EPISODE 76

[INTRODUCTION]

[0:00:10.5] SC: Hello and welcome to another episode of TWiML Talk, the podcast where interview interesting people doing interesting things in machine learning and artificial intelligence. I'm your host, Sam Charrington.

A bit about the show you're about to hear. This show is part of a series that I'm really excited about in part because I've been working in bringing in to you for quite a while now. The focus of this series is a sampling of the really interesting work being done over at OpenAI, the independent AI research lab founded by Elon Musk, Sam Altman and others.

A few quick announcements before we dive into the show. In a few weeks we'll be holding our last TWiML online meet up of the year. On Wednesday, December, please join us and bring your thoughts on the top machine learning an AI stories of 2017 for our discussion segment. For our main presentation, former TWiML Talk guest, Bruno Goncalves, will be discussing the paper; Understanding Deep Learning Requires Rethinking Generalization by Chiyan Zhung from MIT and Google Brain and others. You can find more details and register at twimlai.com/meetup.

Also, we need to build out our 2018 presentation schedule for the meet up. So if you'd like to present your own work or your favorite third-party paper, please reach out to us via email at team@twimlai.com or ping us on social media and let us know.

If you received my newsletter, you already know this, but TWiML is growing and we're looking for an energetic and passionate community manager to managing grow programs, like the podcast and meet up and some other exciting things we've got in store for 2018. This is a full-time role that can be done remotely. If you're interested in learning more, reach out to me for additional details.

I should mention that if you don't already get my newsletter, you are really missing out and should visit twimlai.com/newsletter to sign up.

In this show I'm joined by Jonas Schneider, robotics technical team lead at OpenAI. While in San Francisco a few months ago I sat down with Jonas at the OpenAI office during which we covered a lot of really interesting ground including not just OpenAI's work in robotics, but also OpenAI Gym, which was the first project he worked on at OpenAI as well as how they approach setting up the infrastructure for their experimental work including how they've set up robots as a service environment for their researchers and how they use the open-source Kubernetes project to manage their compute environment. Check it out and let us know what you think.

I really enjoy this one. A quick note before we jump in, support for this OpenAI is brought to you by our friends at NVIDIA, a company which is also a supporter of OpenAI itself. If you're listening to this podcast, you already know about NVIDIA and all the great things they're doing to support advancement in AI research and practice. What you may not know is that the company has a significant presence at the NIPS Conference going on next week in Long Beach, California including four accepted papers.

To learn more about the NVIDIA presence at NIPS, head on over to twimlai.com/nvidia and be sure to visit them at the conference. Of course, I'll be at NIPS as well and I'd love to meet you if you'll be there, so please reach out if you will.

Now, on to the show.

[INTERVIEW]

[0:03:45.1] SC: All right. Hey, everyone. I am here at the OpenAI offices and I am with Jonas Schneider. Jonas is a member of technical staff here as well as being a technical team lead for robotics. Welcome to the podcast, Jonas.

[0:03:58.7] JS: Hey, glad to be here.

[0:03:59.7] SC: It's great to have you on the show. Why don't we get started by having you tell us a little bit about your background and how you got interested in artificial intelligence?

[0:04:09.0] JS: Yeah. My background is actually — I work on OpenAI's robotics team, but my background is actually in software engineering. My background is neither in robotics, like classical academic robotics nor machine learning, which is of course OpenAI's big focus.

In the robotics team, we kind of have a couple of different skillsets coming together. There's roboticists, there's of course machine learning experts and also software engineering people like me.

Before OpenAI I was actually a software engineering intern at Stripe. That's also where I met our CTO, Greg Brockman, who's our CTO at OpenAI. That was actually while I was still in college. After I was done with college, I actually reached out to Greg and was like, "So, what's up these days? What kind of stuff are you working on?" He introduced me to OpenAI.

[0:04:54.9] SC: Fantastic.

[0:04:55.9] JS: Yeah. I actually got started working on OpenAI Gym. I was a contractor for OpenAI there, basically working with Greg. That's how I kind of got started thinking about all of these AI ML things, and then eventually I got started working on OpenAI's robotics team.

[0:05:10.9] SC: Awesome. Tell me a little bit about the work that you do on the robotics team.

[0:05:15.6] JS: Yeah. The goal of the robotics team basically is to enable new capabilities for robots. Today's robots, they're really good at very specific tasks. For example, in a factory robot, you have an assembly line and you have a robot that can place one very specific screw into one specific location on the part that you're assembling, for example. Robots are great at that. They're really good at that. They have very, very high precision. They can repeat this movements tens of thousands of times and they never get tired or anything.

The issue is that these environments are very constrained, of course, because that's how a factory is. Everything is super precise. Whenever something unexpected happens, you just abort basically and the human goes in there and figures it out and then resets and then it goes on again. That's very different from, for example, like a home environment where like if you're trying to clean up someone's apartment, for example, where unexpected stuff happens all the

time and like something falls over and then you have to react to that and pick it back up, for example. This is like a very simple task, but it turns out that like today's robots are actually not that great at operating these unconstrained environments. Usually the reason for that is that they are preprogrammed basically. When you have this factory setting, like when you set up the robot, someone goes in there and basically programs this exact movement pattern for the robot.

