1:47

Alexey

**This week, we'll talk about building production search systems. We have a special guest today, Daniel. Daniel is an entrepreneurial technologist with a 20 years’ career. He is the co-founder of Superlinked.com. What we saw, VectorHub is one of the projects by Superlinked. It's a machine learning infrastructure startup that helps build information retrieval systems, from recommender engines to enterprise focused LLM apps. Before starting his own company, he was a tech lead for machine learning infra at YouTube Ads. So welcome, Daniel.**

2:25

Daniel

Hey, happy to be here! Thanks for having me!

# Daniel’s background

2:29

Alexey

**Before we go into our main topic of building search systems, let's start with your background. Can you tell us about your career journey so far?**

2:40

Daniel

Yes. Very happy to have this kind of… 20 year time horizon is maybe a little bit of an exaggeration. But… Basically, I'm originally from Slovakia, with this kind of typical Eastern European technical background of coding competitions in high school. I've been through a couple of internships during university with Google and IBM Research. Eventually, I decided to start my own company, the first one, out of university, because both of those companies were very big, and it took months to launch new things. I started my first startup already after I moved to Switzerland.

Then, I eventually ended up back at Google, as you said, in YouTube Ads, building systems that predict ad campaign performance. When people buy these ad campaigns, they like to get feedback of “Okay, if you run this campaign, you'll get 1000 clicks,” or something like that. Then they see the forecast, they tweak it. And then once they are happy with the forecast, they bite, and eventually, the systems I was working on powered (or power) all of YouTube ad buying. There is reservation [when it comes to] the auction ads. You know, the ads world is not so transparent for many people, and there are a lot of rabbit holes you can fall into.

Then towards the end of 2018, I left, and started working on my second startup. We went through many different ideas, and a couple of years into that process, we landed on Superlinked. That's the project, or company, I'm working on now.

4:43

Alexey

**I loved how casually you dropped this, “Oh, yeah – I just have this typical Eastern European background where people do competitive programming at school.” [chuckles] Seriously, did you do this at school?**

4:59

Daniel

Yes. In, I think, about the second year of high school, I had this realization that if I don't do something like competitive programming, it will be very difficult to leave the country and work on some interesting problems. So I always get some math kinds of competitions but, basically, programming seemed like the thing [that can help] you can actually get a job, right? So I decided to take it more seriously.

Eventually, it did land me those internships. But it was a few years kind of journey of getting up at 3am and doing TopCoder competitions that were US time-friendly. Also meeting lots of people who eventually ended up all in California optimizing the ads with me, in one way or another. So that cohort of the competitive programming folks is a small one, and I keep running into those people, still. It’s fun.

6:10

Alexey

**It's probably still easy for you to pass all these programming interviews at Google, Meta and the like, right?**

6:20

Daniel

Well, luckily, I didn't have to do it for quite a while in that sense. But you may be surprised at how many of such problems I encounter quite often, now that we work with an infrastructure-focused product. All of those tasks that for many people are maybe hard to connect to real work – doing some rotations of three data structures and things like that. Actually, if you work on infrastructure and these hard problems, you’ll find many applications for those types of things. Like dynamic programming, for example, has always been a little bit out there in terms of applications. But actually, now we have this chunking problem as a part of the search problem overall, and I find certain types of chunking can be solved by dynamic programming. So it all kind of comes together in the end, which is funny, too.

What is search?

7:35

Alexey

**I added a note because this is something I want to ask you, but probably towards the end, because we have quite a few questions that we need to cover first. Since we will talk about production search systems today, what actually is search?**

7:54

Daniel

[chuckles] Depending on how philosophical you want to get here…

7:57

Alexey

**Maybe not too much. Not philosophical.**

8:00

Daniel

One way you can frame search is actually as a decision problem. You have a lot of information somewhere, and you want to decide which pieces of that information matter in a given situation – which are the most relevant pieces of that information? That's the first decision and then you're typically navigating some broader problem. Say somebody wants to buy something on the internet, or discover their next article to read, or you are looking for a machine part that is required to fix a machine. All of those situations probably contain an information retrieval sub problem in there somewhere. Isolating relevant data within a bigger pile of data, and then we can talk about what is “relevant” and what is a “pile”, right?

# Search vs information retrieval system

9:04

Alexey

**I asked you about search, and in your answer, you mentioned an information retrieval system, which is like the same thing, right?**

9:10

Daniel

Basically. By the way, these boundaries are often quite arbitrary. [Boundaries] of where recommender systems start and personalized search ends, for example. Or now we have retrieval augmented generation – how is that any different at all? I think that we try to put some brackets of functionality just to make it easier to talk about it but, at the end of the day, information retrieval is the common name for the field. Maybe one other kind of keyword when people are searching for learning materials, I would say, is “representation learning”. Right? That's the domain of machine learning where we are trying to build models that help us encode this information that we are trying to find or search through in a way that makes it more efficient to search it.

