Determined Podcast Series

Neil Conway on the Challenges of Deep Learning Model Development

Podcast Transcript

Craig: Neil, why don't you give us some of your background, what you did before starting Determined.

Neil: I've been working in the data infrastructure, data management, distributed computing space for almost the last 20 years.

Towards the end of high school, I started to play around with Postgres and joined that development community and spent the next five or six years contributing to Postgres development. Then I got to know a bunch of folks from Berkeley and ended up going to Berkeley to get my PhD, and that's where I got to know my co-founders at Determined. They were really excited about the potential of deep learning. That was just a fundamental difference in the kind of applications that were possible to build within a reasonable amount of time using deep learning.

But three, three and a half years ago, from a tooling and an infrastructure perspective, it was really challenging to build these applications, especially if you happen to be outside of a very small number of elite organizations like Google Brain or Facebook AI Research.

Deep learning is fundamentally different from classical machine learning, both in terms of what it enables you to do, but what some of the challenges are. So with deep learning, you are typically training models on custom accelerators, typically GPUs, and part of the reason you need to use GPUs is just the computational requirements of training these deep learning models is just dramatically, dramatically larger than for classical machine learning. So even with a modern GPU, training a deep learning model might take a week of computation or longer. So the amount of computation, as well as the size of the data sets that people are using for deep learning is really a stark difference.

If you're doing classical ML, you might work with a tool like, R or MATLAB or something like that - you want to find good hyperparameters for your model, so you kick off a tuning operation and you go make some coffee. You come back and, you know, 25 or 30 minutes later, you have a reasonable set of hyperparameters.

With deep learning, if training a single model takes a week, and if your model has five, 10, 15 hyperparameters, then it's a much more complicated, involved process to train one model, let

alone to do architecture, search, or hyperparameter tuning. You really need to make it easy for deep learning engineers to go from training a model in their local development environment, using one GPU to be able to access massive amounts of computation. That's really table stakes to be productive at, doing deep learning today.

Craig: And previously, there wasn't off the shelf tooling, people had to build the connectors between the data and the training model. Is that the problem that Determined solves?

Neil: There is a lot of excitement around tools for deep learning and different software packages to train deep learning models better. A lot of that is centered around PyTorch and TensorFlow, which are really great packages. But I think of those as single user tools. So if you're a single researcher with a single GPU, tools like PyTorch and TensorFlow help you with the basics, right? How do I write down my model architecture? How do I describe how I'm optimizing the model? How do I describe how I'm computing validation metrics? How do I load data into the model? Those kinds of basic questions.

But there's a whole bunch of additional challenges that are essential to being successful with deep learning. Everything from, how do I train my model using a larger amount of computation? How do I track all the metrics that are produced by that training process? How do I tune my hyperparameters? How do I share my computation resources with other people on my team? How does my model get from the training environment to production? Once you go outside the scope of those tools, like TensorFlow and PyTorch, today's tooling often, runs into some challenges for deep learning engineers.

Craig: If an organization is just getting started with deep learning, how should they be thinking about their deep learning infrastructure? What challenges do you see teams encounter most often?

Neil: If you're just getting started with deep learning, then it makes sense to proceed incrementally, but also have a bit of a long-term perspective in terms of picking the right time to make an investment in the infrastructure that is going to be necessary for long-term success. So I think certainly, make sure that your problem is well suited for deep learning. Oftentimes it can be effective to start with cloud resources because you don't need to make that upfront investment. We have a lot of customers who find that they get started in the cloud, using cloud resources for deep learning training, and then as they start to think more seriously about building out a deep learning capability for their organization, they look at the cost of doing cloud-based training and it really can add up pretty quickly. So in some cases it can make a lot of sense to think about building out on premise. But the flip side there is that a lot of those same teams that move towards an on-premise GPU deployment, then struggle to drive good utilization that cluster. One of the things that we try to help customers with at Determined is enabling them to manage a GPU cluster, whether it's in the cloud or on-premise, in a really

flexible way. So that if you have multiple deep learning engineers, they can share that cluster and you can apply all of your GPUs to whatever problem is top of mind for your team.

Craig: How do organizations differ from researchers?

Neil: There's actually quite a lot of overlap between what you need as a researcher today to really be successful doing deep learning and what organizations that are deploying deep learning, what their requirements are.

