EPISODE 1220

[INTRODUCTION]

[00:00:01] JM: Vectors are the foundational mathematical building blocks of. Machine learning machine learning models must transform input data into vectors to perform their operations creating what is known as a vector embedding. Since data is not stored in vector form, an ML application must perform significant work to transform data in different formats into a form that ML models can understand. This can be computationally expense intensive and hard to scale especially for the higher dimensional vectors used in complex models.

Pinecone is a managed database built specifically for working with vector data. Pinecone is serverless and API-driven, which means engineers and data scientists can focus on building their ML application or performing analysis without worrying about the underlying data infrastructure. Edo Liberty is the founder and CEO of Pinecone. Prior to Pinecone, he led the creation of Amazon SageMaker at AWS. He joins the show today to talk about the fundamental importance of vectors in machine learning, how Pinecone built a vector-centric database and why data infrastructure improvements are key to unlocking the next generation of AI applications.

[INTERVIEW]

[00:01:09] JM: Edo, welcome to the show.

[00:01:10] EL: Hi.

[00:01:10] JM: You've done a lot of work on machine learning platforms, and the question I'd like to start out with is what are the databases that get used with machine learning applications?

[00:01:21] EL: That's very interesting. I mean, depending on where on the cycle you are. So if you're in the collection, inspection of the data, some BI tools, things like Snowflake, things like Databricks, I mean, Spark. And later on for training usually those are done on single machines or collection of machines that are not really used by databases, but then when you get to production you usually use things like pup/sub systems. And what we're building is a vector database that needs to like retrieve and score and rank things in real-time, and those are the kind of infrastructure that are usually involved in the different life cycles of a machine learning product.

[00:01:59] JM: You worked at AWS on the very popular SageMaker product. What did you learn about machine learning infrastructure when you were at AWS?

[00:02:08] EL: Wow! A lot. I've learned a lot in the same time – The field itself changed a lot while I was there. So I think the two main things that changed were, A, machine learning has become a lot more mainstream and engineers and machine learning infrastructure folks are expected to be a lot more machine learning and data savvy and expecting to get stuff to the finish line, which was not the case a few years ago. It was kind of more like the data scientists doing that.

And another thing is that it's one thing to get a machine learning solution to work in production in a company that's ready for it and has been investing in it for years and it's very, very different to do that in a company that is either just starting to flex those muscles or is really not even sure that they want to invest in that direction. And so we've seen customers pretty much across the entire spectrum and it's one of the things we're trying to do at Pinecone is really make the bar a lot lower for a lot of folks who try to do machine learning at scale and in production.

[00:03:12] JM: And in what ways are you trying to make that bar lower?

[00:03:17] EL: Well, one of the core difficulties that we see in deploying large-scale machine learning is that machine learning represents objects as high dimensional vectors. Those are

just lists of numbers, either embeddings of text, or embeddings of images, or feature engineered values, but in the end they're all – If the object that you're using is going into a machine learning model, then it's a high-dimensional vector. That's what it is. You can't do math on anything that's not numbers.

And so machine learning applications like recommendation or anomaly detection or similarity search eventually produce those like hundreds or tens or hundreds of millions of these high-dimensional vectors and they need to store them and search through them and retrieve and rank them and do that at scale and quickly at some cost efficient way. And building something like that is hard. It's really hard. And not only does it take a lot of time. It's also just complicated to get right. Most companies either fail altogether they'll spend years doing it.

And so we take the distributed systems effort and the kind of core vector database effort out of the equation and let companies only deal with the logic and the model and the application that they're trying to build and not layer onto that effort yet another almost insurmountable effort of building the distributed system underneath it.

[00:04:45] JM: So can you leverage some existing database under the hood like Postgres or Mongo or something? Or do you have to build an entire database from scratch?