10:13

Alexey

**I think, in Berlin, there is a conference called Haystack, which is about search. I think that the reason they use it is because you have a pile of hay – there is a needle there, and you want to find it, right? This is the search problem. So this pile of information, this is what you have (like the internet) and then the needle is what you want to find. Because you can’t sift through the entire pile, you need a smart method of finding the needle, like a magnet or whatever.**

10:45

Daniel

Yes. Because usually there is the constraint of a finite amount of time. If we had forever, then we could always go through the whole pile and very carefully look at each blade of grass. But typically, there is a constraint of, as they say, “every 10 milliseconds added to a shopping interface significantly impacts the revenue.” These kinds of situations repeat across many domains. So latency is a big factor. Search existed – the problem existed – since the beginning of computer science, right? These concepts are not new at all.

# Vector search

11:29

Alexey

**I think there is a paper about vector spaces, which is from like the 70s, where they explain this bag of words. I wanted to talk about… There is a very good book, which is called Introduction to Information Retrieval. I think one of the authors is Christopher Manning. This is a very good book. This is one of the books I started my journey into NLP with. It is super well-written.**

**There, the idea is that you have a document. The document contains words. And then you have a phrase. Let's say the document is about… I don't know, search. Then we look for the word “search”. Then it's like, “Okay, this document contains the word ‘search’ 1000 times. Must be relevant. Let's show it.” So the book is mostly about text search. Yet, today, we talk so much about vector search, vector embeddings, and all of that. What are these things? How are they related? How is this vector search relevant to what I mentioned – to the text search?**

12:45

Daniel

So if we talk about the search world before vectors became very popular – and they have always been there. I think the difference is, “Have they been used in production systems?” Or “Did people keep production systems a bit simpler, until recently?” Which I think is the case. For the previous generation of search systems, you tried to build data structures that helped you find… There is always a concept of query, and then you want to do things to this query – to prepare it for the evaluation. You might expand the synonyms, you might rewrite the query to make it more efficient to evaluate, and you will end with some object that describes your objective – like, “What do you want?” Then, on the search index side – there is this whole field of infrastructure work, in which you have your haystack – you want to process it, you want to ingest it, and build some kind of data structure on top of it that we call index. [Index] makes it very efficient to take that query object and match it against the whole haystack but as fast as possible.

So three vectors – the core idea of that index structure was this “reverse list,” where you basically have a list of keywords with pointers to where those keywords appear in the original query. Any of those keywords can be parts of the words, they can be all keywords, they can be normalized in different ways. But that's kind of the basic idea. You make a kind of dictionary, and then you use that to match the queries to the text. The obvious problem is the brittleness of the setup because you rely on a very specific handcrafted heuristic of how to create the dictionary and what to do to those queries to match the two sides together. By the way, this is still in the realm of the underlying retrieval step, right.

We should probably also mention that there are usually two distinct steps in each of these search systems. You have the initial retrieval step – you can call it “candidate generation,” where you want to quickly figure out, “Okay, here are the ‘potential’ good results.” You are narrowing the whole haystack to some tiny fraction of it. And then you have the second step of ranking. People consider these problems very differently. In the ranking step, in its final, most sophisticated form, you can think about it as a machine learning problem that's usually framed as, “Okay, for this query, and this potential candidate result, what is the probability that they actually match?” It could be, “What is the probability that the user that's doing the search actually clicks on this particular result?” There are different ways to frame the problem. But this part sounds more like machine learning, while the first part sounds more like engineering. So we had a, in a way, stupid retrieval and smart ranking.

Then in the ranking step is usually where those models get complicated, you have all the MLOps issues. That's where you bring all that context into the picture, and you try to run that model on all the candidates, and reorder them and serve that to the user, or to the system that's doing the searching. So that's the anatomy – you have those two parts, usually.

16:45

Alexey

**I have this book. For those who are listening to this as a recording and don't see – this is a German grammar book. So there is a lot of information. Let's say I want to find something in it. How would we build a search system for that? We need two steps, right? You mentioned candidate generation and binary classification, but even before that, we need to index the entire book – build the index. How would we go about that, step-by-step?**

# Index building

17:21

Daniel

Yeah. You basically… Actually, at the end of the book, you would most likely fetch a lookup table. Maybe call the reference? Or maybe this one doesn't have it, but…

17:33

Alexey

**It does have it. It only has a table of contents. But some books do. Yeah.**

17:40

Daniel

Some books do. I mean, the practical answer is – use Leucine. There are big, open source projects out there that help you solve this problem of ingesting a whole bunch of documents, cutting them up, and building that index structure. And these things are built into databases now. There also exists the standalone solutions. You should really, really not build reverse keyword lookup systems [cross-talk]

18:19

Alexey

**I actually found it. In German it’s called “register index”. And then for “aber” which means “but” – it says to go to page 108. For the word “gigen” go to page 84. For “noch nicht” go to page 126.**

18:39

Daniel

Right. You will notice that an interesting aspect of this page is that these words are usually ordered alphabetically. Right?