As a researcher, you're not just training a single model one time with a single GPU. Even if you are operating at a smaller scale, being able to move from training your model on one GPU to accessing massive computation when that's appropriate and being able to do that in a really flexible way, there's still a lot of value there. Even if you're a single researcher. Similarly being able to keep track of all metrics of the jobs that you're training, to organize that information in a structured way is still really valuable, whether you're a small scale or a large scale. And in our experience talking to a lot of machine learning practitioners, if you're an individual researcher, you don't necessarily have the benefit of a platform team or an infrastructure team at your company. And it can be very easy to spend the time that you had budgeted on deep learning essentially managing infrastructure, dealing with the kind of challenges that come with running compute jobs on cloud resources or an on-premise GPU cluster.

And your productivity actually doing the deep learning task that you set out to do can really suffer, because unless you're careful, you can spend a lot of your time on these sort of ancillary kind of infrastructure tasks rather than the actual, machine learning work that you want to do.

So, certainly there are differences in scale between individual researchers and organizations. There are things that are maybe a bit more unique to large organizations like enterprise security or compliance or some kind of regulatory questions. But, there's actually quite a lot of overlap between what you need to be successful as an individual researcher and what a larger team needs in our experience.

Craig: And where does Determined fit in? Ameet talks about the machine learning pipeline or the deep learning pipeline being roughly divided into three phases: the data preparation gathering, and if it's supervised learning, labeling phase, and then the middle is the model building and training, and then the end is deployment. Can you talk about how Determined fits in with that? And can you talk about how it differs from frameworks like TensorFlow or Pytorch?

Neil: Yeah, absolutely. So we're squarely focused on the middle part of that pipeline. You have a dataset, typically a labeled data set for supervised learning, and you're trying to find a model that performs well, solves your deep learning task well, on that dataset. So that's kind of model

training, model development. We're essentially a first rate model development environment, deep learning training platform for that middle part of the pipeline.

And then once you have an effective model, you typically want to deploy that model to some kind of surveying environment, either web service or embedded device. So, our focus is really enabling deep learning engineers to train better models more quickly, and enable them to focus on the actual model development process rather than managing the kind of training infrastructure and other systems concerns that they otherwise would need to spend a lot of time on.

That being said, we do also integrate well with those other parts of the pipeline. So we make it very easy, if you're using tools like Argo or Kubeflow pipelines or airflow to preprocess your data and to build ML pipelines, to connect those tools to your Determined training platform. And similarly, once you've trained a high quality model, we recently added a built-in model registry and one of the nice things that enables you to do is clearly identify which models you've trained inside the system are suitable for production to promote them from the development environment to a production environment. And then we provide easy APIs to export them outside the system.

Ameet: There are these roughly three phases and there's meaty challenges in all three phases. In the model development and training phase, we're tackling a bunch of the very meaty challenges in a nice integrated sort of fashion. But it is of course very important to be aware of problems before and after that phase and to play nicely with the broader ecosystem.

Craig: We talked with Dave Patterson about the connections to hardware and the hardware proliferation. And that's, what you're talking about, Neil, is that right?

Neil: Yeah. Deep learning is a reason why server hardware is suddenly a lot more interesting recently than it has been in the past. It's been quite a challenge to figure out how to integrate some of these custom accelerators into enterprise computing platforms and cluster management systems and make it simple. It's still a challenge for a lot of teams to manage GPUs in the same way that they're kind of comfortable managing CPUs. But we're headed to an environment where computation and hardware is likely to be a lot more heterogeneous in the future than it has been in the past.

Right now NVIDIA GPUs obviously are very, very widely used for deep learning, but our suspicion is that the hardware is likely to get even more diverse. There's a lot of companies building custom chips for aspects of deep learning, either training or inference or both.

Craig: There's a lot of excitement around open source deep learning frameworks, TensorFlow and Pytorch. How does that compare with what Determined does?

Neil: Determined is open source and we also build on top of a bunch of open source technologies, tools like TensorFlow, Pytorch, Docker, Kubernetes, and a bunch of others are part of the platform that we built. What differentiates us is that we've built a batteries-included deep learning development environment, which makes all those tasks that you need to accomplish as part of training, developing deep learning models as simple as we can and automates stuff that you shouldn't have to think about.

