EPISODE 1109

[INTRODUCTION]

[00:00:00] JM: Ray is a general purpose distributed computing framework. At a low level, Ray provides fault tolerant primitives that support applications running across multiple processors. At a higher level, Ray supports scalable reinforcement learning, including the common problem of hyperparameter tuning.

In a previous episode, we explored the primitives of Ray as well as Anyscale, which is the business built around Ray and reinforcement learning. In today's show, Richard Liaw explores some of the libraries and applications that sit on top of Ray. RLlib gives APIs for reinforcement learning, such as policy serving and multi-agent environments. Tune gives developers an easy way to do scalable hyperparameter tuning, which is necessary for exploring different types of deep learning configurations. In a future show, we will explore Tune with Richard in more detail. But today's show is more about Ray and the platform to build on top of Ray.

If you're interested in sponsoring Software Engineering Daily, send me an email, jeff@softwareengineeringdaily.com. Also, if you are raising money for a company and your company is targeted at developers or it's an infrastructure company, I am investing in great companies, and you can also send an email to jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

[00:01:23] JM: 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. Manage and audit access to servers, databases and Kubernetes clusters no matter where your employees are. With StrongDM, you can easily extend your identity provider to manage infrastructure access. Automate onboarding, off-boarding and moving people within roles. These are annoying problems. You can grant temporary access that automatically expires to your on-call teams. Admins get full auditability into anything anyone does. When they connect, what queries they run, what commands are typed? It's full visibility into everything. For SSH, and RDP, and Kubernetes, that means video replays. For databases, it's a single unified query log across all database management systems. StrongDM is used by companies like Hurst, Peloton, Betterment, Greenhouse and SoFi to manage access. It's more control and less hassle. StrongDM allows you to manage and audit remote access to infrastructure. Start your free 14-day trial today at strongdm.com/sedaily. That's strongdm.com/sedaily to start your free 14-day trial.

[INTERVIEW]

[00:02:53] JM: Richard Liaw, welcome to the show.

[00:02:55] RD: Hi, Jeff. Thanks for having me.

[00:02:57] JM: Yes, it's great to have you, and we're talking today about reinforcement learning and hyperparameter tuning. Describe a typical workflow for reinforcement learning and the developer who is working with reinforcement learning.

[00:03:11] RD: So I would say the typical workflow for someone who's doing reinforcement learning is that they'll start off with some prototype algorithm, because oftentimes the details really matter in reinforcement learning. So it's really important that like a prototype works out of the box. And typically the researcher will create a very simple environment for their algorithm to work in. So now you get to the point where this prototype algorithms works on this really simple environment. What you would do then is you'll probably add more compute and add maybe like a more intensive environment that the agent is going to operate in.

So maybe I should actually take a step back and describe this typical reinforcement learning loop. So the main components of a reinforcement learning algorithm are to have a policy, which dictates some set of actions that is going to be taken. You have an environment, which takes in the actions and provides reward and a current state of the environment. And the policy is going to then take this reward and take this state and figure out what's the best way to maximize its reward, right? In that setting, what we're really focused on here is making sure that we slowly make the environment more and more complex so the reinforcement learning agent is able to be crafted to eventually turn on the target, the final environment that we care about.

So now that we have some context about how reinforcement learning works, going back to the workflow of a reinforcement learning researcher, you probably slowly ramp up your environment and make the algorithm more and more complex. Eventually, you'll get to a point where you're bound by the amount of compute that you have. So that's kind of where I've been working on for the last couple of years. We try to provide some tools that allow reinforcement learning researchers to go from this prototype and this small environment that they have to running in a large scale cluster leveraging compute resources to spin up really intensive simulations while learning from these environment states.

[00:05:25] JM: In talking about reinforcement learning, we should explain these terms agent and policy. Could you describe those two terms?

[00:05:33] RD: Yes. I think probably those are kind of synonymous, and perhaps there are people who disagree with this. But the policies, it's a mapping. So it's a function that takes in state and outputs in action. So when you typically describe a policy, you're probably actually be describing in your network, which takes in something like an image which characterizes the state of the environment. And then outputs an action, like the best action to take. Whether it is to go left, or go right, or whatever.

When we say agent, we kind of – I guess, in my mind, when we put the policy into play, when we actually use the policy inside a system or something like that. So in that sense, we're kind of humanizing this neural network and saying, "Hey, look. This thing is actually able to make decisions by itself even though I guess in code it probably be something like the same thing.