18:47

Alexey

**They are, yeah.**

18:49

Daniel

Yeah. So that's a very basic way to make it easy to find a word in the list. You can basically binary search for an index in that list if that's the data structure that works for a person looking at the book. Normally, you would have some kind of tree sector where you go down a tree, following the letters. You would have the first letter and then the second letter, and as you go through the Word, you follow down the tree, and eventually the leaf of the tree (the node where you end) would have a list of places where the keyword matching that prefix that took you to that node exists in the original data. And then the whole game is, “How do you keep this thing updated when you ingest new documents?” But again, Leucine is your friend for this. Okay, so we talked a little bit about the first part of your question – if you parachute into the 2000s… [cross-talk]

# Increased complexity in indexing

20:00

Alexey

**For this book, just use Lucene, right?**

20:02

Daniel

Yes. And now the question is, “Okay, why do we need something new?” The deficiency of this system is… I would focus on maybe two elements. One is that it's brittle. It relies on very specific forms of these keywords appearing both in the query and in the original document. Even though you may have expanded the synonyms, the practical reality of managing a search system in production is that you have very many special cases and very long configuration files, that helps you…

20:37

Alexey

**I want to use English to search for something in the German book, right?**

20:42

Daniel

I mean, that's a whole other type of problem. You could consider the synonym expansion to maybe go cross-language. But there is always a set of queries that you will find in your logs, where you didn't return any results, for example – or returned the wrong results. So the day-to-day life of a search relevance engineer is to look at that log, somehow figure out which type of queries make the most impact and are still handled poorly, and then you go when you try to create the rules, and you try to address that problem.

This increases the complexity of your system. Then this goes forward, and then this person quits, and a new one joins, and sees all the rules. It's layers of complexity. So that's the first problem. It's very brittle and very heuristic-based. There is the other side of the problem, which is, “Okay, but in reality, we rarely just have text.” We rarely have just documents.

21:53

Alexey

**Pictures?**

21:55

Daniel

There are pictures there. There are… If you imagine a database of an enterprise company with hundreds of columns (sitting somewhere in MySQL, or Postgres) that this company literally runs on. It's a critical table. Some of these columns will be strings, probably. But there'll be all kinds of other things in there. And then you have your data warehouse with all the logs, generated by our infrastructure, by your users. So that's the data reality. And then you somehow want to do retrieval that uses all this data. This is the second part of the problem, “How do you go beyond just matching some strings to strings?” First, the specific way you often encounter this is personalization. We have some users, we have some data about these users, and maybe they send us a search query, or maybe they just show up on the website or in the app? How do we show them the product they want to buy or the document they want to read – all of this stuff?

How do we combine the behavioral data with the content? This typically happens in the ranking step. So that's why it's so complicated. You have all the kinds of machine learning problems there. I would say there's the state of the world, and now we can kind of switch to vectors and talk about the difference. “What's going on there?” Right? I would say that the first problem that gets addressed is this brittleness. How do we make this problem of matching the query to the index object more robust? When we say a manager in the document says “leader” there will be a match. Yes, you can handle this specific case with the synonym expansion, but there are basically infinite such cases.

So how do we make it more robust? The idea is that, instead of trying to figure out all the possible rules in which we can match the words, what about we come up with some representation that will exist in the middle of those two kinds of documents and queries, where we will project both sides into this shared representation – and this projection will be more robust. That's basically what embedding models helps us do – we do this kind of projection, such that, when things on the input of the projection are similar (for some definition of similar) then they will land in a similar place as that representation. This representation is vectors.

In a way, vectors make that initial matching candidate retrieval problem more robust. That then scales across modalities, because it turns out that we can index images on one side into this representation. And then from the other side, we still embed queries – text queries. So now we're matching text to image, somehow, through this common place in the middle. In principle, you can do this for anything. In principle, you can take all kinds of data on the left side, all kinds of contexts on the right side (not just text queries, but also the history of the user – whatever it might be) and you can kind of have a model that encodes or… one model for the left side, one model for right side, and then they encode those two pieces in the middle, and then you do the matching.

The whole hype around vector databases comes from this – that matching and doing it very efficiently seems pretty important. That also kind of helps you understand that “Okay, in this new world of vector-based search, or dense vector-based search, there will probably be two main problems.” One is, “How do you make vectors from data, such that those vectors represent the different properties of your data that you care about?” And then, “How do you index and match those vectors very quickly?” So there'll be some compute problem and there'll be some kind of search/database problem.