[00:04:53] EL: The answer is that you do have to build a database from scratch. So if you think about something like Elastic, it's a fantastic tool for text documents and like searching for terms. If you're looking at something like Mongo, it'll be fantastic for JSON and kind of semi-structured records, and they are heavily optimized for those use cases. If you look at high dimensional vectors, they don't have terms and documents. They don't have keys and values. This is a list of, say, 1024 floating point numbers. And what you're searching for isn't – You're not searching with a term or with a filter or with an SQL query. You're searching with other vectors.

For example, give me everything. In some area of space, give me everything that's close to this point or within some angular distance from some direction in space. Those sound may be

abstract to maybe people who are not used to doing that, but if you've done anything that has to do with similarity search, you effectively have been doing k-nearest neighbors, and that is one of those kinds of queries and to be able to build something. So an engine that is able to answer those questions, it's just a very different kind of index and a very different kind of engine, and it does require its own database. Does that make sense?

[00:06:11] JM: It does. It may be worth exploring more the concept of a vector. I think people who are listening who are not super familiar with machine learning may not be as conscious of what a vector is or why vectors are important to machine learning. Can you describe what a vector is and how you want to store vectors?

[00:06:33] EL: Yeah. So I'll start with the very basic, kind of what a vector is in software, just a float array. It's literally an array of floats. That's what it is. Or you can think about a sparse vector. If most of the entries are zero, then maybe there's some more compact way to represent it maybe as a key like index and value. But in the end it's a list of numbers in some dimension, the dimension being the length of the array.

Now that is the input for any machine learning model. No machine learning model takes anything other than that as input. Now, some models seem to be taking text and so on, but that's not what's actually happening behind the scenes. What's happening behind the scenes is if you take something like a text classifier, the first thing it does is convert the text into a vector and then push it into some neural net. So that's just the input type for any model.

And when you speak about machine learning, why is it important and why are we thinking about this in this way? Is because those vectors have meanings, right? If you represent an image as an embedding, if you convert the image into this high-dimensional vector, you can do it in two ways. You can just represent, say, the pixel values themselves as a long list of just values. That is a vector, but it's a very useless representation for a machine learning model because, for example, if you just measure the similarity between two images, then just the difference in pixel intensity isn't a very good indication of similarities.

So like a picture of a tomato and the flag of japan might look really similar, but they're actually conceptually as different as they can be, right? So you want a more semantic representation of it, and computer vision models would give you that. So they would take that image and process it in a way not unlike your visual cortex would in your brain would do it and get a high-dimensional representation such that two slightly different flags of japan would be close to each other and two different tomatoes would be close to each other, but not between the two groups, right?

And so those high-dimensional presentations are incredibly important and that's how machine learning represents and stores data, be it user behavior, or images, or audio, or text. And that's the kind of data that companies are saving. And so if you want to map images, for example, into some high-dimensional space with computer vision and then do semantic similarity search on those images, you would do that in a vector database. You would convert the images. You would give the database your function, your container that contains your computer vision model for converting the images to vectors. The vector database with converter vectors, index those, and in real-time be able to retrieve similar images based on the semantic representation, based on the vector representation or their embedding, and not based on the image itself.

[00:09:40] JM: Okay. You've given a description for why vectors are important for machine learning. Now, how does one build a database around the concept of vectors?

[00:09:51] EL: So as we said before, your goal is to build more accurate and more scalable machine learning applications, right? And so let's take similarity search, which is a fairly simple application. In similarity search you're trying to retrieve similar items, and then items let's talk about, for example, text here. And by similar I don't mean they contain exactly the same words, but rather they roughly mean the same thing. It might be even in different languages.

And so you now need to – So assume you're using your favorite machine learning model, your favorite NLP model to convert a piece of text to a vector. Now you have a 1024 numbers, and the question is what do you do with it? And if you're using Pinecone, what you do with it is literally just update it, like upload it or insert it into the database like a key value where

somebody give it some ID and you stick it in the database. And now the question is how do you retrieve from such a database, right? So what actions can you have? B it doesn't have any keys and values and SQL doesn't mean anything anymore and it doesn't have any terms. And so what queries can you even run against such database?