[00:06:33] JM: Policies are worth exploring a little bit more. So if I'm an agent in a reinforcement learning environment, I'm going to be faced with all kinds of decisions. And the way that I choose to navigate those decisions are defined by my policy. How does a policy get selected?

[00:06:54] RD: Right. That's actually a great question, and there are multiple answers to this. Obviously, the policy is – Again, as we talked about earlier, it's a neural network. There're then two questions, right? There's the question of how do we choose what type of neural network to create in the first place, or to train in the first place? And then there's the second question of how do we train the neural network so that it performs this optimal policy later on after the training process?

So the answer to the question of how do we select the right policy to begin with or the right neural network function to begin with? And that's typically through some amount of trial and error and maybe something like hyperparameter tuning, where you have some meta loop that is looking for – That tries out different neural network sizes and then selects the best one.

Now, what defines the best? Well, what you will do is you'll take all these different neural networks that you're going to try out and then you're going to run them through this reinforcement learning process. And that process itself also has a bunch of parameters and there's a law of research being done on the different types of ways you can train each particular neural network.

Yeah, there generally is some heuristics about what's the best training process to use? Which ones are the most robust? And then there's also general heuristics about like what type of hyperparameters should I be – How do I select my neural network function in the first place? But I would say there're just the law of trial and error and law of optimization and compute that's required in order to find this ultimate policy that you're going to end up being able to use on your task or environment.

[00:08:41] JM: So every time I do a show on deep learning, I have never programmed anything in deep learning or machine learning. But every time I do one of these shows, I try to explain back in my understanding what the person has just told me.

[00:08:55] RD: Yeah, sure.

[00:08:55] JM: And every time I get a little bit closer. So as I understand it, typical machine learning challenge is you've got a bunch of parameters, like if I'm trying to find the best restaurant to go to on a Saturday night. There are all these different parameters, like the kind of food, and the star rating, and how far it is away from me. So you have this N-dimensions and you've got this N-dimensional space that you're exploring, and then you have all these examples that sit throughout this N-dimensional space. And it's basically a problem that is so –

You can never actually find the optimal solution to it, because it would be too computational-intensive.

So we use these heuristics to just sort of describe how we're going to explore the space of possibilities, and the way in which we are navigating this space is defined by the hyperparameters. And because there are so many ways that we could set the hyperparameters, it's kind of this naively parallelizable problem. And so we parallelize the different hyperparameter examples. Am I understanding that correctly?

[00:10:04] RP: Yeah, I think most of it is there. I would say yes. If you were to take a step back and just talk about this restaurant example, right? In traditional supervised learning, you might be doing something like given characteristics of this restaurant, find like – Predict something like its star rating. And then you have this box, essentially, this box that you can optimize. And the box, not only can you optimize it but you can also configure some different knobs on the box. So that affect the way it's optimized. And the knobs that we're describing here on this box are the hyperparameters. And because there're so many different knobs and because we can like try this box out many different times in parallel, then we kind of like coverage on this idea of a hyperparameter tuning where we just try out a bunch of these boxes with like the knobs set to different settings and then we try to train them on the same data and trying to get it to predict the best and the most accurate way of doing ratings.

[00:11:10] JM: Okay. And so could you give a few examples of hyperparameters?

[00:11:17] RP: Right. In some philosophy, you can kind of characterize almost anything as a hyperparameter. But typical things that people are turning for hyperparameters into include the learning rate, which dictates how big of an update. Basically, how fast your machine learning model will train. Another hyperparameter that people tune is the size of your neural network. You might have a big neural network or small neural network or a wider neural network or something along those lines.

Finally, there're different things that we can put inside the neural network, right? We can put in different like nonlinearities, which affect the thing that the neural network eventually learns. I think those are probably the big fields of parameter tuning that you're going to encounter.

[00:12:07] JM: And let’s just dive a little bit deeper on one of those hyperparameters. A term you hear a lot is the learning rate. Learning rate sounds like something that's great. Don't we just want this thing to always learn as fast as possible? Why would we ever want a smaller learning rate?

[00:12:22] RP: Yeah. So that’s actually a really great question. And sort of the dynamics of learning rate is actually a huge area of research and it affects both the performance of models and also how we leverage our compute to parallelizing and scale this training. So at a very high level, why is learning rate important? Yeah, why is learning rate important? Why can't we just have a very large learning rate that just tells us exactly where we want to go at any point in time, right?

I would say the answer to that is at a very high level, you can imagine your optimization landscape upon which you’re at a certain point, right? And your goal is to minimize your position. And for simplicity’s sake, let’s say the optimization landscape looks like a bowl. So you're standing somewhere in the bowl, like a rice bowl, and you want to get to the bottom. You can imagine, if you take a very large step, or a step that's longer than the width of the ball, then you can step out of the bowl.