Of course, and that works great in this constrained environment, but in the unconstrained environment, you actually can't do that, because you have to react to what's happening in the environment. That's why today's robots are like pretty restricted in these kinds of environments and we're trying to and working on solving that.

[0:06:55.8] SC: Nice. It sounds like the — We're talking about the environments as being constrained and unconstrained, but in a lot of cases is my sense at the constrains themselves are artificially imposed. Like the factory environment is constrained because it has to be because that's the way they can the robots to work. Do you look at applying this stuff in the industrial settings as well?

[0:07:20.9] JS: The thing is — Yeah, you're of course exactly right. Today's automation, basically everything — Like the way these factories are designed is exactly so that the robots can function. I think for the most part, it will be — Or at least with the current state of like capabilities of L-based systems, it will actually be pretty hard to be better than this constrained environment if basically you're application or your business is like fine with operating in the constrained environment.

For example, like while it might be nice to — Let's say, there's a Tesla robot that moves the auto bodies around and then let's say — Like of course it would be nice, for example, if it would be smart enough to react to it if it like drops the thing on the floor, and if could pick it back up and just continue going on. These events are so rare that — Of course, there's a lot of other engineering factors there where like you got to be sure that if the robots in this like reaction mode that it doesn't accidentally cause like more failure somewhere else in the system. Basically just because these events are so rare if your environment is also constrained. We're not really looking at that, I guess, definitely in the short and medium term.

[0:08:28.5] SC: I'm thinking about applications like pick and place types of applications where I've seen a number of research that's kind of in the context of, today, a robot that's being used to bolt stuff in. It's getting these bolts that are like preloaded into very constrained — Like a harness that basically it can bolt in. Some of the examples show that in some instances you might want to just put a box of bolts and let the robot kind of grasp the bolts, and then from there bolt them in. I guess the question is have you looked at the — Is that really practical? Is that where you see the application of this kind of stuff or like, "No, the industrial environments, we've got that kind of figured out. People probably are going to switch from using the very rigid environments to the more unstructured environments to save a little bit of that upfront cost.

[0:09:25.9] JS: Yeah. I think there are definitely some opportunities specifically in the areas I mentioned, like machine tending, basically, where like you have this inbound, like box of parts and they come from like some vendor and you basically have to feed them machines correctly and like unpack them. That is again because it's kind of an unconstrained environment. You don't really know, like they might change like their shipping material and suddenly it looks different and you have to process this package differently.

Also one other kind of interesting example is actually plugging in cables. That's something that these industrial robots are also very bad at, because cables are deformable. Basically, there again you might have to react to like if the cable is bending weirdly, you have to like poke it so it falls into the correct place.

This is actually something that consumes a lot of time in, for example, like auto assembly, where basically you can do the entire frame, so all models, super nice. Then you have a bunch of people like installing all the stuff in the interior for example. That includes actually — Basically, like one very specific task that people have tried to automate using the classical robotics, plugging in these like connectors basically. I guess maybe like a day-to-day. This would be like plugging in a charger into — Like plugging your phone in into a phone charger.

It turns out basically for these problems you actually — These are surprisingly hard, because you have to figure out the exact positioning for the charger and then plug it in correctly.

[0:10:45.2] SC: USBC and like lightning, the reversible ones and make it easy for the robots to charge our phones for us.

[0:10:52.4] JS: Yeah. I think definitely for the industrial applications, like it might be hard to basically, in the short term, improve on the stuff that is already handled by robots. There are definitely a lot of fringe cases that like there's just no way that you could currently even like get close to automating it and new technologies [inaudible 0:11:09.5].

[0:11:10.3] SC: Okay. Tell me a little bit about some other areas of research that you're pursuing to kind of enable this vision.

[0:11:18.1] JS: Yeah. Basically, the thing we're very interested in is basically acting and reacting on feedback from the real world. Basically, this kind of feedback is really also what gives humans this super good manipulation, like super good robotics skills basically. Actually one quick experiment that we could verify that is if you try to — If you put both of your index fingers out horizontally —

[0:11:42.2] SC: I'm doing this now.

[0:11:43.3] JS: Yeah, and you try to move them together and like touch the finger tips. That's very easy to do, and you can always do it. Now, try doing it again with your eyes closed, and it turns this is actually really hard. Yeah, after a while you can — If you have like the touch feedback. Basically, the thing there is that like actually human motion is not very precise, but it is very good at like figuring out, "Oh! I'm off in some direction," and then correcting for it.

[0:12:09.4] SC: And then fine-grain feedback and correction.