Where it makes sense to reuse an open source tool, we're happy to do that. Where we feel like there's a big advantage to building something from scratch internally we've done that as well. So for example, our platform enables deep learning engineers to use distributed training, which is to train one deep learning model using many GPUs. And the underlying kind of distributed training engine we use to do that is an open source project called Horovod, originally from Uber, which is a great piece of software.

But part of the value that we're adding is making it much easier to configure Horovod and to use it in a multi-user setting. With Determined all you need to do is specify the number of GPUs that you want to use for training. And the system takes care of scheduling that job on those GPUs, launching the containers, setting up communication between those containers, configuring Horovod appropriately, running that distributive training job in a fault-tolerant way.

And that all works in a multi-user setting. So if other users are running other training jobs, they kind of play nicely with each other. And you get automatic fault tolerance, experiment tracking, real-time visualization, and so on. I think that if you compare that to the experience you get using Horovod itself or using other kinds of integral open source tools, I think there's a similar consistent pattern which is that in what's available in the open source, the kind of key technical capability might be present. Tools like Horovod are really great at doing distributed training. But if, as a deep learning engineer, if I need to integrate, five, six, seven of those tools in order to get my job done, then just keeping up with the pace of change of those tools, given how fast the ecosystem is moving, just keeping those tools working well together is practically a full-time job. And it's very easy to spend a lot of time configuring the networking I need for Horovod, and the distributed file system I'm storing my data in, and the, job schedule that I'm running my tasks with, and all of a sudden you're just spending a lot of time on these infrastructure questions and system questions, rather than actually developing deep learning models.

Craig: And given that Determined is open source now, there are a number of proprietary model building and training platforms out there. Do you think the market is going to move away from those and move toward open source? I mean, presumably you do because you open sourced it.

Neil: Yeah. I mean, I think that if you look at the deep learning tools that have really seen widespread popularity, almost all of those tools are open source. And I think that's just kind of where development tools are moving these days, it's really kind of open source almost by

default. And we've seen a ton of adoption after open sourcing Determined about six months ago. And, yeah, I think it seems clearly that that's where the market already is or certainly where it's moving pretty rapidly.

Ameet: And it, and it sort of makes sense this, the market is, you know, whether it's an individual researcher or it's somebody working at a, at a big company, the people pushing for these tools.

It's largely kind of a grassroots sort of thing. It's the deep learning engineers who were struggling. They want tools to help them because they know they need this help, and they want to get their hands on whatever is available to them. And so open source is a natural way, you can just get up and running on your own, play around, get help, talk to a community, that sort of thing. It's what these folks are used to, and they're the ones who are driving the building of these communities.

Craig: And can I ask a question? Open source: the community can improve the code. Is there a consensus editing function to keep the code? To sort of make sure that people agree with all edits to the code or does it fork off into different iterations?

Neil: Yeah. So the way that typically works is, you have an open source project, and I would say best practice is if you're running an open source project is to define kind of a procedure for submitting changes to the project. So, often that looks like, talking to the project maintainers and saying, "Hey, I would like to add this capability," or "I noticed this bug," or "What do you think about adding support for this feature?". So I would recommend talking to the maintainers first, to kind of get aligned that this is a change that the maintainers are interested in making. If they're not interested in making it, or if there is that kind of divergence of use, then you do see what's called forking, which is where, a developer basically takes an existing project, makes a copy of it, makes their own changes, and kind of typically renames it and moves in a different direction.

And that can happen when there's disagreement about the direction of a project. But in most cases, contributors kind of discuss the changes they want to make with maintainers. And there's often a contribution or review process where one person proposes a change, one of the other developers or maintainers takes a look at that change makes comments on the change itself, and then there's a back and forth process of editing reviewing that work before it's ultimately accepted into the master copy of the source code. And a lot of this is kind of orchestrated by GitHub these days, in the form of pull requests is a kind of common way of doing that.

Craig: Determined recently announced native support for Kubernetes. Can you tell me a bit more about that?

Neil: Yeah, so it's interesting. You know, there's a lot of excitement about Kubernetes in the container orchestration space. Kubernetes is a container orchestration system from Google. It's inspired by some experience that people at Google internally had building a system called Borg, although Kubernetes itself is not directly related to Borg. And Kubernetes really seeks to make building attributed systems and operating distributed systems simpler -- to provide building blocks for people building distributed systems and people running programs on clusters, to make that more systematic, more straightforward, and as a developer to kind of give you primitives to build upon to build your own distributed systems. So there's a lot of excitement around Kubernetes, and when we started the company about three and a half years ago, my previous experience was working at Mesosphere on Apache Masos, which I would say kind of has similar goals to Kubernetes, although it's a different piece of technology.