And so I want to take you on a little detour to explain the notion of similarity in terms of distance in that space and kind of convert between the abstract notion of similarity to a concrete notion of distance in that space. And so when you think about a point on a page, okay, it's like a 2D, like an actual piece of paper, you can put X and Y axes and you can represent every dot on that page just as the X and Y coordinates in two dimensions, right?

So a two-dimensional point is represented by just a set of two numbers. And the same thing is X, Y, Z coordinates. You can put a point in like 3D with three numbers. If you have, say, a thousand numbers, then that is a point in a thousand dimensional space. While you and I both can't quite visualize a thousand dimensions, mathematically it's the same thing. And so in the very much the same way that I can measure distances in two or three dimensions, I can measure Euclidean distances in a thousand dimensions.

And so our model was trained such that similar semantically. So things that are similar to each other. Either similar images or similar documents map to close places, the Euclidean distance. Their actual physical distance in their high-dimensional locations is small. Then now I can represent my similarity search as a very concrete query, which is go to the database and retrieve all the points, all these high-dimensional vectors whose distance from this location in space. Okay, this point is less than some number. Or give me the 100 most closest points to me. Not most similar or closest points to this location in space, and this being my query.

And so you've converted an abstract notion of give me the most similar objects to two stages. One, convert those objects to vectors and then run a vector query in a vector database, okay? And so that pattern, we see that as kind of being de facto the standard pattern both in similarity, but also in ranking in general, anomaly detection and a slew of other applications.

[00:13:26] JM: Now, I remember learning about vector similarity back in college when I was taking a class, it was about like search. And as you described, the ranking, where you can use these vectors to define all kinds of functions and in terms of how they're going to be similar to one another. So that's somebody entering a search query, you could rank all the documents in the database based on that search query. So this is kind of an old concept. But I guess what's new is building an entire database around this, right? Because I think, historically, this has been thought of as more of like something that you perform as an N-memory operation. You maybe take all the documents out of a database and then you turn them into vectors and then you perform operations on them. You're talking about just viewing the whole thing as a data set that is in a database.

[00:14:18] EL: Correct. Right. So re-ranking is used to call – I don't know if they still call it that way. I think they do. It's called different things. So like you say, I can retrieve from any data store, a search engine, or a database, or what have you. I can retrieve candidates. Like, say, the hundred things that I think would score the highest based on whatever keys and words and whatever, some BI. And then I can use machine learning to score them in some way and re-rank them in some arbitrary way, right? So that is very common. That's still doable. And a lot of people follow that pattern because it's easy and they have the databases for it already.

The problem with it really and one of the main reasons for creating something like Pinecone is that the logic that you use to retrieve the top candidates and the logic they use to rank them are completely separate. And they don't actually gel very well. And so I'll give you as an example. We spoke with an online recruiting kind of job placement company that was using some stock search engine to retrieve candidates for job applications, and they told us, I mean, "Hey, if somebody's looking for a job in Manhattan, then they don't want you to show them job results in New Jersey." But if their dream job happens to be in Hoboken, which is literally like one stop away or whatever in Williamsburg, then they would consider it. And if we filter it that way, then we don't find those results. But our machine learning can actually figure out that, yes, this job might be out of bounds in terms of the hard filter, but is otherwise perfect. And so the person would actually enjoy it.

And so they figured that if they could represent their job, their positions as these high-dimensional vectors based on their machine learning, they could do a much better job. But then their search engine didn't work anymore, because now all their applications, all their job openings were these high-dimensional vectors that they couldn't use the search for. So they were kind of in an in-between state where they knew that they could do better had they ranked natively with the machine learning, but they just couldn't do it. And so Pinecone for them was the answer.

[00:16:38] JM: So at this point I am understanding how it can be useful to build Pinecone for some of these applications like ranking and I guess building search indexes. But that's still to me – I guess I'm still confused about the machine learning aspect of it. Maybe I don't understand machine learning well enough to know why it's so useful to have these things in a database as vectors. Could you just go a little bit deeper into maybe another example of machine learning, a machine learning application that's made a lot easier with Pinecone?