[0:12:11.9] JS: Exactly. That is actually a thing that, again, these industrial robots, they don't really do that on a higher level. They do it on like a very low level of like, "Am I in the right place?" But this gets harder if you have like other objects or something that you're interacting with. This is one area that our research is focused on, basically taking data from the real world and using that online in the loop to have the robot be able to like react to what it's doing and what like the things that it's interacting with which could be like a block or whatever object that

you're working with is doing and if you're grasping it correctly, for example, or if you're moving it to the right place.

[0:12:46.8] SC: How do you do that?

[0:12:48.5] JS: Yeah. That's a very good question. There, we're building on the large body of work that has been done in the ML-based computer vision areas. Basically, we take a convolutional neural network that's been trained on image net to do this image recognition challenge. Then we basically slice off the last layers of that network where it does this thing of like classifying an object into categories —

[0:13:13.2] SC: For aligning?

[0:13:13.8] JS: Yeah. Basically, the parts we want from that is basically the lower level features of the network, like figuring out where edges are and where these like small features are. Then we instead train it to predict the state of the world basically. We're using this machine learning pipeline basically to feed it an image from like a normal webcam or something like that that's attached to the robot and then have this neural network that uses this image to predict like where am I and what state am I and where is the object that I'm interacting with. Then we use that to correct the movement of the robot.

[0:13:48.0] SC: Are we close to doing this? It seems like it will be helpful to do this in three dimensions. Are we close to that at all?

[0:13:54.4] JS: I think so. Maybe not for the general case, but probably if you have basically something where you know at least the little one, like what kinds of objects you're dealing with, then I don't anyone has done it yet, but it's like we think it would probably be doable. The work that we've done, so we use this technique called domain randomization where we basically — Question is of course; how do you train this network? What I just mentioned earlier with like predicting the position of things. The way we do it is basically we create chaos in simulation, because the problem is if you just feed it images from a simulator, the real world will look different and then it will just get confused and be like, "This is weird. Why is there like a pixel in the back that is now bright where it was like dark always before."

The way we get around that is by basically randomizing everything in the simulator. It looks pretty crazy because we basically replace all the textures in this like 3D rendered scene with random textures, like random colors, random patterns. It looks — We used to call it the disco, because it's just random colors everywhere. Basically, this kinds of makes the network robust against variations in the environment. This is actually — Basically, you can measure that, "If you do this, then the real world will look as just like another incarnation of this random setting and it will work. If you don't, it will just be completely off basically."

[0:15:13.9] SC: Okay. One of the projects that Peter Bill is working on is like tying a knot. Using a robot to tie a knot in some string, and he's got an interesting video about this, but one of the comments in the video or the commentary on the page is like, "It works great if the rope is on a green table," but it totally fails if the rope was on a red table or if the rope is like striped or something like that. It sounds like this domain randomization is trying to solve for that same kind of problem.

[0:15:45.8] JS: Yeah, totally. I think for basically rope stuff specifically. The reason why people do this at all is because it's similar to the phone charger, like the phone connector, is that because the robot is flexible. There's not like just like a block or something. Basically, to tie a rope you have to — Either, you have to have some internal model of like how is it going to bend if I turn it this way. I guess people just like work on this and decoupled from the perception way. But this is very plausible if you like put these two results together to create something that can tie knots in an average, very tolerant environments.

[0:16:24.4] SC: Yeah. The other thing that it makes me think of is Apple's recent CVPR paper where they use scan approach to take simulated images, like from a video game engine and kind of make them look like real images so that they would perform better in the real world. Their ultimate goal is to have this robot perform well in the real world. You are trying to avoid over-fitting by kind of doing randomization of your backgrounds and funky colors and all that kind of stuff. Do you still then have the problem of, "Hey, this doesn't look like the real world." How do you approach that?

[0:17:05.0] JS: There's actually a very interesting, like high level questions that like I don't think there's a definitive answer to. It's basically the question of like do you — Basically, should you invest time in making your simulation super nice and photo realistic basically to have like special rendering artifacts of like light reflections and basically just make your simulation be almost indistinguishable from the real world or whether you can actually get away with having really lacking simulation basically that it like very rudimentary. It just has like geometry and just like edges and a couple of like 3D lights and then you can use like neural network regularization techniques to basically use that to make the network robust again.

[0:17:45.7] SC: More generally, in the case of your research, using the domain randomization, what kind of performance have you seen in the real world and have you explored ways to enhance real world performance beyond the domain. Are there specifics to the simulation environment that cause your training to kind of over-fit on simulated looking images?

[0:18:11.0] JS: Yeah. I remember that when we're initially doing this to my randomization work, so the setting that we did there was we basically have a table of like different colors, like wooden blocks. Then we had the robot, basically, you could give it to command something like stack the green block on top of the blue block and then — Basically, correctly localize these box. This was with an accuracy of, I believe, a couple of millimeters It was pretty decent given that this was really just a normal HD webcam, basically nothing that's like super specialized for products equipment.

