EPISODE 1097

[INTRODUCTION]

[00:00:00] JM: Developing machine learning models is not easy. From the perspective of the machine learning researcher, there is the iterative process of tuning hyperparameters and selecting relevant features. From the perspective of the operation's engineer, there is the hand-off from development to production and the management of GPU clusters to parallelize model training.

In the last 5 years, machine learning has become easier to use, thanks to point solutions. Tensorflow, cloud provider tools, Spark, Jupyter Notebooks. But every company works differently and there are few hard and fast rules for the workflows around machine learning operations.

Determined AI is a platform that provides a means for collaborating around data preparation, model development and training and model deployment. Neil Conway is a cofounder of Determined and he joins the show to discuss the challenges around machine learning operations and what he's built with Determined.

I want to mention that we are looking for writers. If you are interested in writing for Software Engineering Daily, send me an email, jeff@softwareengineeringdaily. And personally, I'm making some investments. If you are a developer who's building a developer tools company or something that's closely related to software engineering, you can also send me an email, jeff@softwareengineeringdaily.com. I'd be curious to see what you're building. And thanks for listening.

[SPONSOR MESSAGE]

[00:01:27] JM: JFrog Container Registry is a comprehensive registry that supports Docker containers and Helm chart repositories for your Kubernetes deployments. It supports not only local storage for your artifacts, but also proxying remote registries and repositories and virtual repositories to simplify configuration.

Use any number of Docker registries over the same backend providing quality gates and promotion between those different environments. Use JFrog Container Registry to search the custom metadata of the repositories. You can find out more about JFrog Container Registry by visiting softwareengineeringdaily.com/jfrog. That's softwareengineering.com/jfrog.

[INTERVIEW]

[00:02:22] JM: Neil, welcome to the show.

[00:02:24] NC: Thanks, Jeff. Great to be here.

[00:02:25] JM: Your company is built around the continuous iteration of machine learning. You have exploration and iteration, the hyperparameter search and model training. Describe this continuous iteration of machine learning in more detail.

[00:02:40] NC: Yeah. That's a great question. I think one thing that differentiates machine learning when you're building an application that involves a machine learning from traditional software development is that when we're building traditional software, we have a definition of what a correctness looks like and we have a specification or something like that, and we're trying to build, we're trying to write down a bunch of rules, software code that meets that specification.

When we're doing machine learning on the other hand, what we're actually doing is often searching through a space of possible solutions, and in some cases having the computer help us explore that space. Rather than writing down all the rules ourselves, we're sort of searching over large bodies of possible rule sets and we might encode those rules using a neural network or some other kind of structure. But as the developer, we're sort of orchestrating this search and kind of supervising it. That's just inherently a more of an experimental process, more of an iterative process where you might start with one dataset or a data represented in a certain way and try to find, given that dataset, what kind of models, what kinds of rules am I able – Does the computer produce and then iterate on that many times and explore different parameters,

different ways to structure the rule sets, different kinds of data. It's inherently kind of I would say a much more iterative process maybe than traditional software intensity.

[00:03:56] JM: Okay. And in practice, the bottleneck that I hear about most often is we've got a data scientist that's working with a notebook, a Jupyter Notebook. And then you've got these machine learning engineers that need to productionize these models. And you need to take this work that this data scientist is doing ad hoc and you need to productionize it. We've discussed this ad nauseam on the show, but this handoff between the data scientist and the machine learning engineer, it seems iconic of general frictions that appear in machine learning workflows. Can you tell me about what makes this handoff difficult? More generally, what are the other frictions around machine learning development that you've seen?

[00:04:42] NC: Yeah**,** that's a great question. So, specifically on the handoff from the kind of model development team or the model development process through to the deployment process. Yeah, I mean, there's a lot of sources of friction there. And I think part of the reason why is that the – First of all, there's a lot of moving pieces. There're maybe many different models. You might be deploying it at different environments. You might be deploying it on the edge or on a web service. Those pieces also tend to be the technology that you're actually deploying. A lot of those APIs, a lot of those interfaces are still moving pieces.