[00:17:12] EL: Yeah. So there are plenty. So let's take another customer use case that we can maybe look at. So this is a company that has a ton of service requests, okay? We're talking many, many, many millions of service requests where their customers type in what's happening or they're experiencing some issue and they want to retrieve – Their service folks want to retrieve in real-time. Given some service requests, they want to figure out what it is. What it's similar to and maybe what is the mitigation steps that need to be taken. Okay? So that's a business use case.

The problem is that when you get a service request, it's very hard to figure out which are the most similar other service requests that you should look at. Okay? Because either the words don't match at all, or if they match, they match to like thousands of others. And so you really have no idea what you're looking for. But with machine learning, you can train embeddings of those pieces of text such that semantically very similar requests are actually very close by in high-dimensional space, right? Actually their vectors are much better representations of the similarity between those requests.

So what they do is they convert those strings, those sentences, or short paragraphs, into high-dimensional vectors. They put them in the high-dimensional database, in the vector database. And then in real-time when a service request comes in, it itself is also converted to a high-dimensional vector. You search the vector database, retrieve the 10 most similar requests and bring them back to the person operating the ticket or the service request, right?

Now, what you've gotten out of it is that now those top 10 or top 100 most similar other requests are semantically very similar, it's very relevant, and you've used your machine learning mappings to actually rank them natively, right? And so they don't have to match on any of the words and you didn't have to create some very complex business logic on what makes the request similar. You learn that from your data and then you use a native system for the kind of data that you've learned to retrieve the top results, and you end up getting much more relevant results and end up being able to build the system basically with one or two people in a few weeks instead of like an organizational wide effort to write hundreds of rules and so on, which you otherwise would have had to do. Does that make sense?

[00:19:54] JM: Yeah. That clarifies a lot of things for me. So I think now that we've talked about the high level application for Pinecone, maybe we could talk a little bit about the architecture. So let's first go through like a read and a write to the database. Can you talk about the read and the right API?

[00:20:12] EL: Sure. So, conceptually, Pinecone is built out of two different components. The first of them is the vector index, which is think about as the single container set of libraries, data structures and algorithms that allow you to pack and index high-dimensional vectors such as you can search through them efficiently. And then there is a whole distribution over Kubernetes of potentially hundreds of such containers with all the orchestration that needs to go into sharding, replication, aggregation, re-ranking, like pre-processing and post-processing by machine learning models, and those two are very separate components.

Now when you write to the database – So before you write, you have to set up your service, okay? When you set up your service you define basically what is the model that converts my

images, text, user behavior, whatever my object is. What is the machine learning model that converts that into the high- dimensional vector? You give that as a container or you can skip that and feed vectors directly and you set up kind of the path, right? Say, “Okay. These are the models that I use to create vectors and rank them.” That is the service definition.

Pinecone is a fully-managed service. So once you do that, you just give the definition to our controller and says, “Okay, spin up the service.” It just does it. And after roughly a minute you have a live database. At that point you can start upserting data, and upsert being either update or insert, and you can do that from any machine from anywhere in the world as high-throughput as you can get. If the item is there, it's being updated. If it's not there, it's inserted. What happens is if your definition had the data sharded to several different nodes, whether those nodes are replicated for increased throughput and resilience or not, the shared distribution, the container distribution, Kubernetes takes care of that, and we route the traffic the right indexers.

I want to add that everything is super optimized on our end. So all the communication, everything is GRPC with ZeroMQ. Everything is highly optimized because you want to be very low-latency end-to-end. When you get to the indexes, of course the vectors themselves are inserted into the individual indexes. In read what happens is those requests are also split to the right shards. Each of those gets searched independently. Results are being aggregated, re-ranked with your machine learning ranking model. Or if you don't have it, just re-ranked by the metric like distance or similarity that you have and they are sent back to you. Basically those are the two basic read and write operations, and you can do that, again, from anywhere because it's a managed service. It's up and running for you when – If you're done with it or if you want to run another one, you just spin up another one or close it or do whatever you want with it.