Actually, I remember that initially sometimes it was just failing randomly and we didn't really know why. Then it turned out that if there was someone standing in the background, then —

[0:18:54.0] SC: Casting a shadow or something like that?

[0:18:55.5] JS: Exactly, or just having like a leg or like a foot in the image, that threw it off. That is actually basically like — These kind of things encourage us to just make the simulation crazier, basically. Then you have these random distractors appear in the background during the training as well, then it will actually be, again, robust to that.

[0:19:15.3] SC: How do you characterize the performance gains after using the domain randomization?

[0:19:22.8] JS: The way we measured it actually is we just had like an actual object tracking system and just measured the error from that. But of course it's also just reflected in the success rate of like how often could you like stack the correct block like on top of the other one. Basically, if it didn't do the domain randomization, if we just like hit — run into the table or something. It was like very, very, very off.

[0:19:43.9] SC: So your performance metric was bad versus good.

[0:19:46.6] JS: Yeah, sort of. Basically what we did actually when we're doing the domain randomization is we — Actually, one of our researchers, Josh, spent a very long time basically just moving around pieces in the real world and like adjusting the camera position and basically just fine-tuning it so it would be like just right. Then it would like appear to work, but as soon as you basically — You could see that it wasn't really working. It was only working if you're like basically manually or fitted to like one very specific instance. Basically, the domain randomization helped with that.

[0:20:21.6] SC: Okay. Did you build a custom simulator to do this or did you use some off-the-shelf thing?

[0:20:28.3] JS: Yeah. Pretty much all of what we do these days is in running MuJoCo.

[0:20:33.5] SC: Okay. MuJoCo?

[0:20:34.3] JS: Yeah.

[0:20:35.1] SC: Tell me about that.

[0:20:35.4] JS: Yeah. MuJoCo is this physics simulator developed by a University of Washington professor, Ema Todorov. It's pretty much I would say the standard in basically these like kind of robotics related tasks. It's also used for all the physics-related environments in

OpenAI Gym. Basically, if you've ever seen the yellow humanoid walking around, that's MuJoCo, and basically having these like capsule geometries.

The good thing about MuJoCo as supposed to like — Of course there are many other physics engines, like game physics engines. There's NVIDIA physics. There's a bunch of — From Unreal Engine. They all bring their own —

[0:21:15.3] SC: NVIDIA announced some new simulation engine at the last GTC. I forget — De Vinci or some kind of name thing.

[0:21:22.6] JS: Flex?

[0:21:24.0] SC: Flex? I don't — I thought it was a person's name.

[0:21:26.3] JS: NVIDIA has a whole bunch of them because they are really making beautiful things. The thing that game engines do for the most part is they are often not super concerned about accuracy of physical realism, which is totally fine. They're just working on making something that looks great and just performance too. MuJoCo kind of comes from the different perspective there where it comes from robotics people basically.

They are using this for like optimal control, which is kind of the analytical way of like how do you control a system to which you were given a task? It has pretty good features for accurately describing an actual physical robot. It can do things, like friction and like tendon-based or where you pull on cables to move the robot around.

It has a bunch of these things that you could implement them like on top of the game engines for the most part, but they just don't come with it. Ultimately, MuJoCo is just like engineering where it's pretty practical. It's just like a library that you link against then you can use it.

[0:22:23.5] SC: Link against, so it was like C++ or something.

[0:22:25.3] JS: Yeah, exactly. Yeah. It's not open-source. It's conventional, unfortunately.

[0:22:30.2] SC: It seems like a lot of the popular simulators are — Is there a dominant open-source simulator?

[0:22:35.3] JS: Yeah, there's bullet. Bullet is very popular, and we've actually been looking at it, like maybe switching to that just because it's — Yeah.

[0:22:42.8] SC: Open-source?

[0:22:43.5] JS: Yeah. That's pretty much it. Because it's just like on like a day-to-day, it's just very convenient to be like, "What is happening? This is weird. Let me just dive into the code and see what's going on."

[0:22:53.6] SC: Also, I would think to make it easier for other people to replicate your results and to try to build on the kind of things that you're doing, you don't have to say, well, you have to first go get a license.

[0:23:04.7] JS: I believe they do have like pre-favorable agreements at least for students. I think as a student you can get it for free. The good thing about MuJoCo is that it's basically that it's like while it's kind of like a walled garden, it's a pretty nice walled garden.

[0:23:20.9] SC: Said every walled garden maker ever.

[0:23:23.5] JS: We're not a garden maker.

[0:23:25.9] SC: No I'm putting the words in MuJoCo's behalf.

[0:23:27.5] JS: That's true. It's true. The one thing where we've actually been eyeing some of the game engine physic simulators as well is actually exactly like dealing with these deformable objects, like cable or the rope or liquids even, like stuff like that. That's something that MuJoCo can't really deal with, because it's entirely like a rigid body simulator and it basically — Its performance scales with the — Basically, it's get slow very quickly if you have a bunch of things like moving around. Basically, of course the scale of things doesn't matter all, because it's just measures, it's just numbers, but if you have like — Let's say if you had like a bowl of full of like a

thousand pearls or something, it will be pretty gnarly in MuJoCo probably. We haven't tried it, but we have tried like — I believe you've basically tried to do like a Jenga, like a Jenga like set up and it was just like jiggle around and then eventually like kind of explodes or just fall apart.