And then, broadly, is how I think people should think about the space. Just to finish that thought, we mentioned my newest startup that a bunch of clever people and I have been working on for the last couple years – we work on the compute problem. We're not building a vector database. We actually work with vector databases. The idea is that, together, we can solve those two parts of the problem and then the end client gets the solution that both can do the compute and can do the search.

27:21

Alexey

**What do you actually mean by “the compute”? Maybe I'll take a bit of a step back because there was quite a lot of information, and I want to make sure I understood it. If I go back to my German grammar book example. Previously, we would index each page or maybe each part, each section of the book with a word index. So we put this into Lucene, and then we would have a bunch of rules.**

**But basically, for each word that I have on the page here, we would have a link to that page (or to that section). So if I’m interested in the word “but (aber)” then I know that I need to go to section two of the book. That works up to some point when there are so many rules – synonyms and all that. For example, what if I want to use English to search for German? Or Russian? Or Slovakian? We cannot infinitely expand our index to include all these synonyms and other languages.**

28:34

Daniel

For example, you would want to search for swear words. Somebody comes and says, “Okay, what are the swear words?” That concept doesn't necessarily exist in your index, but it exists in how we understand languages. So you need some strategy for handling that query and that's where I think it becomes quite obvious that you can’t anticipate all these questions that people might be asking.

# Compute in relation to vectors

29:00

Alexey

**So then we can come up with some sort of numerical representation for each word or document. Basically, each document becomes a large array of numbers right, such that, if two things are similar then the numbers are similar. For example, I have a query which is “but” in English, and then I have a section/unit in my book that talks about prepositions right. They would have similar representation, right? [Daniel agrees] So then we have a different way of looking for information – each word (each document) is a sequence of numbers (an array of numbers) and then we use vector databases to store the documents. And then the document would say, “Okay, you need to go to page 110 to read about that.” So this is how we would do things now with vector databases.**

**And then you mentioned “compute”. Okay, I understand what a vector database is – this is a thing that stores vectors. Then I have a vector and I say, “Okay, I need to find top 10 vectors that are similar to this vector. Give them to me. This is what the vector database is doing. But what exactly is the “compute” that you mentioned?**

30:22

Daniel

Right. So you have two places where you are running some models. The first place is the ingestion into the vector database. You have some document somewhere, and you need to compute all the vectors that correspond to these documents. Maybe when the documents change, you need to recompute these vectors. And maybe, when you want to use the submitted data of these documents, for example, “Which documents are people actually clicking on?” Or “When were these documents created?” This creates some kind of data landscape on the input. And then you have your vector database on the “destination” [side] and we somehow need to connect these two sides.

So there'll be some data engineering work happening – some kind of pipeline work. That's half of the problem. The other half of the problem is the modeling work, like, “Which kinds of models are we running on this data? How are we rolling out new model versions? Do we need to recompute the database when we do that? Or not, or partially?” So you have this ingestion problem, where there is a big compute component – basically running models on some data. And then there is the query handling path, where you have, let's say, your user who put in a query and you need to also turn that into a vector so that the database can match those two vectors against each other – the document part vector with the query vector. In both of those pathways, you basically need to construct vectors from some inputs.

Of course, these are different requirements because for the ingestion, you can maybe batch these workloads. For the query handling, we maybe want to be fast. But you always need to be consistent because you are landing into the same vector space since you want to be matching those two sides together. So those are the two instances of what we call the “vector compute problem”.

32:43

Alexey

**Yeah. So if I talk about a specific example – if I can think about a specific example… There is a model from Open AI called CLIP. What this model can do is turn text into vectors, and images into vectors, in such a way that you can use text to look for images. You can just write “black cat” and then you would get images of black cats, right?**

33:11

Daniel

That's right.

# Embedding strategies and hybrid search

33:13

Alexey

**Let's say we use some way of embedding – let's say we use BERT for embedding our book (for embedding all the words, creating this vector, and indexing). But then we heard about CLIP and we thought, “Okay, we also have images in the book. Now we want to switch our embedding strategy. We want to use a different model for embedding. Instead of rewriting the whole pipeline – if we used a special framework for creating this pipeline, for indexing, re-indexing, and all that – for us, replacing one model with another and adding images would be much easier if used that framework, right?**

34:00

Daniel

Yeah, that's exactly right. We are still at the basic level of this problem, because you used an example where we just replaced the old model with a new one. The thing is that, in practice, it is not so easy. At some point, basically, the data that you want to process in this way just starts to not just be one big string, or one image. You start to have these… Let's say your product manager comes in and says, “Hey, the search that we have built for our news website – when I type “car,” I'm getting results which are too old. We are a news website; we need to have fresh results.” What do you do?