Even something as simple as taking a machine learning model that you've trained, serializing it to a file and then de-serializing it back into something that you can use in memory for either further training or for inference, you would think that that would really be table stakes. And pretty much all frameworks will provide a way to do that. But doing that in a way that preserves all the information of a trained model including some of the custom extensions and custom layers that certain frameworks allow you to do, the state of your optimizers, really making that, even that kind of relatively basic operation, that can still – Doing that correctly is not something that always happens out of the box and can require some additional code on the part of the user.

I think those are at least two factors and that like the APIs relatively are moving pretty quickly. I think another factor is just that it's often the set of people who do the machine learning and the actual operating of the production service and the set of people who develop the actual model are two different groups of people in many cases. That's not a model that organizations use.

And just that handoff, the interface between those two teams I think is something that is just right before potentially for miscommunication or one team might have one set of requirements and another team might have totally different set of requirements. One thing we see, for example, is at model development time, the team developing the model might optimize for maximizing accuracy. Building the best model that kind of solves the problem that you're trying to solve with the highest degree of accuracy. They'll train that model typically on data center GPUs with very fast storage and they'll get to a model that is effective. But then when it comes time to actually deploy that model in production either on an edge device or on a web service, that's a very different environment. You might not have access to any GPUs, or maybe there'll be embedded GPUs, and there might be a latency budget or a power budget or some other kind of constraint that might not even have been incorporated at model development time.

That kind of deployment team then has to think about, "Okay, we have this model that is very accurate, but extremely slow. Maybe very memory-intensive and running it on a different custom hardware than it was initially trained on. We need to apply a set of kind of compression techniques or model simplification techniques to improve its runtime performance at the expense of making it less accurate." And that's an example of where you have two different groups with two different objectives, and that kind of dynamic does not always leads to the best outcome.

[00:07:40] JM: That example of having heterogeneous hardware, and you've got one set of hardware that you train the model on. You've got another set of hardware that you deploy it to. That seems like something that's – You couldn't really solve that with a software platform, right? That's just you've got disparate hardware. What's the solution to fixing the dichotomy between a deployment environment and a testing environment? Training environment I should say. Not testing.

[00:08:10] NC: Right. I think one tool that can be helpful is the ability to understand at training time to basically have performance models that will enable you to predict on a certain class of hardware that I'm going to predict and deploy my model one. What is it going to perform like?

That's not I would say common capability, but that's something that would be a really useful tool to be able to have with just the ability to say, "At training time, when I'm training a model, it's essentially a multi-objective organization problem. Yes, I want to have high-degree of model accuracy, but I also want to ensure that my performance on a certain class of hardware fits within this kind of performance window that I have."

But I think the other kind of piece there is just kind of enabling those different teams that are working together to have that kind of shared vocabulary or a shared kind of API interface to be able to say, "What does the acceptance criteria look like for a model that we're about to deploy?" Because right now, accuracy metric, that's certainly something that is I think everyone's around the same about understanding what the accuracy requirements are. But how you quantify or how you say, "Okay, what are the additional performance requirements?" is something that I think not all teams have figured out.

[00:09:16] JM: Determined is an AI platform, machine learning platform. That can mean a lot of different things. And we're in a time where there are so much good tooling. A lot of these tooling is disparate. A lot of it may require writing scripts to wire things together or do a lot of work to wire things together to build the right workflows. I think of a platform as something that can perhaps smooth-out the frictions between different tools. Tell me about what you're trying to accomplish with Determined. What functionality is it serving?

[00:09:55] NC: Yeah, absolutely. Determined is an open source deep learning training platform. To kind of take a look at those pieces kind of one at a time. We're open source. We're under the Apache license, and we're focused on deep learning. In particular, rather than kind of all of classical machine learning, we're really focused on models that are written in TensorFlow, and PyTorch typically running on GPUs. And we're a training platform, and that our goal really is to enable ML engineers to train better models in less time to manage their training hardware resources, like GPUs and to collaborate more effectively with their colleagues.

We make it really easy to either do your training on-premise or in the cloud to work with wherever your existing training data happens to live. And then once you've trained a model, we make it very easy for you to export that model outside of our system to run in an inference wherever you're doing inference, like a serving platform like Seldon, or on an initial device.