I think it's like a limit of like maybe like a couple of hundreds of basically of like rigid individual bodies flying around, or not flying around, but moving around.

[0:24:32.5] SC: Is that rigid body simulation is something that lends itself to distributed compute or does that not work so well, or does it just MuJoCo not support that?

[0:24:40.8] JS: That's a good question. I believe the creators of MuJoCo have tried to basically run it under open CL, so through like a GPU accelerated. I think the main problem with that is they're actually — It's very similar from the [inaudible 0:24:55.1] computations you would run for running a neural network, where it's basically just like a bunch of like matrix multipliers or like related rated things.

For the physics simulation, you actually have a lot of branching, because you run like a collision detection system and then you do something that's like, "If there's a collision, then do these things," and you do that for like all the collisions in the scene or something like that. That is something that like at least like today's [inaudible 0:25:19.3], they can't deal super well with that.

I think actually in the — This was like a long time ago, but I think NVIDIA, they use to sell something like a physics processing unit, PPU. Actually I'm not sure it was NVIDIA, but like some vendor, and they basically try to make it like an established hardware accelerated physics. This was like marketed to gamers I believe.

Another thing — It really took off, because eventually people realized, actually CPUs are pretty good, and especially with like today where you have a bunch of cores. You could just have a physics core, and at least in the game setting it just works pretty well. That being said, basically today or at like at least the way MuJoCo does it, it's just a single thread CPU processing. It's pretty fast, but I think you'd probably hard-pressed to distribute that specific architecture.

Actually, probably some of the other — Basically, if you had something like some of the game engine simulators are particle-based. For example, NVIDIA's Flex. Basically, they don't represent bodies as like a rigid mesh. There's literally just like a bunch of particles that have some — Basically, some stickiness where they stick to each other applied. I believe there have been some attempts of like distributing that where basically we have cells of the world and every node basically computes all the particles that are like in this specific cell, then they like hand it off to some other node, but we're not using that today.

[0:26:39.7] SC: Okay. Interesting. We're talking about simulation. Tell me a little bit about the, I guess, the process for — The relationship between your training data and your simulator and like how you load all that up and like how you build the simulation. Are there things that you've learned about integrating simulation into AI training pipeline that are maybe non-intuitive or maybe simulators weren't really designed to do this, and so it was kind of hard, but we figured out how to do X, Y, Z.

[0:27:12.3] JS: Yeah. I would say for simulation — This is actually kind of following the — What we just talked about with the distributed aspect of it. One reason why we're not like pushing that super hard is because we actually kind of just horizontally scale it. Actually, the way basically do our training, which is basically running a lot of simulators. You can distribute it in this way, it's just that the simulators are all independent of each other. This is ultimately the same — Like the whole scene, but they all have different — They all basically run different — They all run the same scene or some randomized version of it, but they basically instead of like serializing all the attempts, we just paralyze like 10 or actually more like a thousand of them.

The way we actually run our training is we have like one box that actually does like the TensorFlow, the computation for actually training the network and taking in that data. Then we have a bunch of worker or like evaluator machines that basically take in the current best guest for the policy, which is like our trained neural network. Then they basically roll that out in the simulator, which basically just means they run it over and over again.

This happens on like hundreds of thousands of machines in parallel, but they don't really talk to each other. They just do this on their own. Then they actually send this experience back to the optimizer node, and that's basically where we crunch the numbers to actually improve the policy.

[0:28:38.9] SC: You generate a policy, generate a network. You push it out to a bunch of different nodes and parallel. What's the input to those nodes? Are you giving each of those nodes specific input or are they just kind of running things in random and computing a function or something like that across a random distribution?

[0:28:59.9] JS: Yeah. The workers nodes, they receive the parameters for the policy and there's like some shared configuration for like what's the kind of scene I'll be running? What's a robot and like where are the objects that we're interacting with? What're their initial positions? There are actually a couple of different ways of how to communicate the results back to this like central mastermind machine. One way is if you actually have the workers do the gradient computation, they roll out the physics simulators and then they figure out some kind of reward or cost function for this and like once there's a good roll out, like we end up in a good state where like for example when you're moving these blocks around, the red block on top of the other red block and then the workers figure out, "Okay. If my parameters were like treated slightly differently, I would have gotten a better outcome this time."

Then the other approach is to just send over the raw, like what happened in simulation to the optimizer machine then let the optimizer figure out what the parameters should be and like how we should change them.

[0:29:59.6] SC: Which do you tend to do? Do you do both?

