EPISODE 850

[INTRODUCTION]

[00:00:00] JM: Machine learning allows software to improve as that software consumes more data. Machine learning is a tool that every software engineer wants to be able to use. Because machine learning is so broadly applicable, software companies want to make the tools more accessible to the developers across the organization. Because in a limit, every software engineer will be able to take advantage of machine learning.

There are many steps that an engineer must go through to use machine learning, and each additional step inhibits the chances that the engineer will actually get their model into production. An engineer who wants to build machine learning into their application needs access to datasets. They need to join those datasets. They need to load them into a machine or multiple machines where their model can be trained.

Once the model is trained, the model needs to test on additional data to ensure quality. If the initial model quality is insufficient, the engineer might need to tweak the training parameters. Once a model is accurate enough, the engineer needs to deploy that model to production. After deployment, the model might need to be updated with new data later on. If the model is processing sensitive of financial irrelevant data, a prominence process might be necessary to allow for an audit trail of decisions that have been made by the model.

Rob Story and Kelly Rivoire are engineers working on machine learning infrastructure at Stripe. After recognizing the difficulties that engineers faced in creating and deploying machine learning models, Stripe engineers built out Railyard, an API for machine learning workloads within the company. Rob and Kelly join the show to discuss data engineering and machine learning at Stripe and their work on Railyard.

A few quick updates from Software Engineering Daily land. FindCollabs is a place to find collaborators and build projects. FindCollabs is the company I'm building and we're having an online hackathon with $2,500 in prizes. If you're working on a project or you are looking for other programmers to build a project or start a company with, check out FindCollabs. I've been interviewing people from some of the projects on FindCollabs on the FindCollabs Podcast. So if you want to learn more about the community, you can hear that podcast.

We have a new app for Software Daily on iOS. It includes all 1,000 of our old episodes as well as related links, greatest hits, topics. You can comment on episodes. You can have discussions with other members of the Software Daily community, and you can become a paid subscriber for ad-free episodes at softwareengineeringdaily.com/subscribe.

With that, let's get on to today's show.

[SPONSOR MESSAGE]

[00:02:59] JM: Software Engineering Daily is a media company, and we run on WordPress just like lots of other media companies, although it's not just media companies that run on WordPress. I know of many organization that manage multiple WordPress sites and it can be hard to manage all of these sites efficiently.

Pantheon is a platform for hosting and managing your WordPress and Drupal sites. Pantheon makes it easier to build, manage and optimize your websites. Go to pantheon.io/sedaily to see how you can use Pantheon. Pantheon makes it easier to manage your WordPress and Drupal websites with scalable infrastructure, a fast CDN and security features, such as disaster recovery.

Pantheon gives you automated workflows for managing dev, test and production deployments, and Pantheon provides easy integrations with GitHub, CircleCI, Jira and more. If you have a WordPress or a Drupal website, check out pantheon.io/sedaily.

Thanks to Pantheon for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:04:21] JM: Kelly and Rob, welcome to Software Engineering Daily.

[00:04:23] KR: Thanks. Great to be here.

[00:04:25] JM: You are both engineers on machine learning infrastructure, data infrastructure. Describe some of the use cases for machine learning at Stripe.

[00:04:34] KR: So in terms of machine learning at Stripe, I think every company that does machine learning has kind of like the first thing they started doing and then how it grew from there. For most companies it's either initially some internal analytics or some application that's like critical to your business, like ads or spam come up a lot.

For Stripe, that kind of critical production use case was a couple of different types of risk, both like evaluating for each transaction that comes through Stripe, is that a good transaction or is it fraud, which eventually became our product radar as well as kind of for ourselves understanding our users and like whether we might have some users that for different reasons we might not want to support.

So that was kind of how we started it out with machine learning and production and overtime we've kind of grown a bunch of other cases especially as we've built like a little bit more of a platform to enable it. So in our product, in addition to radar, that's our fraud product, we also do, for example, smart retries of failed subscriptions in our billing product.

We use machine learning a lot internally for different types of risk, like I mentioned, like is this a good user? Is this your user selling something that we can support? We also use machine learning a lot internally for analytics to understand kind of our users and who we might want to reach out about something or like to help us suggest resolution paths for our customer support, for example.

