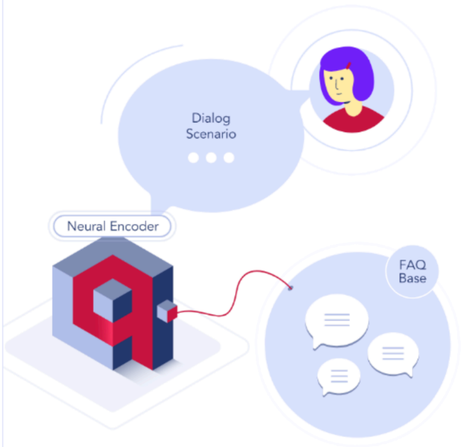
The vector search uses **semantic embeddings** instead of keywords and works best with short texts. With Qdrant, you can build and deploy semantic neural search on your data in minutes. Check out our demo!

Recommendations



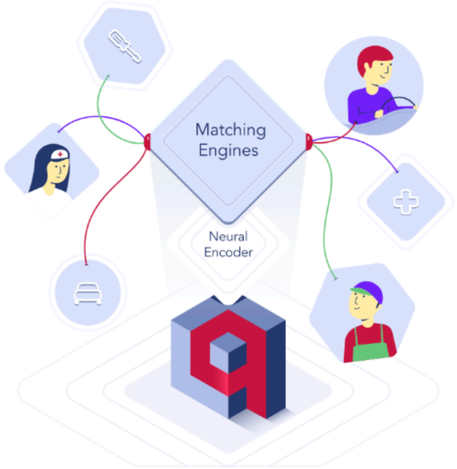
User behavior can be represented as a semantic vector in a similar way as text or images. Vector database allows you to create a real-time recommendation engine. No MapReduce cluster required.

Chat Bots



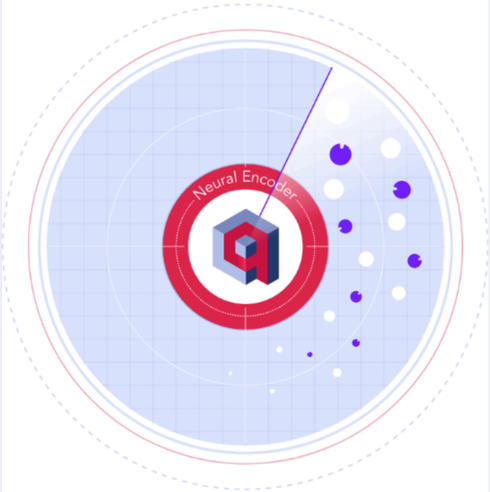
Semantic search for intent detection is the key chatbot technology. In combination with conversation scripts, modern NLP models and Qdrant, it is possible to build an automated FAQ answering system.

Matching Engines



Matching semantically complex objects is a special case of search. Usually a large number of additional conditions are used in matching, which makes Qdrant an ideal tool for building such systems.

Anomalies Detection



Anomaly detection is one of the non-obvious applications of Similarity Learning. However, it has a number of properties that make it an excellent way to approach anomaly detection.

Adventages of Qdrant

Free tier in the cloud offering to test stuff out (that actually can take you far along?) provided to anyone.

Qdrant wins by far, in my experience, when it comes to performance, scalability, durability, ease of use, feature set, flexibility and most importantly community plus the company values.

On top of that you can get started in minutes. It was best performance, easiest to use & set up, well documented, nice community. Clear win.

As you can see I'm totally biased and sold, so take what I wrote with a grain of salt and verify yourself all of the above. I'll however remain quite bullish

Easy to Use API

Provides the OpenAPI v3 specification to generate a client library in almost any programming language. Alternatively utilise ready-made client for Python or other programming languages with additional functionality.

Fast and Accurate

Implement a unique custom modification of the HNSW algorithm for Approximate Nearest Neighbor Search. Search with a State-of-the-Art speed and apply search filters without compromising on results.

Filterable

Support additional payload associated with vectors. Not only stores payload but also allows filter results based on payload values.
Unlike Elasticsearch post-filtering, Qdrant guarantees all relevant vectors are retrieved.

Rich data types

Vector payload supports a large variety of data types and query conditions, including string matching, numerical ranges, geo-locations, and more. Payload filtering conditions allow you to build almost any custom business logic that should work on top of similarity matching.