One of the concepts of how people deal with this kind of stuff is called “hybrid search”. You start to combine, “Okay, I'll have my vector similarity search, and I’ll layer on top of it, some other constraints.” And I say, “Okay, pre-filter all the news articles that are newer than one month, and then, within those, match with the vector proximity to the query.” Now, the problem with that, is that – what if there is a super relevant article that's 32 days old? You will miss it. And then, of course, there are many such instances. The product managers are creative – they come up with all kinds of constraints.

If you layer all of these in this classic “waterfall of constraints” type of model, it will overconstrain your results and it will ultimately not work for the end user, actually. What the end user is looking for is some combination – some compromise. “I want kind of new stuff, but also relevant, (and probably a good idea from the recommender engine side of things) some of these results should be quite popular – or maybe popular for people like me.” And then you get into this complicated and also real world… [cross-talk]

36:19

Alexey

**Has to be popular, right?**

36:21

Daniel

Yeah, yeah. So this is the real world, right? Yes, you start with this, “Okay, let's embed some text. Let's embed the query. Let's match the two.” Maybe we’ll use another language model to reorder the results because that can then be refining the match between the query and the document. That whole system will still just take the text part of the data into account and, as I said in my examples, you very quickly run out of the levers to actually get to a result that you can run in production so that it can power a significant part of your product at scale.

What people do then is they go custom, like, “Okay, custom embedding models, custom ranking models, PyTorch.” There’s all the associated MLOps problems again. “How do we have enough data? How do we train this thing? Should we train embeddings for each retrieval task separately, or have some general embedding at the end, and then maybe have a ranking model separate for the use cases?” And that's where we're looking at a six to nine months project with some ML/Data Science folks.

That is precisely the point where you should come talk to us, because we have a way to embed complicated data. But we productized that process. Our goal is to make it much easier to deal with those more complicated situations and make it not take nine months. [chuckles]

38:11

Alexey

**You said that it's possible that you have multiple embeddings for a single document. So there could be embedding for titles, embedding for content, embedding for images, embedding for some parameters, and then you may have five embeddings and as a lot of extra meta information, like popularity, tags, whether a user clicked or clicked similar items – all this kind of stuff. And then it's all there in the database or databases, and you need to link it together somehow.**

38:50

Daniel

Exactly. Exactly. Ideally, at the end of the day, for your articles (or pieces of the articles and your users) you have one vector each, and this vector encodes everything you know – all the information that you have about your articles (all of the stuff you listed) somehow needs to eventually make it into the article chunk vector. And everything you know about your user needs to make it into the user preference vector. Then the question is “How?” Then you have the ETL problem of “make it in there” in terms of getting the data from wherever it is out and into your processing pipeline. And then you have the modeling problem of “Okay, how do we deal with those clicks, those categories, those separate vectors – how can we bring it all together?”

# Embeddings in relation to queries and vectors

39:53

Alexey

**Yeah. I know that in Lucene… We talked about this problem of recency, right? So what if we are a news website? This means that we want to show something that is recent. But what if there is a super relevant article related to my search that is more than a one month old? And I know that in Lucene, there are these types of query filters “should” and “must”. You can say, “The article *must* be less than one month old.” And then it would just filter all the old articles completely. Or you can say “it *should*” and then if it's super relevant, Lucene would still bring it up. But when it comes to vector databases, I don't know if they can have this sort of functionality. Right? Does it mean we need to always have multiple databases when we want to do things like that?**

40:48

Daniel

That's an interesting question. Our view on this problem is that… We believe that you can basically replicate a lot of that “should” type of functionality in pure vector form. You can basically say, “Hey, I want relevant results towards this query, and biases towards the type of stuff the user clicked on before and also biases it towards popular stuff and recent stuff, with these kinds of weights (for example).” Or you can tune the weights with an added model, for example. And you can express these types of queries purely in the embedding, such that… [cross-talk]

41:34

Alexey

**But how do you do it with a date – with recency? Let's say we have a model that embeds a document and somehow contains the recency information. But in one month, it will no longer be recent. Does that mean we will need to always recalculate the vectors for the old.**

41:56

Daniel

If you do it naively, then yes, you will. So that's a bad idea. But there is a way to encode a timestamp into couple of vector dimensions, such that, when you do cosine similarity between two such encoded timestamps – it behaves like a normal time delta. Because that similarity is basically angled between vectors. There is math, which is – by the way, spoiler alert for the people listening – the math is somewhat similar to how the transformer model that's positional encoding.

So when the transformer model eats a big string, the innovation that is happening there in parallel – it eats all the words in parallel instead of a sequence, like LSTMs – but if it only ate embeddings of all the words on the input in parallel, it would lose the sequence information. It would no longer understand which order the words came in. The transformer architecture solves this with a trick called positional encoding, where you basically add into the same set of dimensions information of the ordering. This is like a little bit crazy, because you literally add it into the same set of dimensions – the translation is that you basically move (perturb) each word in the semantic space with some kind of delta, and then the model (the next layers of the model) disentangle this information somehow. But we do it using a separate set of dimensions. Literally, dimension-wise concatenate all the signals into one big vector for content and users and other entities.