[00:05:53] JM: With this variety of machine learning use cases throughout the company, you've got probably some people who are classified as data scientists. Some people who are machine learning engineers. Maybe people who are classified as researchers. How do the internal user types vary in their technical proficiency and how they want their machine learning tool set to be presented to them?

[00:06:21] KR: Yeah, that's a really, really excellent question that also has diversified a little bit overtime for us. Early on, especially for our production focused applications, we hired what we call ML engineers who are kind of like the unicorn where they can do a little bit of all of it, like they are good software engineers with good programming skills. They also know statistics and data science. So our early tools were kind of both for people who really knew a lot about a lot.

As Stripe has grown, we've seen kind of different types of users who want to onboard to our platforms. In some cases, people who are more traditional data scientists who know a lot about machine learning and stats, but are maybe less software engineers. Also, product engineers who know a lot about their product and how to build it but maybe aren't as familiar with machine learning. So I think that's been one of the challenges and one of the things that, in some cases, we've been able to kind of – As we have a better platform onboard different skillsets, but also that we're kind of thinking about going forward in terms of how to make things easy for people and give them the experience they want when you do have this sort of different set of backgrounds.

[00:07:25] JM: Both of you work on data infrastructure at Stripe. You have data infrastructure and machine learning infrastructure. I'm not sure what the division there is. But I consider both of these forms of "platform engineering", where you're building areas of the infrastructure that multiple different teams within the company are utilizing. How does platform engineering work at Stripe?

[00:07:54] KR: Yeah. That's a good question, and machine learning infrastructure is actually part of our data infrastructure group and we have a little bit of like we're our own users where machine learning infrastructure builds basically all of our other data teams, like the storage teams, the batch end streaming platform teams. Then that kind of sits within what we call kind of foundation engineering, that's a very expensive definition of infrastructure, where basically these are all the teams whose users are other teams at Stripe who, like you said, sort of provide these almost like services to the company that other engineers or data scientists or analysts can use. When we think about like are we making our users happy in building what they want? Those are all kind of people we can talk to inside the company.

[00:08:36] JM: Rob, what's the relationship between a platform engineering team at Stripe and the different business logic teams at Stripe? The kinds of teams that might be utilizing tools given to them by platform engineering.

[00:08:54] RS: Yeah. I think this gets a little back to Kelly, what Kelly was saying about, the ML engineers, where those ML engineers are really the bridge between sort of the product-focused application side, like the uses of ML in our product and then the application and actually like doing all of the software development and modeling to enable that.

Basically, those folks are our customers. The ML engineers are our customers. They say, "Hey, we're trying to do ML for this product or for this application and sort of here are our needs from the infrastructure side. Here are the needs from you all."

So we work with them to build the tools and services to help them do their jobs and then sort of we're also working with the other data platform teams for our needs, right? So we're sort of at this like interesting middle spot and the data stack where we're building infrastructure, but we also have infrastructure needs below us for the things that we're building and then we're interacting you directly with those applications and product engineers to help them build their software and the models.

[00:09:51] KR: Yeah, and our ML infra not only relies on kind of all of data, but also sort of like all of Stripe's infrastructure engineering, like a lot of what Rob did in this work was to work with our orchestration team that runs our Kubernetes clusters. So we really are kind of like in the middle of the stack where we both are users of a lot of other Stripe teams as well as supporting many other Stripe teams.

[00:10:11] JM: Maybe this would be a better question for the orchestration team, but how has Kubernetes affected infrastructure at Stripe?

[00:10:20] RS: I can answer that sort of from our team's perspective. So, Railyard, a machine learning model training service, we run jobs on Kubernetes, but Kubernetes is actually at a much bigger effect for us in terms of like the services that we operate. So, one thing that we do is we generate features for models like real-time in production. So we sort of have like a streaming system that generates these feature values in real-time.

Pre-Kubernetes, a way that we would do this is we'd – In AWS, we'd stand up a bunch of EC2 instances. We're basically running a little cluster of these things for every big feature set that we had, and there were just a ton of toil and management for us to manage like all these service clusters. I think we got to the point where like our team was running like over a hundred instances just to ourselves of these little services, these streaming instances, and we're sort of managing these big clusters of streaming services.