So we do try to scope the set of problems that we're tackling by saying, "We're only going to deep learning and we're really focused on training." But really within that scope, we do try to build a really integrated product where all the pieces that we built work together really naturally, because I think that you're right, that there are a lot of really interesting tools in the machine learning infrastructure space right now, and it's a space that's changing really quickly.

But I think as a data scientist who ultimately wants to do machine learning and train better models, it's incumbent on you to figure out, or you or your team to figure out how do I integrate all those tools together? How do I really kind of put together a package that is going to solve the problems that I actually care about and enable me to do better deep learning? Which can be a really considerable amount of work just given the number different tools in the space and that rate at which that's changing.

[00:11:41] JM: Is it about same defaults?

[00:11:42] NC: I think that's a bunch of it, because things like – One example on our platform is that one of the features that we have is around experiment tracking and metrics management. We've talked to teams who they will give members of their team the ability to record metrics in a database or something like that. But if that is the default behavior and if you don't kind of – Or if you are kind of necessarily prescriptive in terms of how models that are trained, how that data should appear in a metrics database, then it's something that is going to see – Won't see the kind of adaption that you might want. Whereas in our case, every workload that's trained on our system, we automatically capture all the training and validation metrics, the training logs, the hyper parameters. If a model is trained as part of a hyper parameter search, we capture that context. And that's automatically linked together kind of out of the box. That's the default behavior that you get. Rather than saying, "Hey, here's the ability to train a model and here's the ability to record metrics about a model. And if you want to use those together, then good luck." I think they're kind of going that additional extra step and making those pieces work together kind of out of the box. I think that really adds up over time.

[00:12:54] JM: And can you just tell me about the same defaults, or like what the happy path of a workflow for a user who is using Determined AI? What are the set of tools that you're going to be loosely prescriptive around?

[00:13:12] NC: yeah. Maybe just to step back a little bit more and talk about the kind of major functionality or how the pieces fit together. So I would think of Determined, as we said, a training platform. And what we enable you to do is continue to use tools like TensorFlow and PyTorch to write on your model architecture, to write down how the optimization procedure or what your validation metrics are and so on. We allow you to continue using the existing training environment that you're comfortable with. So, GPUs or what have you.

But what we really try to do is enable those tools like TensorFlow and PyTorch, which are really sort of single-user tools. Those tools are kind of focused on a single researcher training a single model with a single GPU or a small number of GPUs, and we try to enable those tools to scale, to scale to larger clusters, larger teams, training many models at once, doing large-scale hyper parameter search and large-scale distributed training.

So some of that kind of key functionality then is saying once you write down your model, and we port your model to our API. So we have an API that is kind of a slight extension of the standard TensorFlow or PyTorch APIs. You do need to go through that adjustment of taking your code and modifying a better API. But once you've done that, you don't need to change your model to go from training on one GPU to training on many GPUs.

If you want to do distributor training, if you want to use many GPUs at once, that's just a configuration setting where you say, “Hey, I want to use 64 GPUs to train this model.” We’ll take care of scheduling a task on all 64 GPUs, orchestrating that training operation, doing data parallel synchronous training, automatically making that fault-tolerant, capturing all the experiment metrics from that workload and doing that in a multiuser way. Where you don't necessarily need to think about, or you definitely don’t need to think about, “Well, how do I install a distributor training package? How do I enable my – How do I install an MPI to make the training package work? What happens if other people are on the cluster and also want to do distributed training?” It’s just sort of capability where once you made that initial investment to

move over to our – To adapt your model to our API, then the number of GPUs you want to use is just a configuration setting.

[00:15:16] JM: Why is that so useful? What's so tricky about just like doing my own work to set up distributed training?

[00:15:24] NC: Yeah. I mean, it's the kind of thing where it's sort of kind of death by a thousand cuts, and that there are great packages for a distributor training, Horovod, for example. We use Horovod internally inside our platform. It's a great piece of software. But as we talked about, there are so many tools in this space and so many different problems that you need to solve as an ML engineer, that taking the time to understand, install, configure, and continue to operate, all the different pieces of software that you need to really put together a full-featured ML platform is just a sizable undertaking by itself. And just doing that work doesn't actually enable you as an ML engineer to train better models.