But there was a lot of momentum behind Kubernetes and a lot of excitement around using it. So my initial thought was actually that we would probably build Determined around Kubernetes on day one. But we actually found that for deep learning workloads and for organizations that were adopting deep learning, many of them we're not using Kubernetes for doing deep learning. In some cases they might use Kubernetes somewhere inside the organization, but at least three and a half years ago, I would say relatively few people were using Kubernetes for deep learning. I think partly that was because GPU support in Kubernetes at the time was still, alpha or beta quality. And also just, I would say, the newness of both deep learning and Kubernetes, it was just kind of like a relatively early degree of adoption of both those technologies.

And so since then, I would say there's still a good mix of teams using Kubernetes for deep learning versus not. But, we're excited recently that we added Kubernetes support to Determined as an optional capability. So you can run Determined on top of Kubernetes. If the rest of your compute infrastructure is based on Kubernetes, that has a lot of advantages.

It gives you a uniform way to manage your infrastructure. Logging metrics, observability, and so on. All of your deep learning workloads that you're launching by Determined will run as Kubernetes pods, and for a lot of infrastructure and platform teams, that's a really comfortable way to run their infrastructure. That being said, using Determined, a lot of that kind of Kubernetes complexity is abstracted away from you. So as a deep learning engineer, using Determined on top of Kubernetes is no different than using Determined on top of a cloud environment or kind of a bare metal on-premise environment.

You don't need to think about the kind of configuring Kubernetes or the details of the Kubernetes resources that your jobs are running on. We give you a kind of very simple, deep learning-focused configuration file that you're writing, and we really allow you to focus on deep learning rather than on the details of the kind of Kubernetes environment that your workloads happen to run in. And then for teams that haven't gotten started on Kubernetes yet, we continue to support running Determined in a bare metal native environment. There is quite a bit of

complexity in terms of getting started with Kubernetes. So for teams that are either not using Kubernetes or at least not using it yet, that can be a way to get started on deep learning without also having to figure out how to operate a Kubernetes cluster, which can be pretty complicated.

Craig: Aren't you worried, or shouldn't start ups in the broader ML ops space be worried that the big tech companies will eat their lunch? Will decide to focus on building an open source platform and, in the world of journalism the term is, "big foot" the smaller guys?

Neil: Yeah. I mean, I think it's natural and probably wise for startups to be aware of the larger tech companies. Certainly they have a lot of resources, a lot of really smart people working at those companies, and certainly have a good distribution channel in terms of their public cloud infrastructure. That being said, I mean, I think that there's still a real opportunity as a startup in this space to build a really compelling product, for a few reasons. So I think, one: on premise deep learning is something that a lot of teams are looking at seriously. Even if they might start their deep learning efforts in the cloud, right now the economics of doing deep learning training in cloud GPUs just work out to be quite expensive.

So for some teams that's fine, but for many teams there's an opportunity to realize significant cost savings by building out on premise capacity, maybe to complement your cloud GPU capacity. I think that's one opportunity is startups that are building deep learning infrastructure or machine learning structure that can utilize on-premise resources or cloud resources in a flexible way.

Another factor is this kind of hardware heterogeneity that we think is true already of deep learning to an extent, but it's likely to become even more true in the future. So, I think as a deep learning engineer, it's going to be natural that you're going to want access to the ability to run your models on whatever hardware offers the best price performance. And so I think it'll be interesting to see how that evolves and how that interacts with some of these cloud platforms where is it likely that Amazon is going to offer their customers the ability to run workloads on Google's TPU chips. At least today, that's certainly not possible. Good reason to think that won't be possible. So committing to a big cloud vendor might involve cutting off your access to a lot of this more diverse hardware environment that we think is going to become true in the future.

Whereas if you think about building out your deep learning infrastructure on top of a platform that is cloud agnostic, that lets you run on any of the major clouds or on premise, I think that that kind of flexibility will be a really important asset for deep learning teams in the future. And I think the third, if you look at the success of companies like Snowflake, I think there is quite a significant track rate of startup companies that are able to compete effectively with the big cloud vendors.