So Kubernetes like really took a lot of that burden off of us. We're able to move most of those clusters to basically like Kubernetes pods, and now to stand up one of these things, we sort of have a declarative specification that we can write to stand up a new one. Whereas previous, it was like, "All right. How do we get a new one to stand up?" "Well, it'll probably take a week or two." Now we can do it in just a couple of days, because the orchestration has built a platform that lets us fairly quickly sort of get these new streaming clusters up for us to do a real-time feature generation. That's been a huge impact for us. It was very, very toilsome for us to do this before. Now, it's reasonably quick.

[00:11:50] JM: With the reduction of friction in standing up infrastructure, what are the downstream advantages for teams like yourself that are building platform infrastructure so you get to take advantage of the lack of friction from Kubernetes and perhaps – I don't know, other benefits reliability and so on. How has that impacted your work?

[00:12:17] KR: Well, there's kind of like two ways in which we're impacted. I think Rob talked about one. So like on the one hand, for our infrastructure teams, if we have services that we can onboard to Kubernetes, it reduces like our toil and the time we spend managing them. Then the flipside of that and sort of the other advantage is that we can do more for our users. In some cases, that's just like redirecting time we would have spent managing instances into time like building products for them.

I think in the case of the work that Rob did with Railyard, it also actually produces a better platform, not just one that's less work to run where we can give them a lot more flexibility about like what types of resources they need to train their models like better solution from what other people are doing. So I think it's not just like less work to run. It's like actually a better platform.

[00:13:06] JM: The story that we have told so far is that there are a diverse set of teams throughout Stripe that need machine learning infrastructure. They need to do machine learning jobs for fraud, risk and a variety of other tasks. Basically, Stripe is a numbers company. You've got a lot of numbers. You want to have machine learning on a lot of those numbers. So you want to have tools for people to apply machine learning to their numerical calculations. So you set up these infrastructure teams, like a data infrastructure and machine learning team to offer a platform, to offer internal services to these different business logic users.

Your team is that data infrastructure team. In order to build the right tools, you have to interface with those teams. You have to figure out what the problems they have in setting up their data infrastructure, in setting up their machine learning models. What are the challenges that you hear repeatedly from engineers who are building – Maybe it's an internal accounting tool. Maybe it's a fraud and risk model, but the chronic challenges of somebody building machine learning infrastructure.

[00:14:21] RS: That's a good question. I think one of the biggest challenges, and this isn't actually one of the things that Railyard always put through. We've got some other tools and services that help with this. This sort of understanding what data do we have? What does that data look like in terms of building features and then how do we sort of turn these features into something meaningful for a machine learning model?

So, I think that all the teams across a company, like question one when they sit down and they say, "I want to build a model," is what data do I have? Critically, what labels do I have? Not just the data that turns into features, but sort of what is the label for this thing.

So given these features, for example for fraud, it's like is it fraudulent or not? That's an example of a label. With like the smart retries thing, it's like, "Did the retry succeed or not?" That's another example of a label.

I think a lot of the challenges, especially upfront, is around sort of figuring out what data do we have? What data do we have labeled? Then moving in into sort of the modeling piece. To some degree, the modeling piece kind of actually like building the model, training the model is kind of the easy step compared to actually like doing a feature engineer and understanding the data. That is a lot of the work in the machine learning process, is just getting a handle on sort of what data you have. Whether or not how clean that data is. Whether or not you have a messy dataset and then whether or not you have labels for sort of the problem that you're trying to model.

[00:15:51] JM: It's interesting to hear you say that, because in preparing for this show, thinking about Railyard, which we're going to get to talking about eventually, which is the platform that you have both worked on. I read most of the blog post, and I don't remember seeing data discovery as a problem. It doesn't surprise me at all that data discovery or data understanding, metadata understanding of what datasets you have available to you, what those datasets consist of. Do you have an internal data discovery platform where engineers can kind of like dig through what – Because, I mean, for people who have not worked at a company like Stripe, you've got S3 buckets. You've got SQL databases. You've got Mongo databases. You've got databases on databases. You've got databases of databases.