If you look at something like Horovod, it solves the distributor training problem in the narrow well, but it's doesn't solve, "Well, how do I work with other colleagues in my team who also want to do distributor training?" It doesn't solve fault tolerance when. In order to install, you need to grapple with stuff like MPI or enabling networking to work between the machines you want to do distributor training over.

If that was the only kind of tool that you needed to install, I think it would be manageable, but it's more that that's just one of a bunch of tools, each of which is sort of special purpose, each of which has to be configured and maintained that adds up to a burden especially when you consider that this is not directly machine learning. These are kind of tools to enable machine learning. So at the end of the day, it kind of does end up having a lot of work that's not directly what you're trying to do.

[SPONSOR MESSAGE]

[00:17:02] JM: Today's show is sponsored by Today's show is sponsored by StrongDM. Managing your remote team as they work from home can be difficult. You might be managing a gazillion SSH keys and database passwords and Kubernetes certs. So meet StrongDM.

[INTERVIEW CONTINUED]

[00:18:31] JM: You mentioned Horovod, which is a distributed deep learning framework that was developed Uber, I believe?

[00:18:38] NC: Yup.

[00:18:39] JM: How do you use Horovod?

[00:18:41] NC: Yeah. We use Horovod internally inside our platform to do distributor training. And as I mentioned, it's not something that as a user you need to directly interact with. You just tell us if you want to do distributor training on our system. You just tell us the number of GPUs that you want to use, and we take care of scheduling and orchestrating that workload.

[00:19:00] JM: In terms of how to parallelizable a deep learning job is, I guess one naïvely parallelizable task I could see would be if you've got a bunch of hyper parameters that you want to test in a bunch of different configurations, you could have those different hyper parameter training jobs just parallelized. Is that the main use of a distributed training framework?

[00:19:31] NC: Yes. That's a great point and that there are multiple different ways to exploit parallelism when you're training deep learning models. And there's a question of do you want parallelism within the training of a single model or do you want to be training many models of once? Our system supports both.

So if you want to do hyperparameter search, you are training many different models, and each model uses a different set of hyperparameters. Those models can be trained in parallel. That's a very efficient way to exploit parallel resources, because each of those training operations is completely independent. There's no kind of synchronization happening between them. If your task is hyperparamter search, then it's pretty easy to use parallel resources in an efficient way.

Where a tool like Horovod comes in is where you're training a single model or a small number of models on a much larger set of resources. If I have 32 or 128 GPU and I'm only training one model at a time, by default, that model will only use one GPU to train. Again, if we're doing the hyperparameter search, we want to explore thousands of model conversions. We can train each model with a single GPU. That will be very efficient and we don't need – That's not typically considered distributed training. Whereas if we're training a single model, that model might take a week to train to conversions, and we had many of these GPUs available. If that's the bottleneck on the next step in my machine learning workflow, then the more – If we can throw many GPUs at that to train that much more quickly, then you're able to iterate more quickly.

Typically, training a single model with multiple GPUs is not going to scale quite as efficiently as using multiple GPUs to train multiple models in parallel, just because that training process requires a lot of communication between the GPUs if they're coordinating and training a single model. And that's one of the things that Horovod and other kind of distributed training systems are trying to do you, is kind of be very intelligent with how they schedule that communication and how they overlay communication and computation to try to minimize the amount of communication that's happening and make that scale better. But I think it is inherently a very communication-intensive problem. So it does take some work and certain models will scale better than others in multi-GPU settings.

[00:21:47] JM: Can we talk a little bit more about that?. What would you say, if you had to boil down what parts of machine learning training can be easily parallelized and what parts cannot,

can you just – And I know you just described that in some detail, but maybe you could just give an even more simplified explanation for what parts of machine learning training are parallelizable.

[00:22:17] NC: Sure. So within training a single model or kind of higher level what parts of the kind of machine learning model process?

[00:22:24] JM: I would say, yeah, the higher level. You can just run down the different parts.