However, if you take a step that's too small, you’ll makes some progress, but not enough progress. Another thing to know is that as you get closer and closer to the bottom of the bowl, you can't actually maintain the same size of steps that you take. Say you're taking like small steps, but big enough to make progress, right? Eventually, you’ll get to the point where you're eventually going to just bounce around in the bottom of the bowl, but never get to the very bottom.

In that stage, it's important to start decreasing your step size. Effectively, this translates to neural network dynamics as this learning rate. Eventually, you want to converge to such a small value that allows you to step into the very, very bottom of the bowl. Does that make sense?

[00:14:12] JM: It does. Yes. So you would want the learning rate to be greater as you're further from the bottom of the bowl and you would want it to be smaller at the bottom of bowl, because you don't want to move past the bottom of the bowl. You really want to hit the ideal spot.

[00:14:25] RP: Yeah, that's a great rehash.

[00:14:28] JM: Okay. And just to drive this home a little bit further for people who are not very familiar with deep learning, such as myself. In a deep learning algorithm, so a reinforcement learning run, how do we even know that we're close to the bottom of the bowl? How do we get to the bottom of the bowl? How do we know where the ideal is? How do we strive for that ideal?

[00:14:54] RP: Yeah. So, Jeff, this is a very philosophical question. And often times you might simply just be in the wrong bowl in the first place, and you'll never know. But what we can know is sort of we can try to see how close this objective that we're trying to optimize is reflective of the actual test that we want to learn.

Let me give you an example. So say what you're trying to do is you're trying to classify like images, and you are given a dataset, a bunch of pictures of cats and dogs. Now, when you observe these pictures, you can provide a label to all of them. But the real question that you want your neural network to the answer is if I put this neural network out there, and it's been trained, it should be able to classify a cat and dog whenever, wherever all the time with perfect accuracy.

Now, the problem arises where your dataset itself is not actually representative of the real world that you want to deploy your neural network. And so as a result, there is this question of like there is this research being done on dataset bias and there is also like a lot of work being done for data collection, where the goal is to make sure the dataset is as real and as accurate as it can be. Or else you're just simply going to be optimizing the wrong objective.

[00:16:21] JM: And so I guess there's also different ways of dividing up your training and your test data, which would be the ways in which the landscape is being defined and the way which the landscape is being tested. So there's just – I think what's worth noting is just that there are so many different ways that machine learning can be parallelized. I think that's probably relevant for the conversation we're about to have about Ray and some of the libraries built on top of it.

[00:16:49] RP: Yeah. Yeah. SO I would say parallelization here in the example that we just gave about these cats and dogs would simply just be like if the dataset is large enough, how do we cut it and slice it across all these different processors and machines in a way that like allows us to maximize our compute resources?

I think actually one thing to note here, just following up on your previous comment, is that the problem of parallelization actually doesn't so much arise from needing to split validation and training. The splitting of that is one problem, but it's typically not the main problem that we have when we're trying to do scaling of training.

[00:17:37] JM: Got you. Okay. So, really, the problem is the size of the model or the different iterations of processing through the model when we're trying different learning rates and other different hyperparameters?

[00:17:49] RP: Yes. I would say you can get three different points. So we're talking specifically about scaling machine learning, right? So like given a law of compute, a law of GPU's, what are different ways we can leverage this like large amount of resources? And so the three different ways that I would say are dealing with big models, so doing model parallelism, dealing with big amounts of data. Doing data parallelism, and then also a higher level, which is doing this hyperparameter turning as you alluded to, trying out all these different models in parallel with different knobs set to them.

[SPONSOR MESSAGE]

[00:18:34] JM: If you listen to this show, you are probably a software engineer or a data scientist. If you want to develop skills to build machine learning models, check out Springboard. Springboard is an online education program that gives you hands-on experience with creating and deploying machine learning models into production, and every student who goes through Springboard is paired with a mentor, a machine learning expert who gives that student one-on-one mentorship support over video.

The Springboard program offers a job guarantee in its career tracks, meaning that you do not have to pay until you secure a job in machine learning. If you're curious about transitioning into machine learning, go to softwareengineeringdaily.com/springboard. Listeners can get $500 in scholarship if they use the code AI Springboard. This scholarship is for 20 students who enroll by going to softwareengineeringdaily.com/springboard and enter the code AI springboard. It takes about 10 minutes to apply. It's free and it's awarded on a first-come first-served basis. If you're interested in transitioning into machine learning, go to softwareengineeringdaily.com/springboard.