But yeah, these are the types of puzzles that you have to solve when you decide, “Okay, I want to express these complicated objectives and these complicated data objects purely in the vector form.” Each new property type will generate this kind of puzzle. Or then you go completely custom, like, “Let's just make a custom embedding model. Let's feature-engineer all these inputs. Let's train a model that encodes all this data into embedding. And let's figure out how to constrain and train the model.” And you kind of go that way.

# Knowing when to implement weights and biases

44:36

Alexey

**Yeah, interesting. Basically, the summary is that you can encode the timestamp also in vector form. Then the similarity between now and the timestamp in the past gives a sense of recency. Right? [Daniel agrees] Then you can also prioritize recent articles if it makes sense. Or prioritize relevancy if it makes sense. Right? [Daniel agrees] The model would be smart enough to figure out what is more [important]. Because I guess there will be one vector, and then one part of this vector is recency, and one part is relevancy.**

45:11

Daniel

The key observation is to normalize all these components. When you index any kind of data, you want to do this as bias-free as possible. This means that you will not be recomputing the index matches when you find your favorite biases. You want to postpone the decision of, “Which signal matters how much in which context” as late as possible – at the query time, ideally. In our system, basically, when we embed these complicated entities, we normalize those components and then, when you use RSDK to formulate those queries, that's where you can start applying weights. That's where you can also start to train the weights. Because in different contexts, the weights will be different. Your landing page is probably different from your “for you” page, and a category page. Obviously, this depends very much on the use case as well.

# LLM implementation strategies

46:18

Alexey

**Speaking of this, I'm thinking about ChatGPT. I know GPTs don't have this information about the time. So if you say… You somehow need to be explicit in your prompt and you say, “Today is this day.” Then you add a bunch of articles and say, “When you answer my question, keep in mind that today's this day, and the timestamps you have in the prompt are these [days].”**

**And then it can figure out the answer. For example, we haven't talked about that, but in DataTalks.Club, we have a bot – one of the community members, Alex, created this chatbot for our courses, to help students find the answers. We have long FAQ documents, which are very hard to use to find things. So we have a bot that answers questions. And the prompt, what Alex is doing is – it says, “Today's this day and keep that in mind when answering questions, right?” When somebody asks, “Can I still get enrolled in the course?” or something like that, it knows. I think it's similar to what you said. Right?**

47:37

Daniel

Right. This kind of thought of, basically, stringifying timestamps and then eating them with a language model is within the broader bracket of thoughts of, “Hey, let's string the five things and encode them with the LLM.” This has limitations, because the underlying model doesn't have exactly the same understanding as you or me of how timestamps increment. There can be surprising results of holes, for example, or misordering.

I think the main problem of “Yeah, let's just take this complicated entity (user) with all the history, make a big string and run it through a language model,” is that you lose the control. You don't get to say how important things are. You also lose the efficiency of using specialized models to encode subsets of the data. Because there are separate research fields in how you turn graphs vectors into vectors, or how to turn time series into vectors, or other types of data.

They are dedicated models for doing that for different data types, and if you try to do the whole thing with an LLM, it’s computationally inefficient, hard to control, and the resulting retrieval quality won’t be as good. So I think those are the three main issues.

# Transforming different types of input into vectors

49:19

Alexey

**I see an interesting question from Demetrios. Demetrios is asking if you have any publications that go into detail about the approaches you described on how to combine various signals into a single vector.**

49:36

Daniel

We have a few pieces that you can understand – between tutorial and research exploration on vector hub. So if you go to hub.superlink.com (and I think you'll also include the link in the notes) we already have an article out there that illustrates how to combine graph and text embeddings together, and also image and text embeddings. Hopefully, that can serve as inspiration.

50:11

Alexey

**I found one on retrieval from image and text modalities. Is this the one you’re referring to?**

50:16

Daniel

Yeah. There is that one and, also, another article that's worth checking out, called Representation Learning on Graphs – or something like that.

50:25

Alexey

**Yeah, I can see it. Representation Learning on Graph Structured Data. You see this in the navigation panel. It's under the blog section.**

50:41

Daniel

Yeah. And, again, this idea to take structured and unstructured data and put them into a vector is not new, right? In Big Tech, people have been building custom embedding models that combine structured and unstructured information into a shared vector representation for a very long time. The question is now more about, “How do we productionize it? How do we let people quickly experiment, iterate?” We are very close to launching our actual product. We have been private… I don't want to be too salesy, but basically, we have a framework that's coming out that helps you do this.