[00:23:09] JM: So that's a great description. I’d love to know more about the distributed systems strategies. So what's the story around availability, reliability and replication?

[00:23:20] EL: Yeah. So when starting out on this route, we knew a lot already about scatter-gather type of technologies and search engines and so on and we figured that a lot of our components are really containerized. And in some sense we can't even expect them to ever not be containerized. So if you have TensorFlow models or PyTorch models, or Scikit-learn models as ranking or as pre-processing, customers might have bespoke code that takes, for example, text and converts it to some other format first. And so we figured that the basic function, the basic building block of this thing has to be a container, and we have to let customers actually inject and use their own containers in different parts of the system. And so this entire database is built off of highly-optimized communication between containers. Some of them are ours and some of them are the customers. So that's one conceptual departure from the standout construction of a database. And we actually have a blog post coming out soon from one of our engineers, Jack Pertschuk, which I'd love to send you when that becomes available on exactly how we've optimized that portion.

With availability, everything is written to write the head-logs. Everything is persisted. When a node goes down, it immediately gets re-spun up and rehydrates. And so if you have one shard, even if one shard is unavailable for a few, I think it's roughly 10 or 20 seconds until it's rehydrated and back up. Then that's roughly the time in which reads would be affected. But if you run two replicas, which we of course allow in the definition of the graph, you won't have that. If you have a node going down for any reason, then the other node, the other replica would take the load, and of course still the other one that went down it still takes about 10 or 20 seconds to rehydrate and come back up.

And so we highly encourage any production grade – People would you usually run their dev or the unit test or whatever. When they develop, they would use like a single shower just to save on resources. And then production usually use like two or three replicas, which give them both resiliency and kind of extra throughput just, just extra oomph just in case.

[00:25:47] JM: What are some of the big architectural challenges that you've encountered in building Pinecone?

[00:25:52] EL: We've seen several things. So first is just the container ecosystem. As much as we have invested in it as a society, as like a community, we feel like it's still very young. There's just a lot of things that we try to do that we either are impossible or somebody like us from the kind of the core of the team to really figure out some internal workings of Kubernetes and Go and find some really creative solutions. So I think that part I think was still breaking ground on that.

And the second part is utilizing very low-level hardware accelerations through containers on public clouds ends up being way more tricky than one would imagine. So, yeah, if you're running on your laptop or on whatever hardware you run, if it's a specific machine, yeah, you can always optimize the crap out of your like bit twiddling and whatever libraries you install and so on. Making that work uniformly well on AWS and GCP, making your entire CI/CD compile everything correctly so you actually do everything right and so on. All of these things are incredibly challenging, and I think that even if you put all the algorithms and kind of all the “fun stuff” aside. Even just the kind of the more pedestrian stuff that you don't necessarily think of as is – I personally didn't think that they would be as challenging they're actually quite hard to get, right?

[00:27:24] JM: Let's ground the conversation a little bit more and revisit something a little bit more basic. Why does it make sense to have a database specifically for vectors? Like I just want to really drive this point home. Like we've been able to develop machine learning models without a database specifically for this function. Why do we need a database for it?

[00:27:49] EL: So the database is not used for training the models. The database is used for serving large-scale applications that use machine learning data. And so you don't need a vector database to train in a language model or an NLP model, right? But you do need to use it for serving similar documents at scale, okay? And why do we need a specialized database? Because that's a very specialized data type, okay? If you have JSON or structure, you should think of Mongo. When you have documents, you think about Elastic. If you have – Different databases would specialize in different workloads and different data types. And a vector

database specializes in high-dimensional vectors and is able to answer those queries very efficiently.