Distributed

Cloud-native and scales horizontally.
No matter how much data you need to serve - Qdrant can always be used with just the right amount of computational resources.

Efficient

Effectively utilizes your resources. Developed entirely in Rust language, Qdrant implements dynamic query planning and payload data indexing. Hardware-aware builds are also available for Enterprises.

Summary

1. **Qdrant rocks and wins**. In all categories. It's postgres of the VDB world.
2. Weaviate seems to be nice too, but don't quote me on that. Features seem nice and rich for LLMs.
3. Pinecone gets you started in minutes, but so does qdrant and why would you lock yourself in their ecosystem? Plus the performance and price, but might be valid choice for some folks just coz of the Managed mindset and convenience.
4. Chromadb lacks certain features, but develops quickly, however it needs evaluation and observation for bigger projects or situations where we need real clusters. For smaller ones with proper setup I'd say it's a valid choice + look out for their development and upcoming features. A bit like sqlite of the VDB world, but on steroids currently.
5. Redis is acceptable, but doesn't shine IMO. Positive surprise performance wise. It's not core of their business tho, plus they seem to be sunsetting certain parts that do not belong to the core like RedisGraph.
6. pgvector is disappointing, but still can be valid in some use cases and scenarios, it levarage the postgres brand and benefits which also enforce certain limitations on it. Do not count on great performance though or accuracy with concurrency.
7. Milvus is what I'd stay away from. Old Oracle of VDB world.
8. For anything that can hit production, FAISS is a no-go, it's just a lib. For simple play-around hobby projects? Why not.

Yes, this was written by a human in a 4h long flow state powered act of uninterrupted creativity. It was fun, I've actually learned a lot. I've written this in one go almost, except minor stuff or including the feedback from Jeff. I'll leave it mostly unedited, with original wording and typos.

TLDR of TLDR:

1. For bigger data, scalability and performance go for qdrant/weaviate.
2. In some cases you can get away with using FAISS/chromadb and it will still do a good job.
3. If you want to experiment you can implement your very own VDB for simple purposes (or use e.g. ready made from pip - vectordb looking at ya)

https://github.com/jina-ai/vectordb
https://www.linkedin.com/pulse/vector-databases-demystified-part-2-building-your-own-adie-kaye/

https://medium.com/@saptarshichaudhuri/implementing-a-simple-vector-store-and-retrieval-functionality-using-python-f868bb9fff06

https://medium.com/@vidiptvashist/building-a-vector-database-from-scratch-in-python-6bd683ba5171

Lastly, recently some of friends of mine released this handy comparison:

https://superlinked.com/vector-db-comparison

Vector DB Comparison

by Superlinked | Last Updated : Today

Search

Get insights | Give us a star

Vendor	About				Search								Models			APIs				Ops	
	OSS	License	Dev Lang	VSS Launch	Filters	Hybrid Search	Facets	Geo Search	Multi-Vector	Sparse	BM25	Full-Text	Text Model	Image Model	Struct Model	RAG	RecSys	LangChain	LlamaInd...	Managed	Pricing
Activeloo...	✅	MPL 2.0	python c++	2023	✅	-	-	❌	✅	🟨	❌	🟨	🟨	🟨	-	-	-	✅	✅	✅	Free 200G...
Aerospike	❌	Proprietary	java	2024	❌	❌	❌	❌	✅	❌	❌	❌	❌	❌	-	-	-	❌	❌	✅	-
Anari AI	❌	Proprietary	-	2023	-	❌	-	❌	-	-	❌	-	-	-	-	-	-	❌	❌	✅	-
Apache C...	✅	Apache-2.0	java	2023	✅	✅	-	-	-	❌	-	❌	-	-	-	-	-	-	-	-	-
Apache S...	✅	Apache-2.0	java	2022	✅	✅	✅	✅	✅	-	✅	✅	❌	❌	-	❌	✅	-	❌	✅	-
ApertureDB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Azure AI S...	❌	Proprietary	c++	2023	✅	✅	✅	✅	✅	❌	✅	✅	✅	🟨	-	✅	❌	✅	✅	✅	Pricing - A...
Chroma	✅	Apache-2.0	rust python	2022	✅	❌	-	-	❌	❌	❌	🟨	✅	✅	-	-	-	✅	✅	❌	-
ClickHouse	✅	Apache 2.0	c++	2022	✅	❌	✅	✅	✅	❌	❌	✅	🟨	🟨	-	❌	❌	✅	✅	✅	Clickhous...
Couchbase	🟨	Pr	c++ erlang	2024	✅	✅	✅	✅	✅			✅	❌	❌	❌	❌		✅	✅	✅	Couchbas...
CrateDB	✅	Apache 2.0	java	2023	✅	-	✅	✅	✅	-	✅	✅	-	-	-	-	-	✅	-	✅	CrateDB P...
DataStax ...	❌	Proprietary	java go	2023	✅	✅	✅	✅	❌	❌	❌	✅	✅	❌	-	✅	-	✅	✅	✅	https://ww...
Elasticsea...	❌	Elastic License	java	2021	✅	✅	✅	✅	✅	✅	✅	✅	✅	-	-	-	-	✅	✅	✅	-
Epsilla	✅	GPL-3.0	c++	2023	✅	✅	-	-	✅	✅	-	-	✅	-	-	✅	-	✅	✅	✅	Storage: $...
GCP Verte...	❌	-	-	2021	✅	❌	-	-	❌	❌	❌	❌	❌	-	-	-	-	✅	❌	✅	-
Hyperspace	❌	Proprietary	python jav	2023	✅	✅	🟨	✅	✅	-	❌	🟨	❌	❌	-	-	-	-	-	✅	Per usage
KDB.AI	❌	Proprietary	python	2023	✅	✅	🟨	-	-	✅	✅	-	-	-	-	-	-	✅	✅	✅	Cloud = Free: 4 G...
LanceDB	✅	Apache-2.0	rust	2023	✅	✅	-	-	❌	-	✅	✅	✅	-	-	-	-	✅	✅	✅	-
Marqo	✅	Apache-2.0	python	2022	✅	✅	-	-	✅	❌	✅	❌	✅	✅	-	-	-	✅	❌	✅	Pricing - ...
Meilisearch	✅	MIT	rust	2023	✅	✅	✅	✅	✅	❌	🟨	✅	✅	❌	-	-	-	✅	❌	✅	Meilisearc...
Milvus	✅	Apache-2.0	go c++	2019	✅	✅	-	❌	✅	✅	✅	✅	✅	-	-	-	-	✅	✅	✅	Zilliz Cloud
MongoDB...	❌	SSPL	c++ java	2023	✅	✅	✅	✅	✅	❌	✅	✅	❌	-	-	-	-	✅	✅	✅	MongoDB ...
MyScale	✅	Apache-2.0	c++	2023	✅	✅	✅	✅	✅	❌	✅	✅	✅	❌	❌	❌	❌	✅	✅	✅	MyScale P...
Neo4j	❌	Propr	java scala	2023	✅	✅	❌	✅	✅	❌	❌	✅	❌	❌	🟨	✅	-	✅	✅	✅	-
Nuclia DB	✅	AGPLv3	rust python	2021	✅	✅	✅	❌	-	❌	✅	✅	-	-	-	✅	❌	✅	❌	✅	-
OpenSear...	✅	Apache-2.0	java	2021	✅	✅	✅	✅	✅	✅	✅	✅	✅	🟨	-	✅	-	✅	✅	✅	-
OramaSe...	✅	Apache-2.0	typescript	2022	✅	❌	✅	✅	❌	❌	✅	✅	✅	-	-	-	-	❌	❌	✅	free plan / ...
ParadeDB	✅	AGPL	rust	2023	✅	✅	-	✅	❌	✅	✅	✅	❌	-	-	-	-	-	-	-	-
pgvecto.rs	✅	Apacl	rust	2023	✅	✅	-	🟨	✅	✅	🟨	✅	-	-	-	-	-	✅	✅	-	-
pgvector	✅	Postg	c	2021	✅	✅	-	✅	✅	✅	✅	✅	❌	-	-	-	-	✅	✅	✅	-
Pinecone	❌	Proprietary	rust	2019	✅	✅	❌	❌	❌	✅	❌	❌	🟨	❌	-	-	✅	✅	✅	✅	full
Qdrant	✅	Apache-2.0	rust	2021	✅	✅	❌	✅	✅	✅	❌	✅	✅	✅	-	✅	✅	✅	✅	✅	Qdrant Pri...
Redis Sea...	🟨	(i) Redis Sourc	c	2021	✅	-	✅	✅	✅	❌	✅	✅	❌	-	-	-	-	✅	✅	✅	-

And with that, finally, DONE!