If you want to build a machine learning model within Stripe, you might be able to leverage datasets that are in a totally different area of the company. So, how do you solve data discovery here?

[00:16:55] KR: Yeah, that's something we thought about a lot, and I think Stripe is also a company where we have a lot of people who want to dig in. We have met probably a majority of the company NoSQL and like people are sort of really advocates for looking into things themselves and not just asking other people.

A couple of years ago we built some internal tooling focused particularly at the SQL side to help people kind of navigate of the data they have access to, like what's in it or like be able to kind of easily write queries and make simple graphs. So I think we continue to try to empower making something like that available.

For machine learning specifically, the blog post focuses on Railyard, which is the training platform, but there are a few other complementary pieces. There's like how do we do scoring in real time? Another big piece is like how do we do feature generation? We have done a lot of work on having kind of like a framework for feature generation and increasingly making it support like sharing of features between related applications.

[00:17:54] JM: Not to disorient you, but what is feature generation? The people who've done that.

[00:17:56] KR: Yeah, good question. So feature generation, this is basically what Rob was saying, is let's say you want to train a model. Usually, you have this idea that like you'll turn some data into some matrix and then you'll feed that into some algorithm and that will give you your model. Basically, each row in the matrix you feed in is like an observation and each column in that matrix is a feature.

So a feature is basically like sort of like a fact about something. So it might be – If I'm trying to predict fraud, like what country was the card issued from? How many countries has this card been used from in the past to today?

[00:18:32] JM: This is creating and embedding space.

[00:18:35] KR: And embedding is kind of one type of feature, but you can also have features that are like counters or features that are just like simple facts.

[00:18:40] JM: But in any case, you're looking at different examples and putting them in some high-dimensional space, right?

[00:18:49] KR: Yeah. So you look at like different, let's say, event streams that come in, like charges or like login events or many other things and you can either try and use those wholesale or you can turn them into embedding or you can kind of like aggregate over them. Basically, like find different ways to pull things out, combine, or look at the data that then you can feed into your model.

[00:19:08] JM: So you might have a set of credit transactions and you might take those credit card transactions and build additional features on top of them, like look at the data among them and then build features on top of them, or are you just saying that in each of those transactions, the features that they have within those transactions, that's enough data to build your model.

[00:19:31] KR: Yeah. So you can and we do do all of the above. So you can pull off, like here's one element of that transaction. You can say like, "Let me look across all the transactions or all the transactions on this card or whatever attribute and kind of do some sort of aggregation. You can do joins like across different event streams, and that's something we support in real-time, or like you were saying, you can maybe like try and create, pre-learn some embedding and use that as a feature.

There's a lot of different options, but one of the things we try and do as a platform is find ways for people to leverage related investments, so like teams who work on radar or fraud product and teams who work on risk internally. Well, a lot of the features they look at, you could kind of use for both applications. So we try and make our platform sort of like support that like more naturally both from a discovery point of view and also increasingly from like a computation point of view where sometimes these computations are like quite large and maybe there are also pieces that we can reuse at that as well.

[SPONSOR MESSAGE]

[00:20:37] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:22:29] JM: So if I think about the job of somebody who wants to build a model for detecting fraudulent transactions, for example, I might take a set of transactions. I look at the features that are just in those transactions, maybe they take place in a certain geo. They take place in a certain timeframe. They're from a certain merchant, and maybe I can look across the features that are just in those transactions and I can find correlations. I can find interesting correlations and use that to create a model for doing classifications in the future, or maybe I want to look at those datasets and join them with other datasets throughout the company in order to kind of enrich the features that I have available and, again, create model for finding correlations that will help me identify future transactions as being fraudulent or not.

[00:23:23] RS: I think that's pretty spot on, and you can start with sort of the features in an individual row. But I think that we've seen to build really good models, you do end up having to use sort of these aggregates and sort of joining other data sources in across the company.

[00:23:38] JM: I went off on a tangent a little bit. So the data discovery question, is that a solved problem within Stripe or is it still kind of unsolved?

[00:23:46] KR: I would say something we have some solutions for, but I don't think we've like solved the entire space. I think in some cases, like I was saying, we've built some kind of internal UIs and tools where people can do some data discovery for machine learning specifically by having this shared feature's framework that allows kind of some sense of that.