You know, when snowflake started a few years ago, I think the conventional wisdom was, "Hey, how is this company going to effectively compete against Redshift, given all of Amazon's resources?" And I think we've seen the way that that's turned out - ultimately building a better product and making our customers really happy, that there's still a real opportunity to do that despite the state of play with the big cloud vendors. Certainly I think it would be foolish for any kind of startup in the space to not be concerned about some of the big cloud vendors. But I really still think there's an opportunity to build a great business in the ML infrastructure space.

Craig: Okay, Ameet: did you want to talk more specifically about the pain points that people face in building models and training them?

Ameet: Yeah, sure. So, going back to this very simple straw man of there being three stages: the first stage is data prep. You need to start with data and labels for that data if you want to train these potentially really complex deep learning models. But then the model development and training phase is focused on, okay, let's say I've gone through the process of collecting this data and labeling it. How do I take advantage of massive amounts of data and really expensive hardware and novel modeling techniques in deep learning to spit out a predictive model that's high-performing, very accurate for whatever my application is at hand. Whether it's computer vision, task, an NLP task, or some other task. And there are a bunch of concerns here, right? So if you're an individual researcher, everybody who has these concerns in terms of, "What model do I pick in the first place?" Right? These deep learning models are very complex.

There's no one way to design these models. There's lots of architectural decisions in terms of how you design the model itself. Once you even design the model, there are a bunch of ways to fine tune that design, both of the model itself, but also of the training routines that you use to kind of learn this model.

So problems related to architecture design and hyper parameter search are clearly important problems. Maybe even before thinking about those, there's this problem of, how do I train even a single fixed model with a massive amount of data on my hardware in a reasonable amount of time?

So training a single deep learning model can take days to weeks, even on a single machine. If you want to now tune that model, instead of training that model once, you might be training at tens or hundreds of times. So people often, whether it's individuals or teams, they leverage multiple GPUs rather than one to kind of parallelize the work.

There's then this question of, how do I train my individual model more quickly? And/or how do I do hyperparameter tuning more quickly by leveraging, not one GPU, but a bunch of GPUs. There's then this question of, maybe I'm doing this on the cloud and some of these GPUs are cheaper versus others. If I'm willing to pay for cheaper GPUs, I have less reliability in terms of

how do I use these GPUs. Or if I'm in a shared cluster of some other sort, I might get preempted for other reasons because my coworkers have to also use these GPUs. So there's just this general question of cluster resource management if you're either working alone in a cloud-based environment, or in a team in an on-prem environment.

There's other questions related to visualizing and debugging. So, I train this model. Machine learning, like most other things, it doesn't work the first time just out of the box. So you might train a model, you might've made a mistake, the model might not be working, you might need to train it for longer, you might need to go back and preprocess your data. There's a bunch of ways that this thing can go wrong. Or maybe you're getting good results, but they're not good enough so you need to collect more data. Whatever it is. But how do you visualize and debug your code and the progress of the model, understanding edge cases and so on? If it takes a really long time to train a model and you would like to inspect in real time as things are happening, if you're spitting out a bunch of artifacts, the log files and model weights and so on, how do I get access to all of that intermediate and final results to be able to explore and visualize?

There's also questions related to, you might train a bunch of models today, but you want to revisit the work tomorrow or six months from now, or you might want to share that work with a collaborator. How can you do that in a straightforward, reproducible way? That's incredibly challenged in the context of deep learning. There's lots of code, lots of modeling decisions. Lots of ways you can introduce non-determinism into the process.

There's no shortage of problems that people face doing this model development problem. Tensor Flow does a great job of, or Pytorch, of training a simple model on a single GPU or CPU. But when you want to do things kind of in practice in the real world, there's lots of ways things can go wrong, become inefficient, or just become unreliable. And so the question of how to do these things in a more streamlined process without building a bunch of infrastructure on your own, it is really challenging.

And I guess that the final piece of all this is, as we mentioned I think at some other point maybe, the idea that while model development and model training obviously is a crucial component of actually getting a model out the door that you can use for your application, it's not something you do in isolation, right? You need to play nicely, and easily be able to import your data from the data prep stages. You need to be able to export the models that you've developed so that they can be used for whatever final purpose you want them to be used. So playing with other tools in the ecosystem is also kind of really important.

Craig: And Neil, can you talk about how the Determined platform addresses those? Can you sort of walk us through how a user connects to the data and builds the model and trains the model on the platform?