Anyone who is interested and likes the idea of building and deploying machine learning models, deep learning models, you might like Springboard. Go to softwareengineeringdaily.com/springboard, and thank you to Springboard for being a sponsor.

[INTERVIEW CONTINUED]

[00:20:11] JM: Now, we’re going to get to talking about hyperparameter tuning in practice. But let's talk a little bit about the lower levels of what you have built on top of. So your library, Tune, is I believe built on Ray. Is that correct?

[00:20:29] RP: Yup.

[00:20:30] JM: So Ray is this general-purpose distributed computing framework. We did a show about it. What are the problems that Ray solves and how are those solutions identified or leveraged in your library, Tune?

[00:20:44] RP: Yeah. Ray solves a larger problem of making distributed computing simple, and it provides two particular interfaces; the task interface and an actor interface, which allows us to do both stateless and stateful computations across a large cluster while seemingly programming like you’re on a single process.

So Tune specifically leverages these stateful interfaces, so this this actor interface that Ray provides to scale out the evaluation and tuning of different hyperparameter and neural networks.

And furthermore, leverages this idea of the stateful computation and being able to interact with these stateful processes to implement a law of distributed hyperparameter tuning algorithms that otherwise would need custom-made solutions or custom-made implementations that might not be available for the general public.

[00:21:52] JM: Got you. And the library, Tune, could you give a little bit more description for what problem it solves?

[00:22:02] RP: Right. Generally speaking, the problem that – Tune is a scalable library for parameter tuning and experiment execution. So it basically addresses three following problems. The first one is these optimization algorithms, right? So complex machine learning tests are largely dependent on these hyperparameters, and Tune provides dozens of algorithms with easy-to-use abstractions that allow researchers to simplify that problem. So that's the first thing, optimization algorithms.

The second one is to simplify distributed systems and it leverages Ray for this. The high-level idea is here. Machine learning model developers, people who are training neural networks, they're rarely distributed system engineers. And so leveraging Ray and leveraging sort of these underlying distributed abstractions, Tune is able to provide an even higher level that abstracts these problems, so fault tolerance, resource scheduling, auto scaling, logging, etc., from these small developers that only really want to care about their dataset and their task and don't want to deal with all these distributed system's problems. That the second one, simplifying distributed systems.

The third final one is that it reduces some code craft for machine learning users. Often times, especially when you get to distributed systems, there is a law of things that you need to do in your machine learning code, including checkpointing. You need to visualize your results and you have to log your training outputs.

Tune is built to specifically handle those things, providing a really simple to use obstruction for machine learning law developers to really ultimately reduce their code complexity and increase their developer philosophy. So to rehash the three problems, I would say that Tune really addresses or providing really nice to use optimization algorithms, simplifying the complexity of distributed hyperparameter tuning, and finally, producing the code complexity for these machine learning engineers.

[00:24:08] JM: Okay. And the interface for programming in Tune, could you give a little bit more color there. What am I actually specifying?

[00:24:20] RP: Yeah. At the very simplest level, you can just take your code, your machine learning model training code. You can put it in the function and you throw this function to Tune, and Tune will automatically execute this function across all the available resources on your cluster. Now, where this gets a little bit more interesting is that your model function might itself be distributed or parallelizable. And so you can provide a resource specification to Tune to have it automatically handle the scheduling of this particular parallelizable training function on the correct resources across the cluster. I mean, it's a very simple interface, and the goal again is to reduce the code complexity and the requirement of like imposing and avoid imposing a framework upon the developer.

[00:25:15] JM: Could we explore this through an example? Could you give an example of a reinforcement learning application and describe how that reinforcement learning application might be made more easy to train through Tune?

[00:25:30] RP: Yeah. So how do we put this? Typically, where you might start off when you're entering Tune is you might already have a script that works, right? You don't want to add too much layer of a complexity upon an algorithm that doesn't quite work yet. So let's assume that there is a model developer or someone who's doing reinforcement learning who already has their training script ready that works on a single machine. And now what they want to do is they really want to optimize the parameters or the hyperparameters of this reinforcing learning algorithm. And this is a very typical situation, because reinforcement learning algorithms are just terribly sensitive to hyperparameters.

What they'll do is take this training function that they already have and they'll just pass it into – They'll take this function, it's a Python function and then they can just pass it into Tune. And Tune will serialize this Python function and ship it to different parts of the cluster to execute. And the same code will run in parallel across all these different machines across all these different

GPUs or CPUs and Tune can interact with them and optimize kind of like this larger group of different hyperparameters choosing the best hyperparameter to evaluate and killing off the ones that don't perform really well.