51:31

Alexey

**I'm looking at the article in the blog post you mentioned – Representation Learning on Graph Structure Data. From what I see, you show how to use PyTorch?**

51:41

Daniel

Yeah. In all of those examples, we use standard tools [that are] out there – SciKit Learn, NumPy, PyTorch. Right now, our goal in VectorHub is to help people learn the techniques and not push our product. There'll be a separate place for describing what our product does and how it relates to all of this. But as you will notice on VectorHub – it's kind of external, contributor-driven. And it's kind of a compromise between entertaining the research thoughts of practitioners out there and steering them towards, “Okay, let's look at vectors and information retrieval.” I think that's just a good place to start learning about the concepts.

# Choosing vector database vendors

52:35

Alexey

**And then I see that you also have a Vector DB Comparison, which is a super relevant thing. Because if you Google, or if you just open any article about LLMs, (or take our interviews as an example) – there are so many different vector databases. From what I understood, this article that you have here (or not article, this comparison database) actually helps us to pick the right database for our use case, right?**

53:10

Daniel

That's right. We crowdsourced this… It's powered by a git repository – the same as VectorHub. And we crowdsource this. Also, we got a bunch of contributions from the vendors. It's basically a feature comparison of vector databases for different types of search constraints and operational questions, “Can I run this in the process with my app? And separately? Is there a managed offering? What is the open source license?” We also have stats now of GitHub stars and NPM pools and Pbin styles and all of that stuff. People sort of look at the table and they think, “Okay, this is way too many offerings in the market.”

But actually, I think that we haven't really seen the full potential that this technology enables. And I think as we go and apply the technology, there'll be a bunch of different specializations and different buckets, where different solutions perform better. Do you want a few big industries versus many small ones? This, alone, is one of those decisions that inspire completely different designs of the underlying system. So I believe there'll be a bunch of these things and, obviously, the incumbent databases, they all basically launch the vector index as well, and there are different trade-offs for that, of course. But yeah, I think that table (vdbs.superlinked.com) is a good starting point for that exploration, I would say.

# Just throwing everything at Lucene

54:56

Alexey

**I see that we have another interesting question from Vishaka. The question is, “Is there any reason why you wouldn't use a database that goes beyond just vector search?” And then I immediately started thinking about databases like Elasticsearch or Lucene, where we can actually combine… We have these “must” and “should” type of queries, we have the inverted index (like the word index), then we have a bunch of other things. Also, in Elasticsearch – I don't remember if you still need to install a plugin, I think now it comes with Lucene. You organically have vector search in all Lucene-based databases. Then you can just use vector search in your database. So why can't we just put everything in Lucene and let it do its magic?**

55:53

Daniel

You might? Maybe that's a good place to start. Because it's right there. The question… I think the considerations break into a few different categories. For a while, it used to be performance – if you have a couple million vectors, the PG vector and other solutions were considered orders of magnitude slower than the dedicated solutions. I think there is still some difference. People should check out if the difference is big enough for them that it matters. So, performance. The other category of thought is, “What is the right set of tools and abstractions around this new type of search?” For example, query language, “To what extent can you tap into…” [cross-talk]

56:43

Alexey

**[inaudible]**

56:44

Daniel

Right.

56:45

Alexey

**Elasticsearch is [inaudible]**

56:47

Daniel

Yeah. People had similar thoughts with graph databases. I think we had some successes and some challenges to learn from, from that era, as well. I would say, the question is, “What are you optimizing for? Are you optimizing for only ever having one database? Or are you optimizing for solving business problems and building stuff that works as fast as possible?” Then, if the new abstraction helps you deliver faster – it still gives you the expressive power you need, but also gets you to the destination faster – then maybe you should look at the tool, right? I think that's kind of the decision space.

57:39

Alexey

**Do you have more time, or do you need to go?**

57:43

Daniel

Yeah, we can keep going for a couple more minutes.

# Choosing vendors for your use case

57:48

Alexey

**Well, maybe this will require more time. The question from Adjay is, “If I'm a midsize D2C (direct to consumer) brand, what would be the best way to build my search tech? I'm looking only to add personalization and switch from pricey third-party vendors.”**

58:11

Daniel

Okay, that's… [chuckles] That’s a big one.

58:14

Alexey

**Yeah. They probably need a consultation, right? [chuckles]**

58:17

Daniel

Also, for any questions that remain unanswered, I think there'll be a link to my LinkedIn – people should connect to me and shoot those questions over. For e-commerce, I think there is a huge opportunity to do real-time personalization across many different surfaces – feeds, category pages, product detail pages, basket pages, personalized emails, [etc.]. In fact, our first production deployment is of this type, and we see large lifts of revenue and so on. So I think there is a big opportunity there. Also, because of the multi-modality of the e-commerce Data, (you often have product images and descriptions and behavioral data) I think there isn't a go-to stack that you should absolutely use.