[0:30:01.9] JS: Yeah. Right now we do the sending over the experience, so the later thing, which is just because it's simpler for the most part, because then basically the workers are kind of dumb and they don't need to worry about that much. There's a situation where this can actually would be problematic especially if your observations are big.

For example, if your observation, which is like what's your policy or your agent is using to make a decision for what to do next, that is something big, like an image from a simulated camera, that you don't want to send that over the network just because there's so many of them and just clogs up the entire bandwidth. That's where we probably look into the — And just switching to

the — Sending over the gradients over the network over the next few months. Especially as we move through these more like high-dimensional observations.

[0:30:43.6] SC: Is the infrastructure that you're doing all this with, is this a hand-crafted stuff or — Like I had a conversation with Ian Stoica about Ray the other day, like are you using something like Ray to do this?

[0:30:55.9] JS: We're actually using Kubernetes. Yeah, I believe we did like an infrastructure blog post about this a couple of months back. Basically, we run lots of Kubernetes clusters across all of the major clouds as well. Like we're running on Azure, we're running on AWS. We also have some stuff running on Google. Yeah, there's a lot of compute. The way you actually augment this using these tools is actually somewhat simple. You basically just tell Kubernetes, in our case, to, "Here's this batch job. Learn this thing once for the GPU optimizer and run this thing once for the couple of hundred worker machines," and just kind of goes and does it.

But there is a ceiling there where like you can't — Just because this is not really the original use case for Kubernetes. I believe like what they recommend as a limit is like not more than, say, 10,000 nodes, which is a lot of course.

[0:31:49.0] SC: Have you bumped up against that limits?

[0:31:50.5] JS: We have bumped against previous limits, where the limit was something like 5,000. Then it won't crash but it will just become like more and more unhappy and just stop responding. Basically things will become weird in the cluster.

[0:32:03.4] SC: How long are these simulation jobs? They're not like on the order of training jobs that are days and days. They're much shorter? Is that the case here?

[0:32:12.4] JS: Yeah. Basically, one specific cycle is much shorter. One specific cycle of basically get the current policy parameters, do a bunch of roll outs, send them back. That's something like maybe like a second. But the way we actually start them is we basically just keep these roll out machines around the same way that we keep the training.

Basically, they're just one temporary cluster basically, and that cluster will just stay around for the duration of the entire training, which is maybe like a day or half a day.

[0:32:43.0] SC: Interesting. All these is to help you develop kind of a better model via simulation. What do you do when you want to test that in the real world?

[0:32:53.9] JS: Yeah.

[0:32:55.6] SC: Kind of an important part of this, right?

[0:32:56.6] JS: Yeah, exactly. Yeah, the interesting thing is that in a lot f other areas in machine learning, they just kind of stopped short of that. They're like, "Well, this model looks pretty reasonable. So it's fine. We're done here."

We're actually really adamant about like jumping through all the hoops to actually make it run on the robot, because we found that it helps like keep us honest basically and just like it's very to be impressed by cool stuff happening in the simulator, but usually there's some caveats that make it harder, for example, if like some of the robots meshes like aren't actually touching each other or something.

[0:33:31.3] SC: What's a robot mesh?

[0:33:32.9] JS: Basically, that's kind of the — Just the shape of like some part of the robot, like the limb of the robot. Usually, for example, for when you're determining whether like the robot is pushing something, you have to do this like collision test of like is the robot colliding with an object. If yes, then you push the object away.

The thing is like while you have this super nice like visual rendition of the robot with like a nice like it models all the aspects and screws or something. Often, the geometry is actually used for this collision check which actually determines what's happening in the physics-wise might be like a simpler version of this geometry. It might just be like a cylinder where the actual robot is something super complex with like rounded corners or something like that.

[0:34:14.2] SC: Your angles roll off and kind of this [inaudible 0:34:15.6] just compounds.

[0:34:17.3] JS: Yeah, exactly. At the same time, this can also like kind of trick you into thinking about something works, where it wouldn't actually work if you tried it in the real world right now.

The way we run, we actually run our policies on the real world, we have developed a system called robots as a service, which basically means that the robot goes under the network and then people can connect to it and like run specific algorithms or like the models that they trained no the robot. There is some like specific technical things that is non-trivial because it's just like a real-time environment that you can't just — Basically, it's much harder than to simulate it in like a piece of software, like people often do with Atari games, for example, for training. You have to be careful that you don't miss like a cycle, because otherwise your policy will get super confused, because your time sample will just be longer, for example. Basically, working on all these artifacts has actually turned this robots as a service system into a pretty big engineering project here.

[0:35:16.3] SC: The example you gave struck me as something as more of a training artifact and training issue as supposed to deploying it to the physical robot. How does it manifest itself on the physical robot in such a way that you could do something about it there?

[0:35:32.2] JS: Yeah, the specific example of like — In simulation, you would start at like T=0 and you get some like current state, you decide what you want to do and you set the action. Basically, while you're thinking about this, simulation is stopped, it's paused. Basically, you do its computation, then you get your output and then you advance the simulation by one step and then you repeat. You think again and you step again.