[00:26:57] JM: So how do I specify what performance actually entails? As the reinforcement learning developer, am I just specifying a reward function and I am optimizing for that reward function and then Tune is going to test with different hyperparameters? And based on the reward function, it's going to kill off the branches of the decision tree that have the lower reward output?

[00:27:27] RP: Yeah. That’s pretty good characterization. I would say that's probably most of what's happening in reinforcement learning. There is also other like – So as you know with any system, there is probably a variety of metrics that are outputted to measure progress. For reinforcement learning, the biggest one is this reward, that like this average reward that it's getting throughout the evaluation of its environment. And this reward is then passed back through a callback to Tune, and then Tune will then compare all these different rewards and decide which ones to start or which ones to kill, whether to try a new training run with different hyperparameters, etc.

I have to say though, one of the most common things in reinforcement learning is not actually the hyperparameter tuning, but rather taking advantage of the simplification of the distributed system that Tune provides. The reason is because, oftentimes, the reinforcement learning algorithms are so unstable and so parameter-sensitive that what you’re really trying to do is just try to figure out whether or not the parameter is sensitive in the first place.

And so a lot of our users who are reinforcement learning users, the end up just doing a simple grid search across a particular parameter and they want to plot what is the sensitivity of this particular parameter across the grid.

[00:28:57] JM: In that case, they're not even trying to find some sort of like classification function. They're just trying to see the impact of some particular parameter.

[00:29:10] RP: Yes. I mean, I just want to put a disclaimer out there. That's because I work with these reinforcement learning researchers, and they might not be the ones who are tasked with optimizing an actual business level objective function, right? They're called researchers to provide some work in new research that other people can build on. So, yeah, they end up trying to just evaluate what the goodness of a particular – And just understand the quality of a particular hyperparameter.

[00:29:40] JM: Could we revisit the subject of hyperparameters? Define the term hyperparameter sweep.

[00:29:47] RP: Yes. So this word hyperparameter sweep comes in different flavors too. Some people will call it an experiment. Some people will call it like a tuning run, etc. And they're all kind of going out the same thing, which is that at the end of the day, there's a bunch of different knobs, and you got to turn each knob in different ways so that you can find the best configuration of knobs. And what that means in practice when you're actually leveraging your compute, it means that you're going to evaluate a bunch of different of these hyperparameter configurations, all these knobs that's set to positions and you're going to execute them all at once, and then you have this aggregate result across all these different runs that you're going to sort of gain some information from, or again the best, trying to find the best particular configuration from. And so this sort of aggregate is what we call a sweep.

[00:30:51] JM: And can you tell me more about how Tune would be used to execute a hyperparameter sweep?

[00:30:59] RP: Right. Specifically, what happens here is, again, we're going back to this mental model of the box being a particular training function, right? So the box, again, has like, say, it has three knobs. And this box is – In Python, it's just a simple function. And what we'll do then is you'll pass this into to Tune and you can specify a number of evaluations you want to have for this particular function and you can also specify a space of different hyperparameters to sample from.

So how do I describe this? I would say there's – Typically what you would do is you would specify something like, "Oh! For this particular knob, that's uniformly sample from 0 to 1." And because we have three knobs, we'll provide three different of these distributions to sample from. So uniform sample 0 to 1 for knob 1, knob 2, knob 3.

And after this like high-level specification is given to Tune, Tune will then leverage Ray to execute these functions and sample from the given uniform distributions to get some three values for the three knobs and pass these values to this training function that is executing in Ray and then retrieve the results after the training function is done. Since Ray provides a really simple abstraction to paralyze and distributor code, this ends up working seamlessly across a large number of processors or even a large number of machines on the cloud.

[00:32:40] JM: Can you say more about how those different execution runs are interacting with each other? Is data being shared across these different executions across different processors?

[00:32:55] RP: Yes. I guess there're two answers to this question. So one is like how much state is being shared across the different runs. And then also, how do the different runs affect each other? To answer the first question, oftentimes, what we'll do in larger datasets is we all either – We'll download the data set onto a node. So imagine, you had a cluster of, say, five nodes where Tune is running over all of them. And on each node, you can run something like 64 different hyperparameter configurations at once. So 64 times 5 gives you 220 different hyperparameter configurations you can evaluate in parallel.