And so if you're looking for similar documents in hundreds of millions of documents and you want to use machine learning for that, well, good luck doing it on anything other than Pinecone. I mean, it's just going to be brutally – Either very difficult or just very slow and inefficient. And so these workloads are so common these days and becoming more common by the day that we decided that a dedicated database for it is the right way to go. And the more we invested in it, the more we understood how right that decision was, because pretty much the entire index and almost everything that we do we had to really rethink some core design decisions of how a database is built to be able to optimize for these workloads. Yeah, I mean, I think the results kind of speak for themselves. The ease of use that you get out of Pinecone today when building these semantic search applications or de-duplication or anomaly detection based on machine learning it's just unparalleled. I mean, we're talking about completing like a production-ready POC in almost leading to production POC in a few days and maybe getting to production in a couple of weeks, which was unheard of. And without it, we're talking about many months at the least and sometimes a year.

[00:30:10] JM: So let's talk a little bit about the key concepts in Pinecone. So you've got several key concepts. You've got a service, a graph, a function, a model hub and a traffic router. Can you explain the key concepts in Pinecone?

[00:30:26] EL: Sure. So let's start with a function. The function is easiest. I think the function you should think about is one node that does one thing. A function could be either a data node, an index, okay? So something that holds your vectors and searches through them and so on. And a function could be a model, okay? A container holding a model that knows how to execute the model and apply it, okay? These are strung together by your graph definition. So your graph is just the dag that defines the execution of how you connect functions together. So when I get a query, I maybe pass it through like two different functions that create the graph, to create the vector, sorry. And then I distribute it to maybe 20 data shards and each shard is

replicated three times. Each one of those is a function by the way. So your graph just defines that computation graph, okay?

The computation graph is a blueprint. It's the definition. It's a configuration for a service, but it's an object. It's a configuration file. It's an inanimate like object. In some sense, yeah, I mean, it's not live in any way. A service is an actual running database based on a graph definition. So once you've defined your functions, you created the graph that lays them out and then you deploy a graph. And deploying a graph means go and start all those containers and connect them in the right way and make sure that – And let me know when that's done. Once you have that, you have a running service and you have a database. You have your database ready to go.

Once you do that, you can connect to it and you can connect to it from anywhere, from Java, from Python, from your frontend, from your backend. It doesn't matter. Once it's live, it's live. And your connection is kind of like a cursor. It's a channel to upload, delete and so on. Oh! And the router is the one thing I didn't cover, which is sometimes you want to switch between applications maybe for an AB test. Maybe you just trained new models and you're trying to switch to like new vector embeddings. So we allow you to have your service basically behind a switch almost. And so instead of pinging a specific service, you're pinging the router and the router just routes to a specific service. So you can say to the router. Now switch to kind of the new service. And then you just have a zero downtime rollout of a new model or a new service. That's it. So router is kind of like a switch more than – It's a router. That’s what it is. Did I miss anything? Or did I cover all the core components?

[00:33:11] JM: I think that's all of them. Okay. So now we've covered the basics of the terminology. Let's talk a little bit more about usage. So let's say I’m a developer. I’m developing a model in Torch or TensorFlow. How am I using Pinecone? How does it come into my workflow?

[00:33:28] EL: Great. So the question is what do you use the model for, right? So let's assume you're using – You have a model in PyTorch that embeds text documents in this vector or

presentation and you're trying to do a semantic similarity. Let's look at the kind of service, customer service application that I spoke out before. The way you would use it is that first you of course onboard Pinecone. That means you just go and ask for an API key. Like a minute later you'll have it in your inbox. For the first month you'd use it for free while you figure out how to use it. So you have it most – You have 10 nodes used for free for the first month until you get your bearings.

The way you do it is simple. I mean, you'll create a function in your model hub with your model, which we have instructions on how to do. It's quite simple. You create a service with your model as a preprocessor for your data, and that's it. You can spin it up and then ingest however many documents you want into it. And now you have a semantic search engine and you can now ping that service from your app or from your back end or from wherever it is that lives and you have just built a semantic search engine in like a day's worth of work instead of I don't know how long it would take you to do otherwise.

[00:34:49] JM: And presumably the users are generally using Pinecone as a service, right? It's not something that they would deploy themselves on their own infrastructure.