[00:22:29] NC: Sure. Yeah. Kind of really high-level, when we're doing machine learning, we typically start with a problem that we want to solve that's cast as a machine learning problem, and a dataset, and typically in some kind of raw format. Typically, we need to preprocess the dataset or do data implementation in order to get into a format that's appropriate to actually train the model on.

Then the ML engineer typically explores a bunch of model architectures, model hyperparameters, choices of which features to use and kind of evaluates which of those choices lead to good model performance. For each of those choices, we're going to train the model either partially or through to different versions.

And then once we have modeled, it works well, we'll kind of deploy that to production or we might do either real-time inference or we may also do batch inference, where we're rescoring a large dataset in an offline way.

If those are the kind of major kind steps in the process, data augmentation typically paralyzes well in most cases depending on exactly the way you're doing data implementation, but there's usually a way to phrase that in a way that is reasonably easy to paralyze.

Hyperparameter search usually does paralyze effectively as well. Because each of the hyperperameter configurations is independent of the other configurations, as long as your search algorithm is friendly to parallelism, and the ones that we typically recommend are able to be easily parallelized, then that's something again that kind of hyperparameter search aspect is something that's easier parallelize.

Then when get down to training a single model, that is harder to paralyze. It's just because it's a very communication-intensive problem. So you're doing a lot of computation, yes, but the results of that computation in terms of the gradient updates for the model itself, those are quite large. They need to be communicated pretty frequently in order to keep the different workers in the computation up-to-date. So that's something that can definitely be parallelized, but the scaling efficiency that you're going to see there differs depending on the model architecture that you're using.

In a common technique that folks will apply in order to make training a single model scale better is they'll use a larger batch size. They'll essentially do more computation before doing that gradient update. That does change the actual training algorithm itself though. That can actually change the behavior of the model that we're training.

In some cases, training with large batch sizes, so doing a lot of local computation before doing that communication phase. Some models that works well. In other cases though, there might be limits on how large of a batch size you can use either due to the kind of GPU memory, because larger batch, they're going to need more GPU memory, or because the model itself doesn't train as effectively with a large batch size. Training itself may or may not parallelize super well depending on the characters of the model.

[00:25:24] JM: As far as the work that needs to be done for resource sharing between different parallel jobs, is that something the you have to write scheduling infrastructure to actually do the resource management yourself, or can you offload all that work to some pre-written scheduler in order to parallelize if a user is doing some training work within Determined? How much scheduler code have you actually had to write?

[00:25:57] NC: Yeah. We've actually built our own job scheduler, and we did that for a couple reasons. First, we felt like building a job scheduler that is specialized for deep learning workloads and for scheduling tasks on top of GPS, we thought that we could just do better scheduling logic.

For example, one of the capabilities that you want if you doing larger hyperparameter search is the ability to explore a large number of hyperparameter configurations on a relatively small number of GPUs If you have 32 GPUs and you want to explore a thousand or 10,000 hyperparmeter configurations, the main hyperparameter search algorithm we use inside the product is based on an algorithm called Hyperband, which one of my cofounders was the inventor of. And the intuition there is that you're training many different models for a relatively short period of time and then looking at those results to decide which models to train further and which configurations are not performing very well.

You might start by training 10,000 models, but each for a relatively short period of time before you decide which to train further. That ability to take a large number of models and multiplex them over a much smaller number of GPUs and have efficiently move them in and out is a pretty important capability for us. And that's not something you're actually going to get on top of an off-the-shelf batch scheduling system or kind of standard container scheduler, like something like Kubernetes. We did make the decision to build our on scheduler partly for that reason.

I think another factor there is, honestly, that talking to a lot of organizations that are doing deep learning today, a lot of people – And we started the company three years ago. So I think even more so back then, kind of integrating GPUs into their container orchestration system and their kind of task scheduling system. For a lot of companies, it was still kind of aspirational where they might've had a monoculture management system, but GPU's were kind of experimental or they hadn't quite figured how they want to manage them yet. They hadn't rolled out Kubernetes for GPUs, for example.