But you actually have five nodes. So what you can do is you can just download the data on to each node and have everyone leverage that same data at the same time. So all 64 processors or training evaluations on that node will read from the same data source. And if your data is small enough to fit in-memory, then you can also leverage, raise shared memory abstractions so that you basically can speed up your read time when you're trying to query this data to be trained, because the shared memory is available for all 64 processors to access. So that's the state aspect. And you can say the same thing for like a large model that you might be downloading or a similar other state that can be shared across the processors and the evaluations.

So the second question just to go back to a discussion of how do different evaluations affect each other? The second question is how does optimization change given more and more evaluations done in parallel? Or how do the evaluations affect each other in terms of the optimization process?

So one very standard technique for doing this is something like success of having where it's a comparison-based hyperparameter tuning algorithm. The high-level idea is that we will compare across all the different evaluations at the same time and we will choose the best view to train for a larger amount of time and we'll terminate the ones that are not in this select few. Basically, the underperforming population of evaluations. And then what we'll do with the select few is we'll either allocate more resources to it. Maybe allocate more GPUs to it or we'll just train it for a longer amount of time and release the rest of the resources. I guess to sum up here, it's really a competitive sort of optimization algorithm that the performance is affected by like each individual evaluation, the performance of those things.

[00:35:53] JM: Got you. And how much of this parallelization is taken care of by the Tune interface? Is there any kind of specification I would need to give to Tune to describe what kinds of data sharing I would want or what kinds of parallelism I would want?

[00:36:11] RP: Yeah. So the data sharing aspect, it requires a little bit of handholding. And typically what you would do is, in your training function, you could just specify like a location, maybe somewhere on the hard drive where you're just going to read data from. And the idea is that whoever reads that data first could – Actually, just to backup. Oftentimes, the cluster doesn't have the data yet.

So typical training functions will also include the downloading of data, especially when it's like a standard dataset. Now, then the typical programming model for this is that if your training function is evaluated on a node that doesn't have a dataset, it will download that data first. What we'll do is we'll provide some utilities to avoid multiple writers writing to that same location. But what ends up happening is after the first download occurs and everyone can read from the same data source. So there's a little bit of manual configuration involved. Sorry. Was there a second question to this?

[00:37:22] JM: No. I was just asking for how much specification there was required by the developer? But I think you gave a good enough description for that.

One other interface I'd like to explore is RLlib. RLlib is a reinforcement learning library built on top of Ray. Is there a relationship between RLlib and Tune?

[00:37:43] RP: Yeah. I can give a quick story, actually. As you know, Ray has taken the academic position of being a reinforcement learning, like a reinforcement learning library or providing a system that tackles these emerging AI applications, specifically highlighting reinforcement learning as one of these emerging AI applications.

Now, I think the story begins at the start of my grad school time, where the first thing I was doing was doing some reinforcement learning research. The first task was just to look at how we can validate Ray as a tool for reinforcing learning research. So I was in charge of the like basically breaking Ray. Implementing these algorithms and telling Robert, or Philip, or Ion, who are other people who work on the Ray team that like these abstractions just don't work for this particular algorithm that I'm supposed to create.

My friend Eric, Eric Liang, who is now the maintainer of RLlib, and I, we would know implement these RL algorithms. And the typical experience would go something like, "Oh. We find this new RL algorithm that we want to implement," and we look at the code, the pseudo code that's in the paper. It's 10 lines of pseudo code. How hard could this be? Three weeks later, we still don't have the reproduced results, because we've been trying to get the hyperparameters to perform correctly. And it just seems like everything is super unstable. The RL algorithm doesn't work across different environments, etc.

And at some point, we had a realization where, "Hey, look. This is not sustainable. We're going to spend all of grad school doing this. So let's just automate this tuning process." So we built Tune. The whole goal of Tune is so that it could just run a bunch of different hyperparameters at once, and diversion the cloud for that.

Eventually, it grew out to become its own standalone tool for all these scaling hyperparameter tuning and expanded beyond this reinforcement learning use case.

[SPONSOR MESSAGE]

[00:40:01] JM: Your code is going to have errors, even code written by an amazing developer. And when bad things happen, it's nice to know that Honeybadger has your back. Honeybadger combines error monitoring, uptime monitoring and cron monitoring into a single easy to use monitoring platform for less cost than you're probably paying right now. Honeybadger monitors and sends error alerts in real-time with all the context needed to see what's causing the error and where it's hiding in your code so that you can quickly fix it and get on with your day.

The included uptime monitoring and cron monitoring also lets you know when your external services are having issues or your background jobs are going AWOL or silently failing. Honeybadger.io is 100% bootstrapped. It's a monitoring solution that's bootstrapped, and this is important, because a self-funded business means that they're priced simply. They operate the business simply and they answer to you, not investors.