In the real world, of course, all of these happens simultaneously and you don't get the chance to just like pause and think for a while.

[0:36:01.9] SC: You've got inertia and continuous variables and all these stuff that —

[0:36:05.8] JS: Exactly. There is a very interesting question, like is it a trading problem or an evaluation problem? Actually, for most of the things, these things it can be both. As sticking with

the timing discrepancy example. There's basically always — This is pretty much like a decision we have to make for every issue similar to this that comes up, is like are we okay with like investing time in actually like ironing this issue in our software stack or should we just be like, "Oh! Well, if there's fluctuation in the timestamp, then we can just train with that and basically just add that to the set of things that will be randomized in simulation."

Basically, there's always a balance of these two choices. Either make the simulation harder or make your real world system more predictor to execute. This is kind of an area where you have to pick your battles to some extent, because if you just have too many unknowns, then it's wild. Like there's a theoretical — It should be theoretically possible for you to be able to learn a model doesn't cope with all of these uncertainties. It will just very hard in practice to debug it and inspect it and see if something goes wrong and you can't — Basically, you will see it break and then you'll be like, "Well, why did it break?" Basically, getting to the bottom of that requires you to exactly do this part where you remove like as many of the unknown as possible.

But it's really rad that basically it — in the end, our policies, eventually we want them to be capable of — Basically, you put them on new robot that is very terribly instrumented or has like really weird software like causes like lags and delays and just figure it out. I don't think we're there quite yet.

[0:37:37.2] SC: Okay. You call this robot as a service. Is it like on-demand, like you have a bank upstairs of like thousands of robots just swinging around or do you have a robot or two connected to the network that folks can like — There's a calendar, an Outlook or whatever that they schedule their robot time with? How cloud-like is this?

[0:38:00.6] JS: I really wish it would be the former, but it is in fact closer to the latter. We actually thought and actually some other. Some folks over at Google, I think it was [inaudible 0:38:09.1] they bought a bank of robots. They bought I think like around 50 or something robot arms and just had them do these grasping tasks over and over. We thought about doing that too. But at least with the current set up, it's effective. We have to fetch. We have a couple — We have a couple of other like robot arms. These are like individually connected to the network. We usually do the scheduling by like whoever is around the robot at the time.

[0:38:34.8] SC: Scheduling about proximity?

[0:38:36.4] JS: Exactly. The main reason why they actually need like a specific system for like accessing the robot there is that — These robots all come with software. They come with some kind of software. They come either with some integration for ROS, which is Robot Operating System, which is a big open-source effort to provide like a unified framework for doing all [inaudible 0:38:58.1].

[0:38:57.0] SC: You're not up at ROSCon? That's going on now I think or this week.

[0:39:00.7] JS: Oh! Is it going on right now?

[0:39:01.3] SC: In Vancouver.

[0:39:02.7] JS: Oh, great. Shout out to ROSCon. We're actually not using Ros.

[0:39:08.3] SC: Well, that explains it.

[0:39:13.8] JS: Yeah, we found that Ros is actually really great if what you have is you have a robot and then you have a bunch of like other things around it that you want to use. For example, you have your robot arm and then you might have like some like a live [inaudible 0:39:27.8].

[0:39:27.0] SC: Right. They've got a bunch of modules.

[0:39:29.2] JS: Yeah, exactly.

[0:39:29.9] SC: It's very extensible.

[0:39:30.7] JS: Yeah. You have like tracking cameras or you have like — Maybe you have like one robot that is your arm and then one robot that is your gripper and they need to be independently controlled, something like that. We found that it's really great like all these things

already ship with drivers for us and you can just plug them together and they will just work pretty much out of the box.

I think for us it kind of comes back to the issue we talked about before this way, where they're just timing uncertainties basically and you don't really know what's going on inside the system. For example, if you had this case where you have a robot and you have some external sensor and in Ros they would just appear as like, "Here's your robot. Here's a sensor. Great. You're all set."

Actually, there might be like subtle things where for example the timing updates for both of these systems might be out of phase where like the robot updates and then there's delay and then sensor updates and then you will have a timing lag there. While this wouldn't matter like for a lot of the classical applications where you do something like where you collect data over 10 seconds or something and you're basically doing it in a very slow way, and then you don't really care about what these like tiny differences. They'd caused problems for our case, where basically we try to instantly react to like if something change in the environment, you might be at tens of milliseconds before we feed the spec into the policy to correct for that.

This is why we actually have to be like super careful about what are the exact timing phases of all these sensors, and so we found that a lot of the Ros abstractions are actually — They kind of encapsulate this and hide it from you, which is actually great I think for the majority of use cases which doesn't work well for us, so we actually do need full control over these things.