Then one of the things that we're trying to do is just like make that even easier, and like I mentioned also, if we can share more of the computation, because a lot of what people find is that machine learning is computationally intensive and a lot of the time you want to be iterating as quickly as possible on like try this new feature, get this new model out. The less computation you have to do in order to do that, the better. So if you just added one new thing, it's cool if you don't have to re-compute everything. I think that's something that the ML infra is kind of like thinking about a lot and trying to improve for users.

[00:24:40] JM: Before we get into Railyard, just to go through a little bit more context on the day-to-day life or somebody who is building and iterating and maintain machine learning models. I might build this system for detecting fraudulent transactions. Overtime, I want to feed new data into that model. I want to look at new features and update my model with new features. I may want to do AB testing to see what leads to more fraud coming through my system.

This starts to look something like a continuous delivery problem where continuous integration problem where you start to say, "Okay. How am I iterating on my program that happens to be a machine learning model? How do the problems of machine learning deployment and maintenance manifest at Stripe?

[00:25:43] RS: I think that you've sort of touched on and this will actually be a nice segue for Railyard, because sort of the things that you just described or some of the really driving forces and motivators to build Railyard in the first place.

A lot of the things, like sort of training models and deploying models and building features in that sort of thing, in the early days of Stripe, that was basically – A human would sort of SSH into an EC2 instance to do these things by hand, right? Getting these things out the door were sort of a log process, because a human would have to be involved every step of the way. As Stripe has grown and as our machine learning infrastructure team has sort of built more of these tools and services, what we're trying to do is make that process as streamlined as possible.

Computers are really good at doing things once a day, for example. It's really pretty straightforward to have a system that like trains a model once a day. That's a thing that you can get a computer to do rather than a human to do. Same thing with sort of generating features on a daily basis, with deploying a model on a daily basis. You can automate all of these things.

I think there're still some things that you want humans in the loop for, right? When it comes to sort of like deploying new models to production, that sort of thing. In some cases, or in most cases, we still have someone actually hitting the button and hit to say go, or like launching a new AB test. That's something that – Someone will actually set that up manually, that sort of thing. But a lot of sort of the upstream things, the actual retraining. We're trying to get to a world where that is as automated as we can make it.

[00:27:07] JM: Tell me about the process of specking out Railyard. What is Railyard? What were the objectives in getting it off the ground and what were the early days of building Railyard like?

[00:27:25] RS: Yeah. So, the first impetus for Railyard was – I kind of just described this process of sort of humans like kind of manually training a model and then deploying that model, fetching features, that sort of thing. One of our customer teams, one of ML infrastructure's customer team came to us and said, "We'd really like to take the human out of the loop for this. We'd really like to retrain our models on a daily or weekly cadence and we'd like this to large be hands off. Not only would we like to train just one model, because with just model, maybe it's okay for a person to just train it and say, "Okay. I'm done."

They were interested in training tens or hundreds of models at a time. For that, we're just like, "Okay. It doesn't make sense for a person to be doing this. We have schedulers like Airflow. We can build software to do this." But, critically, what we need, is we needed API in front of the model training process. What we really want is a way to sort of make an HTTP call to train a model. Not fire up a process on the [inaudible 00:28:26], whatever. So we sort of wanted to separate concerns there a little bit, right? We wanted to have kind of the upstream components that were responsible for orchestrating that I want to retain this model on this data. Then ML infrastructure was going to sort of build this API to do this training.

So, we talked to a bunch of customers. It was really focused on sort of fraud classifiers at first. That was like the big use case. One thing we mentioned in the blog post is sort of how Railyard has been used in ways that we didn't really anticipate at the time. But the first use case was sort of very focused on sort of this fraud classification problem. We already had some pretty good sort of Python task run our code. Again, it was in the genre of a human runs a command and does some Python things.

So, the idea was, "Okay. What if we put an API in front of this?" We don't change the Python code that much. We sort of execute the same Python code, training code that we're executing before. But we've restructured a little bit so that the interfaces on the edges are a bit nicer so that we have something that sort of looks like steps of a workflow so that someone who wants to train a new model, they don't have to get into the super nitty-gritty. That they can sort of hand-off the responsibility for a couple of things to us.