Software Engineering Daily listeners can get 30% off for 6 months of Honeybadger, and you can simply mention Software Engineering Daily when signing up. They'll apply the discount to your account and you don't need a credit card. Thanks for listening, and thanks to Honeybadger for being a sponsor.

[INTERVIEW CONTINUED]

[00:41:22] JM: Can you tell me more about the difference between the academic environment and the business application environment? You've now basically straddled those two areas after spending time in the RISE Lab as well as the company that you now work at, Anyscale. So tell me about those two environments.

[00:41:46] RP: Oh, yeah. That's a pretty fun question. I think it's a very unique. Maybe there're two or three other labs in the US that have this sort of flavor. But I think, in Berkeley, there is a strong emphasis from the advisors that what we build – Especially for systems, what we build has to be useful. I think actually this might be particularly an Ion Stoica mentality. But we really care about adaption, and we really care about the system actually being leveraged by researchers, developers, industry, etc.

So, a lot of my grad school was actually spent closing GitHub issues. Obviously, you would do some papers on the side. You'd write some research about some interesting problem that came up while developing the tool. But there is a lot of effort being put into kind of growing this open source community.

I would actually also add that the research that we did, the research that we published was really kind of just – It fell out of this open source work. So one of the works that I worked on, which was this like resource allocation when doing hyperparameter tuning was kind of – The high-level idea is how do we parallelize given a K number of processors or a K number of GPUs. How do we best leverage these resources in a deadline-constrained hyperparameter tuning setting?

And this was very much motivated by user requests asking like how do I know how much to parallelize, right? And what happens if I have a time constraint? How do I maximize my accuracy there? And so we did some research on that. And there is a lot of other research projects that kind of just came out of the practical needs from building the system and making it usable in real-life. So that's kind of the academic world.

Now, moving into industry, I think now that we have this company, Anyscale, which is founded to commercialize Ray and also provide a large compute platform for developers to use and scale their or applications. I would say a lot of the flavor from this academic setting naturally falls over into the Anyscale setting. We still continue to do GitHub issues. We still continue to find interesting problems to work on. I would say there's this much stronger focus now on users and honing down that specific subset of customers that we want to target and grow and help define the future of Ray in a more productionized environment.

[00:44:31] JM: So, what's interesting about Anyscale is it does seem to be a company that's positioned for where the puck is headed. Because I think that enterprises have enough trouble just getting supervised learning algorithms to work productively. And Anyscale is well-positioned to productionize reinforcement learning. In what sense do you feel it's too early? Where the rough edges around reinforcement learning? What are the bottlenecks around reinforcement learning that make it tough to productionize these days? And where are we going with enterprise reinforcement learning?

[00:45:11] RP: Oh! That's super interesting, Jeff. I would say probably – I know a couple of successfull use cases of enterprise reinforcement learning. And you probably use it too. From what I've heard, Facebook has made great progress leveraging reinforcement learning to maximize notification engagement. And because there is a bunch of social networks in Silicon Valley, I think a lot of other companies that have these other social networks are also trying the same thing. Trying to use this enterprise level reinforcing learning, specifically like trying to increase engagement on these social network platforms. So that's one big area, and slowly, that might trickle down to some other applications.

The other big use case that we've seen for reinforcing learning, and I'm not too familiar with this, but I think supply chain optimization seems to have made some nice progress by leveraging reinforcement learning just given our user calls and given the blog posts that other third parties have written about using reinforcement learning tools that Ray has.

So supply chain optimization, notifications, and finally I think J.P. Morgan has found success in a very particular niche of their trading market to somehow use reinforcement learning to place great – What's it called? It's like some sort of trading strategy or some – I don't know the particular details, but they found great success using it. They've gone into public saying like, "Hey, we're using reinforcing learning with Ray to maximize trade executions, or to maximize profit on trade executions. And we've shown some great results." So those are kind of the three cases that we've seen. The main problem moving forward is to find these use cases where reinforcement learning is just going to be 10X the other existing solutions.

[00:47:12] JM: I mean, you got to love – Those examples you gave, they are so well-defined as reinforcement learning problems. It's like notification engagement, "Okay. I sent a bunch of notifications under different reinforcement learning suggestions, and it's very easy to note which ones of these resulted in more engagement. Okay. That's a very easy reward function." "J.P. Morgan trading. Okay. Do they make more money or less money?" Very well-defined reward transaction. Supply chain optimization. Does the package arrived faster? Very well-defined reward function. It almost seems like if you compare it to a classification algorithm, classification algorithm doesn't necessarily have such a great built-in feedback loop as to whether you're successfully classifying anything.