Still, today, we talk to a lot of organizations that might use Kubernetes, but they're only certain to think today about how to use Kubernetes to manage GPUs. That was the other factor, was that I think, certainly three years ago, this was true, and it still should at quite a few places today, is that GPU management is done, a GPU scheduling is done with a very kind of simple techniques. One engineer gets one 8-GPU box. You might statically partition your GPU box. [inaudible 00:28:25]. Or you might have a simple calendar system where users manually reserve GPUs. Those are both kind of techniques that I've seen at relatively sophisticated organizations.

[00:28:36] JM: Rough.

[00:28:37] NC: Yeah.

[00:28:38] JM: And you used to work at Mesosphere. Do you take any lessons from your time at Mesosphere around building scheduler infrastructure? I don't know which worked on at Mesosphere, but –

[00:28:47] NC: Yeah. I worked on kind of core Mesos when I was at my Mesosphere. Kind of more focused on making Mesos work well for databases and stateful workloads than HPC in particular. But certainly, that experience was very relevant to me as we're starting the company and thinking about how we want to do task scheduling.

Yeah. Definitely, it was helpful when it came time to build our own job scheduler. But, honestly, my initial thinking around when we're starting the company making initial kind of technology choices was to say, "Well, there's a ton of momentum around Kubernetes. Let's just be Kubernetes only and let's just assume that everyone's going to have Kubernetes and then we'll kind of build on top of that.

But we spent a while talking to customers trying to understand what the state of play look like around deep learning infrastructure and ML platforms at a lot of companies today, and we really found that the level of Kubernetes penetration for GPU and deep learning workloads three years ago was very small. That kind of meant man being Kubernetes only. We decided not go down that path.

I think that's ticked up slightly over the last couple years, but it's still, I would say, probably something we see in a minority of the folks that we talk to that are doing machine learning is kind of doing that on top of Kubernetes at least at the moment.

[00:30:06] JM: The platform is open sourced. So why is that relevant? When I think about these large kind of platforms for enterprises to adopt that is – Like at least companies like I think Data Robot is one of them, or like Databricks. I think of companies with proprietary, like large proprietary platforms, what's the go-to-market value of having an open sourced platform?

[00:30:36] NC: Yeah. The kind of strategy that we took in terms of getting to the decision to open source the product was we started by wanted to work really closely with a pretty small group of early adapters, and that's how we spent the first two and a half years at the company, is really working closely with companies that were really using deep learning in a serious way and building a product that was a good fit for their use cases and kind of iterating based on feedback from them.

But we also felt like, for a lot of company, two things became true. One, the product became mature enough and we felt like it was ready for broader adaption. We were excited to kind of share with the broader community of teams doing machine learning, deep learning. But the second is just that I think for a lot of teams today when they're looking at what their machine learning infrastructure stack is going to look like, most of those machines are open source.

When you think about kind of developer tools, that's frequently a requirement or certainly very welcome attribute for technology that a company is thinking about really kind of committing to as a core part of their infrastructure stack. Being open source, I think a lot of the folks that we've talk to are really excited about that aspect the product.

[00:31:49] JM: And when you look at the overall space of machine learning infrastructure and you're offering something that is somewhat prescriptive, is there anything that falls outside of that prescriptive workflow that you're offering with Determined that isn't like not so easy to plug into, or are there any frictions with the – That the user base is uncovered in using Determine?

[00:32:16] NC: I wouldn't say there's like fundamental things that people have tried to do that they haven't been able to do. I would say that it's kind of a continual process of getting feedback from customers trying to understand how to make the APIs in a system and the capabilities work more smoothly for the kinds of things people want to do. As the saying goes, everything should be possible, whatever the phrase is. Everything should be possible and easy things should be easier.

I think it's more than a conversation of it is possible to run just an arbitrary container inside the system. That's attached to a GPU where you don't get access to a lot of the product's

functionality, but you do have that capability to sort of do anything that you want. And then there is a more kind of integrated mode where you can run workloads that fit into our APIs where you get a lot of this kind of functionality out of the box and kind of evolving those APIs so that they strike the right balance between expressiveness versus convenience and so on. That's been an evolution.