Neil: Yeah, absolutely. So when choosing Determined, you're still using an underlying application framework like PyTorch or TensorFlow. We support both of those out of the box.

You do kind of refactor your code slightly to fit into some APIs that we provide. But you basically start with your off the shelf Tensor Flow or Pytorch model, and then you make a few adaptations to fit into the APIs that we provide. Then the other thing that you give us if you want to train a model is a simple configuration file, which describes the kind of training tasks that you want to perform.

So if you want to train a single model for, let's say, 10 epochs worth of training data, so you want to go through the training set 10 times, then you basically specify that and then you create a workload. So you use our command line tool to create a new experiment, which is our term for a training workload. And then the system takes care automatically of packaging your model definition up, along with your configuration file, scheduling that using GPUs either an on-premise GPU cluster or, in a cloud environment, we can provision GPU instances for you automatically. Then we'll launch one or more containers to actually train your model, inject your model code into those containers. Your model will be trained on GPUs, typically. And then we provide a bunch of that sort of additional functionality on top of the basic kind of TensorFlow or Pytorch training functionality. So things like automatic fall tolerance, if there's any provision of the GPUs in the first place. If the GPUs that we provision, if that agent crashes or if your booklet crashes we'll automatically restore the state of your training job on another agent, and it can continue training with no interruption.

As your workload trains, we're capturing training invalidation metrics and visualizing those in real time, so that you can understand as your model trains, what it's physical performance looks like. We also make it very easy if you want to do either hyper parameter search or distributed training. So if we start with distributed training, we kind of have that integrated into the product as a very seamless capability. So if you want to use multiple GPUs to train your model, all you need to do is specify the number of GPUs that you want to use. And otherwise that process of submitting a workload with a config file and your model code looks exactly the same. You just say, "I want to use, let's say 48 GPUs." Then the system will take care of, again, either scheduling your workload on 48 GPUs on your on-premise cluster or provisioning 48 GPU from your cloud provider.

Then Determined will take care of launching containers on all those machines, setting up networking so those containers can communicate. Behind the scenes, we'll use Horovod to actually orchestrate synchronous data parallel training, but that's a detail that you don't need to think about as an end user. You don't have to worry about configuring NPI and Nickel and some of the complexity that that normally entails. So if you want to use distributed training, in other words, get access to really significant computation and use that to speed up your model development process. That's a very easy thing to do.

Another kind of built in capability that our users find really helpful is that we built hyper parameter search directly into the product, as a really kind of first-class capability. I think it's interesting that there's been a lot of research into methods for more advanced hyper parameter search algorithms.

And, you know, Ameet's lab at Carnegie Mellon have been some of the leaders in that space. But despite a lot of very clever algorithms for efficiently searching hyper parameter spaces, if you look at surveys that have been done of machine learning practitioners, most of them still use very simple methods for hyper parameter tuning, grid search or random search.

And I think part of the reason why is that a lot of the time these fancier algorithms are implemented as a separate tool, as a tool that in many cases maybe doesn't run on a cluster or it makes it hard to run to access massive computation, is often difficult to use the API, and to run your hyperparameter search is pretty complicated.

And just that hassle of setting up a new tool, figuring how to run it on a cluster, figuring out how it integrates with the rest of your experiment tracking or your model development workflow is just a pain that often means these fancier algorithms see pretty limited adoption.

So in Determined, again, you fall exactly in that same workflow if you want to do a hyperparameter search versus if you want to train a single model. So you identify the hyper parameters that you care about, and that choice on hyper parameters is totally up to the user, but it's really kind of just whatever knobs are hyper variables you think are going to influence the performance of your model. And then you tell us what the ranges of values you want to search over are, and then you configure the search. So you say, "I want to use this search algorithm." We have a highly tuned implementation of hyper band, which is a state-of-the-art search method built into the system, but it's very easy for you to configure that and say, how many resources you want the search to use. But then otherwise, that process of launching a search workload is no different than training a single model. So you just submit a workload to the cluster and Determined takes care of scheduling all of the containers associated with that search on your resources, running them in a fault-tolerant way, visualizing their progress in real time, so that as a deep learning engineer, this is a tool that anytime you want to explore a space of possible alternatives, possible model architectures or data augmentation techniques or drop out percentages or what have you, anytime you have a decision like that, you can very easily kick off one of these search operations to explore that space of alternatives in a systematic and efficient way.