[00:35:00] EL: Correct. It's only a service. You cannot deploy it on yours. I mean, we can. For enterprise customers, we actually bind into their VPC, but the shared tenant environment that you get free access to is – Yeah, it's only a managed service. That's how we can give you completely hands-free management so you don't have to care about machines and scaling and full tolerant and monitor. And all that stuff is done for you and you can have a completely hands-free experience doing that.

And the only thing that you install is really just a very thin client that communicates with the service that you can install really anywhere. That's it. I think it's worth maybe saying that while Pinecone is a managed service, we take both security and privacy extremely seriously. Everything is encrypted. Everything is isolated in terms of communication, network, containers, everything. So we're happy to discuss that and we'll again release more materials very soon.

But the fact that it's a managed service doesn't make it any less secure or private than anything else you'd be using on your own.

[00:36:19] JM: Well, let's begin to wind down. I'd love to get your perspective for things going forward. Well, I guess actually there are a number of different subcategories. So first, given that you worked on SageMaker and AWS for a while, tell me about the future of how machine learning developers are going to be interacting with cloud services. How will that evolve over time?

[00:36:40] EL: I'm a great believer in the future of AI and machine learning. I think that a lot of what we do today either manually or work very hard for will become a lot easier with machine learning. But for that, the entire machine learning ecosystem needs to evolve and needs to provide a lot better tools for pretty much everything that we do. And Pinecone is our contribution in that effort. We took one of the core components, one of the things that we think is hardest to build, and we built it as a service and we give you vector search as a service anywhere you are in the world no matter if you're a one-person company or a five-thousand-person company. And that trend will keep on happening. And I think you'll see components and sub-parts of the process being perfected and serviced and services being built around it such that it's accessible to the machine learning engineer and the infrastructure engineer just as much as it was made accessible to the scientist in the lab before. And so this penetration of machine learning and AI to the everyday kind of just workload of – The everyday kind of life of a machine learning engineer and infrastructure engineer require that to happen, and it is happening. And we see that. I mean, SageMaker was our effort at AWS to do something like that. There are many efforts on by small companies and large alike, and of course Pinecone is not different. We, like I said, took what we think we are best at and we can do the best job at and we are devoting ourselves to building that.

[00:38:19] JM: Any broader predictions about machine learning in general? How is machine learning going to change in the next few years?

[00:38:27] EL: It's a good question. Look. I mean, I'm a scientist by training and I'm gravitating towards the math and the algorithms when I think about the far future. I actually think our understanding of machine learning in general is still quite poor to be honest with the – I don't know how many, probably hundreds of thousands of human years already dedicated to this field. Yeah, maybe in the millions of human years already dedicated to this field. We amazingly know very little. I'm not daunted by this. This is exciting to me. That means we have still so much to learn and so much to discover.

And so I think anything from training, to retrieval, to representation of data, to just like completely new tasks that we thought were completely impossible with machine learning, they'll be possible and probably very soon. In the same time we'll see increased understanding of how to make machine learning model and data systems more fair, more transparent, more explainable, more useful on kind of all the way from a tiny device to like thousands of servers. The horizon is wide and far for machine learning. So almost in any direction that you want to go, there's still a lot to do. So that's I think – We're in an exciting time in our lives.

I think the main change is going to happen is something I already said, which is the main people practicing AI and machine learning are going to move out of the lab and out of the textbooks and into the kind of main core engineering force in companies and are going to just move to production and become everyday tools and everyday application rather than theoretical exercise and blog posts.

[00:40:21] JM: Wool. Well anything else you want to add, Edo? It's been a pleasure talking to you.

[00:40:24] EL: Same here, man. I'm glad I could share my thoughts, and I hope to see everybody listening here contributing to ideas in this field and deploying cool machine learning applications soon. And if Pinecone is good for you, then great. We'd love to hear about it.

[00:40:42] JM: Wonderful. Okay. Well, thanks for coming on.

[00:40:43] EL: Thank you.

[END]