In particular, one thing recently that we've been working on is kind of extending the APIs beyond sort of kind of simple supervised models to kind of make it easier to train things like GANs, do reinforcement learning and have multiple optimizers, multiple models inside a single [inaudible 00:33:32], which is based on a lot of feedback that we've gotten from customers. That's something that they're excited about.

[SPONSOR MESSAGE]

[00:33:46] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW CONTINUED]

[00:35:22] JM: Let's return to the subject of hyperparameter search. I'm doing a hyperparameter search. That can take a pretty long-time. It can consume a lot of resources. Tell me more about when you have been writing this scheduler. How do you manage the tradeoff between the potential for better performance if you were to just use more and more resources and the cost of that performance? Because obviously, GPU time can add up.

[00:35:54] NC: Yeah. I think that's something where we try to give users the flexibility to make those kinds of tradeoffs at a high level, right? In on-premise setting, you have a fixed pool of resources, and we give you tools to make decisions about how to share those resources over the users of the cluster in a pretty flexible way.

By default, we'll fair share the resources and the cluster over all the active workloads in an equally weighted way. If you have 64 GPU's and you have two hyperparameter searches active, each one by default can use up to 32 GPU's. You can adjust that waiting and you can say this one is more importan. This one is less important.

But the other kind of property there is that we're able to elastically scale those workloads up and down. If one of the searches finishes or if a user cancels one of the searchers, because they are no longer interested in results, that other search can dynamically and automatically be scaled up to use the entire cluster to use all 64 GPUs. Similarly, if a third search then joints the cluster, we're able to dynamically adjust those resources down to each one using approximately 21 GPU's.

I think that's one thing that we give you. In a cloud environment though, because you don't have a static pool of resources, what the product can do for you is kind of automatically provision GPU instances automatically. You can kind of set parameters there in terms of the maximum number of instances you want to provision or what type of instance you want to provision. But that's something that as a user I think you can make different kind of cost versus performance tradeoffs. But I think that is a pretty interesting aspect of the cloud and that if you're doing a large hyperparameter search, we make it very easy for you to spin up a thousand GPUs all at once. Do a very large hyperparameter search completely in parallel, but only use those GPUs for pretty short period of time. That's the kind of thing where just building at that tooling to make

that process of dynamically using all those resources make you do that from scratch. That would be pretty painful. But within the product, it's a pretty straightforward thing to do.

[00:37:58] JM: Let's run through an example. So paint me a picture of a company that wants to use deep learning for some business case. Let's say they want to do image recognition or segmentation and detect objects in an image. They need to deploy deep learning models for the first time in the company. What are the engineers in this company going to do and how will they use Determine potentially?

[00:38:26] NC: Yeah. I would say we often see a pretty similar kind of narrative, and that often, a company, if they're exploring deep learning, is going to start with a proof of concept. And they want to see some evidence that deep learning can be an effective solution to the problems that they're trying to solve. Maybe you kind of use a small amount of cloud resources, or you buy four AGPU box. You kind of dedicate a small team to that, maybe a single engineer or a handful of engineers. And then you try to demonstrate that for the problem that you want to solve. There's some promise there that deep learning is leading you in the direction of an effective solution.

Then for that initial kind of POC project, tools like TensorFlow and PyTorch are reasonably good, are well-suited, and that you're not yet thinking about how do I do deep learning at scale. How do I do it in production? How I do it over a sustained period of time?

But then once a company kind of sees promising results from that initial POC, they start to think about, "Okay. Well, how do we make that next investment to really make deep learning kind of a part of the toolbox and something that we can apply to a variety of problems?" That's I think where Determined really – The sorts of challenges that you face there are largely outside the scope of tools like TensorFlow and PyTorch and are really kind of what are really in the sweet spot for Determined. Questions of how do I share my GPU cluster among a team of users? How do I apply all of my GPU's to training a single model or to do in hyperparameter searches in a very flexible way? How do I track all my model metrics and metadata? If I've deployed a model of production six months ago and I want to revisit, "Well, what version of TensorFlow do they use to train that model? What hyper parameters? What training and validation behavior did I see on which dataset?" We'll automatically capture all that information for you.