[00:48:02] RP: Yeah. Yeah. I mean, yeah, this idea of having a proper reward function is pretty important. And you can actually look at like any system and say, “Hey, look. We should just optimize this system,” and we know what the metric is, right? We know it has to go faster. And so then we can grow a reinforcement learning habit.

I think perhaps one of the bigger bottlenecks is just the availability of data, and maybe that's wrong. But my hunch is that a lot of times the feedback that we’re going to get from a particular interaction is just going to be so delayed. And so it's going to be hard to use reinforcement learning to sort of navigate that signal versus node tradeoff to find what actually is the reward and how does that propagate through the different actions that we took. Yeah, so that's one thing.

Now, on the other hand, classification, it’s good for what it does, right? It's good for these well-defined tasks of like, “Oh! Hey, look. This spreadsheet, I have a bunch of things and want to fill out this label, or I want to fill out this particular column. So let’s just – We have all these data. We throw at it, and it’s very well-defined. Okay. And then we can measure the accuracy and let’s just move on with our lives.”

So, yeah. I would say maybe one of the hardest part is about reinforcement learning is finding a particular problem where kind of all of like the requirements faulty other of having a lot of fast feedback, having great rewards signal and having a lot of different things you can try at once.

[00:49:36] JM: All right. Well, let's close off by just talking a little bit about the framework landscape. Like, “Okay. We had Spark. We got TensorFlow and got this new thing called Ray, which is its own ecosystem.” And I talked to Ion a little bit about this when he came on the show several months ago. But what are we doing? Why do we need another one of these ecosystems? What is it about Ray that makes something new? What is new about it and why did we need yet another framework?

[00:50:11] RP: Right. So that's a good question. I think one of the biggest things is closing this gap between development and deployment. So deployment can obviously mean multiple things. In the context of hyperparameter tuning, it's simply writing this distributed system in the first place. So, because Ray provides a really simple deployment strategy, it provides auto scaling, it provides the ability to leverage these distributed computational resources in a really simple way. That barrier to entry for our users trying to do distributed hyperparameter tuning is just significantly lower now.

And you can say the same thing for reinforcement learning, right? It’s not easy to program a really efficient, scalable reinforcement learning algorithm, let alone like 30. And because Ray provides this really simple abstraction for dealing with data, fast data movement and parallelizing and scaling across clusters, that become simple. And now reinforcement learning researchers can benefit.

The broader Ray vision is to sort of democratize distributed computing and make it really easy for developers to write their distributed applications, scale them, not worry about the complexities that might come with running – Just running distributed systems in the first place. And this is kind of a niche that isn't well-supported in the current landscape, right? Like you mentioned, Spark. Spark is great for BSP. It’s not great for async execution. It’s not great for these fine-grained tasks that you can do with Ray.

You mentioned TensorFlow. TensorFlow is a model development framework, but it's not itself a distributed execution framework. Typically, in order to scale up your TensorFlow application, you have to implement some other distributed layer. Yeah. I would say Ray kind of sits in this place where machine learning researchers want to go distributed, but just kind of have to face this gap, and Ray is kind of trying to sit in that gap right now.

[00:52:30] JM: Richard, I want to thank you for coming on the show. It's been such a pleasure talking to you.

[00:52:32] RP: Thanks, Jeff. Thanks for having me.

[END OF INTERVIEW]

[00:52:44] JM: Errors, and bugs, and crashes happen all the time across my software. Most often, these crashes have to do with obscure exceptions that come from React components, failing to render on the client device. Source maps and stack traces would be useful, but in many cases, I'm not able to identify the root cause because the error is occurring on a client device. It's not on my infrastructure. And what can I do about that? I can use Sentry. Sentry.io can quickly triage and resolve issues in real-time.

Visit sentry.io/signup and use code SEDAILY to sign up for two free months. You can simply choose the platform that you're integrating with. Sentry works for every major language and framework, from Rails, to C#, to Java, or React Native, and you just install the SDKs. You integrate it with your application. After that, errors will be caught by Sentry and you'll be alerted the moment that they occur so that you can triage, and resolve, and keep your users happy with functional applications. If you're building an application and you want to monitor your errors and your performance, try Sentry by visiting sentry.io/signup. And new users can use code SEDAILY for two free months.

Thank you for being a sponsor of Software Engineering Daily, Sentry, and you can go to sentry.io/signup and use code SEDAILY if you're curious about it.

[END]