I think it depends somewhat on the constraints. Look, if you have, let's say, 100,000 data points across all your stuff – behavioral events and products, users, and everything is on the order of 100,000 (a couple hundred thousand) then I would just pull the data into a Python notebook and just kind of see what you can do with basic tools out there. Do some embeddings, do some matching, pull frequent queries that you get on search, see if you can make embeddings of users and cluster them to see if there are some clusters to be exploited. I think you can explore quite a lot in this way. And then, if you are getting results that are dramatically different from what your current production system is doing – you can literally just eyeball this.

For example, the CLIP model that you mentioned from Open AI – I think this is an eye-opener for many people, that they can ingest a bunch of photos of clothes and then they get a search query like “blue t-shirt with short sleeves,” and it actually works. And it differentiates between short sleeves and long sleeves. This feels kind of magical. Most people start this way, right? They create a demo of some queries that are dramatically better than the current system and then they figure out how to productionize that – probably some kind of tightened server, getting all this data on the input, handling those queries, there's probably a vector database (or a vector-enabled traditional database) somewhere in there. I think that's a cool place to start. Maybe we can do one more, and then I'll have to jump.

# In the end, the main metric is USD

61:25

Alexey

**Yeah, well… We have other questions. This one is also big. The question is, “What are some metrics that can be used to monitor search performance?”**

61:41

Daniel

Yeah. I mean, that's… [chuckles] That's a huge one, because performance is very ambiguous, right? Actually, our Chief Architect likes to say that “The main metric that should be used is USD.” And I love this joke because people hear this “mean squared error.” People are trying to figure out what USD stands for? It's dollars, in the end. Right? So high level thoughts on this very long question is – you will get more funding for a project as a data scientist or engineer in your company, if you can connect your metrics to the actual business performance. Then, do A/B testing carefully and intentionally. I mean, there is so much content about this out there that I don't think I can do it justice. But yeah, I think the dollars – that's the delta to most of the content that I see out there. It’s connected to something that the business cares about, and not having 50 Grafana charts that only you care about.

63:06

Alexey

**Yeah. Sometimes it's not immediately possible to calculate the impact in dollars. But sometimes you can have some other business metrics that are important as well. For example, in the company where I used to work, we cared about contacts. It was a marketplace. What we wanted from search is – if somebody is looking for something, then they contact the sellers, right? So this is one of the important things. That, or they click at a certain thing, or order a delivery. These are two metrics that are important. And then for each of these “successful events,” we can attribute some monetary value, right?**

63:50

Daniel

Yes. Those are proxies – proxies for the dollars. That's the only reason that you would care about somebody contacting a seller. Somebody figured out that there is some probability of that leading to a transaction down the line. And then you think about that funnel and those probabilities and “all things being equal,” more clicks probably means more money. Usually, the “all things being equal” does a lot of heavy lifting, because we have experiments that are not fully isolated and all kinds of seasonal effects that upset e-commerce. That's why we run control groups. Search relevance monitoring is definitely… One more thing I'll say on this topic. Having metrics that engineers can affect without going through the data scientist and iterating on them quickly – I think that's interesting. Basically, how can you create metrics that facilitate fast iteration?

Sometimes that could be offline evaluation tests, sometimes it can be A/B tests. But one of my goals (or our goals with Superlinked) is to enable the engineers to solve a lot of these challenges, without going through the data scientist. Because the data scientist is busy and has many problems (and should work on them) but for some of the more basic stuff or clearer stuff, we want to give engineers the levers to explore and still have the power. But, also, [we want to give them] the abstraction that helps them actually navigate the problem, so it feels more like engineering, and less like magic. I think with the pre-trained models, this is one way to understand the current opportunity in the current ML hype wave – engineers can solve information retrieval problems directly. This, I think, will unlock a lot of value.

# Closing

66:19

Alexey

**Sadly, we didn't talk about the algorithms, and competitive programming, and their relevance to everyday work – maybe some other time. Actually, by the way, right behind your head, I see a bluish patch of sky.**

66:38

Daniel

Yeah. Okay, I'll play with your optimism and say that I also see it. But I’m tuning in from London today, and it's the beginning of March, so…

66:48

Alexey

**Yeah, I was going to say maybe you can now go celebrate the sky. [chuckles] Thanks a lot for joining us today. Thanks, everyone, too, for joining us today – and for asking your questions, tuning in. And also, thanks, Superlinked and VectorHub for supporting this podcast interview.**

67:10

Daniel

Thank you, Alexey. Big fan of the community, of the podcast. I actually binged a few episodes just recently. Please keep doing what you're doing. It’s always good to have this kind of engineer-first view. Hopefully, we get to chat soon.