[0:41:17.2] SC: Okay. Do you write your own operating system or is it more like less like an operating system, more or like a thin layer of it? I'm assuming that the bottom of this is all kind of just you're controlling stuff for motors and stuff like that through I/O ports. There's kind of be some kind of — You want some kind of software layer there to make that a little easier. Did you just wrote that yourself?

[0:41:39.1] JS: Exactly. It's pretty much like a layer of middleware basically. We didn't really write our own OS system. For some of our robots, we basically get rid of all the software that they ship with and just basically like if there's some firmware on like some embedded

microcontrollers on the robot then we usually won't touch that, because that will also usually be fast and predictable, like well-defined.

[0:42:02.4] SC: The issue with the off-the-shelf option was like code path [inaudible 0:42:08.9] variable, because they are accounting for plugging in like external modules and all that kind of stuff. Is it that?

[0:42:14.7] JS: Yes. There's definitely a lot of complexity in that — Like they basically have this entire framework for being like plug and play, but they're actually also like real operational issues where — For example, the timing issue I just mentioned, like wouldn't be because they like — It's not that crazy basically that we have to like make sure that our code has like constant execution time or something. It's just you have to like — For example, usually these systems have some kind of trigger signal and you basically just have to lay your code out in a way that like know my cycle starts, you triggered all the systems. Basically, you synchronize all of them and then you read out all the data or something like that. That's how you synchronize all these things.

It's not so much like really dark magic of like performance optimization. It's basically for the most part we just have like a very specific usage pattern that requires like carefully thinking about like basically when do we do what.

[0:43:07.3] SC: Pulling these all together, you're trying to develop a technique that allows the robot to be more like the human and can kind of do fine tune cores correction as it is operating, and you train all these models and simulation. How do you then use that with the robots and do the inference to make those course corrections? How does all that part work?

[0:43:34.5] JS: Yeah. Basically, right now, what we're still aiming for is that we actually don't do any fine-tuning basically of our model in the real world. We train a model simulation and then we basically just throw it out on the real robot. Like you will gather all your sensor data like from the robot and from your external cameras and tracking systems and stuff like that and you feed that into the policy, do the inference and then you react to that.

Basically, the adaptation loop for figuring out like differences between the real world and the simulator right now is actually — At least for us, it's pretty much manual. There are lots of ideas for basically doing this kind of like meta-learning where you learn to learn to adopt to a new environment. We've had some initial success with that for basically imitating a human doing some behavior, and then you would figure out, it's like, "Oh! What are the semantics that the human intended to achieve with this task?"

I think this is what we — Like more like general statement. This is stuff is still pretty early both in the robotics and the ML communities, but it is super interesting. Eventually, we'll want to do something where like we have a bunch of the different simulators we talked about. Might even have like different kinds of robots, basically just increasing the breadth of this distribution so you encapsulate more and more things and really hope that the policy gets to the bottom of like, "I see. This is how I learn to control a new robot in an entirely new environment." That's something we're super excited about it. Eventually we're hoping that this will enable a robot that can truly solve a variety of like very complex tasks on a variety of different robot platforms.

[0:45:16.5] SC: Okay. If I wanted to learn more about this, dig in to the details, like see it in action, have you published code on this? What will be required for someone that kind of play with this and try to replicate what you did?

[0:45:31.7] JS: Yeah. We did a pretty big release a couple of months ago where we basically put some of the things we talked together, like the domain randomization part and the part where you observe a human and figure out, "Okay. I want to do this specific task and the robot imitates this in a new setting." We have a release there where we basically show how it works and like show how the networks are aligned and like basically how we feed the data from one to the other and how they're trained.

[0:45:56.8] SC: Okay.

[0:45:58.4] JS: We'll probably be publishing at least parts of this robots as a service later that I just talked about together with like our next set of research results there. We figured it doesn't really make sense for them to just be like kind of their own, because then you —

[0:46:10.9] SC: There's a lot of moving parts, right?

[0:46:11.5] JS: Exactly. The issue there is that like even if you open-source a code, and it's like, "Well, great. You can started today. Just add a really expensive robot to the mix." We'll definitely be — As we publish our research, we want to release both the research and the infrastructure parts required for it.

[0:46:27.2] SC: Okay. Cool. Awesome. Jonas, thanks so much for taking the time to chat with me about this. It's really cool stuff.

[0:46:33.0] JS: Awesome. Thank you for having me.

[0:46:34.5] SC: For sure.

[END OF INTERVIEW]

[0:46:38.6] SC: All right, everyone. That's our show for today. Thanks so much for listening and for your continued feedback and support. For more information on Jonas or any of the topics covered in this episode, head on over to twimlai.com/talk/76.

To follow along with our OpenAI series, visit twimlai.com/openai. Of course, you can send along your feedback or questions via Twitter to @twimlai or @samcharrington or leave a comment right on the show notes page.

Thanks once again to NVIDIA for their support of this series. To learn more about what they're doing at NIPS, visit twimlai.com/nvidia.

Of course, thanks once again to you for listening, and catch you next time.

[END]