Whereas that question of how to manage my models and how do I track my metrics? During the initial POC phase, something like a spreadsheet is a totally adequate solution to that problem, right? But once you go from training a single model with a team of 1 to 2 to training 10s, 20 or 30 models on a team of 5, 10, 15, that's really an area where investing in some process and investing some tooling is going to give you a pretty important thing to do.

[00:40:42] JM: From a business perspective, what have you found to be difficult around the go-to-market process for Determined?

[00:40:50] NC: I mean, I think one thing that's been an evolution is just I think that striking the right balance between talking to – When you're trying to kind of market a very technical product like this to a technical user base in a fairly crowded space. Figuring out the right way to describe what makes a product unique, what the value proposition is, has been something that has been an evolution.

This is an area where different companies are at pretty different stages of their kind of deep adaption process. They are certainly plenty of companies talking about AI. Some companies are really far down that path of really adapting it in practice and exploiting it successfully. Other companies are kind of much earlier.

So being able to adapt the way you talk about the product and the set of problems to the kind of a sophistication of the audience and just kind of being clear about who the intended user is and the benefits they are going to see is something that has taken us a little while to figure out. I think that some people have kind of gone down that path of really trying to scale deep learning and trying to productionize it, and they've seen a bunch of – They've gotten battle scars from some of the issues that arise there. Whereas other people, they're sort of still trying to look around the corner and predict, "Well, once we go and do that, what issues are we going to run into?" I think that's two different instances that you probably want to talk to a little bit differently.

[00:42:17] JM: All right. Well, just to close off, what do you think are the emergent problems on the horizon that are going to be more acute problems for enterprises that are trying to develop

machine learning models, develop a well-rounded process for machine learning development? What's around the corner?

[00:42:36] NC: Yeah. I mean, one area that we're watching pretty carefully is the hardware space. I think compared to classical software, deep learning is already pretty interesting from a hardware perspective. Anyone building standard software, you're probably pretty content building and developing environment around multicore CPU. Deep learning obviously needs to take an energy of GPUs, but it's basically been – NVIDIA GPUs has been kind of the only interesting training environment for the history of deep learning.

There are a lot of companies that are trying to change that and developing kind of custom trips for doing deep learning and training. And those products are starting to come to market now. I think that'll be a pretty interesting event to see what that looks like. And both the price-performance of some of those different offerings, but also what you need to do as a deep learning team to make your training environment kind of hardware-agnostic or able to exploit the best kind of class of hardware for your particular model. Because I think it's also likely to be the case that different kinds of hardware might be more effective at training different kinds of models, and it's not necessarily a one-size-fits-all kind of a situation. Building your ML infrastructure in a way that gives you that kind of hardware portability is something that I think is going to be an interesting challenge for the industry in the future.

[00:43:57] JM: This is like a wider range of chip types, wider range of I guess memory size of machine that you're talking about? Like all the configurations?

[00:44:08] NC: Yeah. There are a bunch of startups building deep learning training accelerators, companies like Cerebras and Graphcore. There's quite a few of them. As well as a bunch of the incumbents and cloud vendors are also building their own kind of custom hardware. Google have TPUs. Then both the training time added inference time as well, kind of custom trips for doing low-power, high-performance, departing inference. I think that's something where being able to kind of manage that hardware diversity and make your training, give you the capability to move between those different hardware environments is going to be an interesting capability if you have that. It's also something that's going to intersect, I think, interestingly with questions around cloud portability, because you can certainly make a choice to say, "Okay, we're going to

double down on going with a certain cloud vendor and a certain kind of limited set of hardware accelerators that are available through that vendor that will have certain advantages in terms of maybe that's not going to be more integrated or what have you. But it does mean kind of maybe kind of committing to working well in that environment, but pretty difficult transition story to working on either different cloud or on a different set of hardware accelerators.

[00:45:15] JM: Cool. All right, Neil. Well, thank you so much for coming on the show. It's been a real pleasure talking to you at Determined, and congratulations on all the success.

[00:45:22] NC: Thanks very much, Jeff. Real pleasure to be on here.

[END OF INTERVIEW]

[00:45:33] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[END]