For example, they can say, "I want to fetch these columns, the columns of these data for this timeframe. Railyard, please go fetch that for me. You hand me data and pandas data frame and an interface that I understand. I'll train my model and then I'm going to give my model to you and I'd expect you to serialize that model and serialize all my evaluation data and make it available for scoring so that I just really have to focus on the model training piece." So, that was the idea in the beginning. It's like we were going to build an API for model training. This API was going to enable sort of our upstream product teams to build services and software on top of it and be able to train models rapidly and do so with as little friction as possible.

[00:30:19] JM: Okay. An API for model training. The way that people think about training a model today in most cases is like we went through – First, I've got to find my datasets. I've got to find out if I have permissions to them. I've got to wrangle the datasets. I got to figure out the features that I'm using. There's like a 10-step process that requires all these disjoint pieces.

If I understand correctly, the vision for Railyard was to unify that into an API where almost in the same way that like Kubernetes gives you this declarative interface to doing a lot of complex stuff under the hood. You would want to specify just in a single API request, "Here is how my model should be trained."

[00:31:11] RS: Yup, that's correct. For the model exploration piece, folks are still doing a lot of model training or doing a lot of the exploration like in a Jupyter Notebook, for example. There is still like the upstream part of model training, the actual feature discovery and that sort of thing. Railyard doesn't help as much with.

One thing that we've worked on a lot is sort of letting people – Giving people ways to do that exploration within Railyard's interfaces. So, Railyard kind of has – There are two ways to execute the Python code that Railyard executes. One way is to make an API call to the Railyard surface that spins up this job on the Kubernetes cluster and trained you model, and we think of that as like the production path.

You can also execute the Raiyard Python hooks by just calling a little command line tool and running your Railyard workflow locally. So, sort of what we moved to is people run this local execution of the Railyard Python code and they iterate quickly that way. So they iterate, they're trying out different features. They're tinkering. They're changing their API requests because they execute this local runner with like a JSON blob. So they're iterating, iterating, iterating locally. They might run this thing a thousand times. Then they get to the point where have the features that they want, they have the model that they want. They think the results look pretty good. At that point they say, "There's sort of this threshold where they're like, "Okay. This is "production ready". I'm ready to train this thing daily.

Where in the iteration process, they might be changing sort of the Railyard specification a lot. They might be trying a bunch of different data features. They might be trying a bunch of model parameters. But then in the end what we've seen is they get to the point where they're like, "All right, very little of the API request is going to change on a day-to-day basis." For some of our jobs, all that changes is just sort of two dates, just like I just want to retrain this model on this new set of dates today. All the other things are locked down.

That's kind of how model exploration works in Railyard right now. One thing that's been on our list to work on is like making notebook integration for that. Our notebook integration is not very good right now. But we've seen people, especially folks coming into the company, they come in and expecting to be able to use Jupyter Notebooks and do sort of like notebook-based data discovery and exploration, and we're trying to make that easier too.

[00:33:25] KR: Yeah. I think Railyard has really focused on the productionization and automation side and also on sort of the reproducibility aspect, like some of our machine learning is pretty high-stakes. You don't want to like just start declining all the charges. So just kind of making that really robust. As Rob said, I think we have some work to do still on making exploration like really well integrated with that.

I think one thing that's also been cool, which is not part of Railyard itself, but built on top of it, is the idea of like auto retraining. Then one thing that kind of forces you to do if you want to automatically retrain models and decide, "Is the new one better than the old one?" One thing you have to do is kind of like codify your model evaluation logic, because there has to be a way a computer can decide, "Is this one better than that one?"

I think that kind of really drives you away from like, "Let me stare at this AUC curve and kind of think about it," into like, "Okay, my algorithm is like I want my false positive rate to be this, and after that, maximize recall," and figuring out also – In some cases, we kind of compose models. Try and run this one, but fallback to that one.

[00:34:31] JM: All the stuff in the API.

[00:34:32] KR: I think that Railyard has been like a really good force and kind of having this automatic retraining on top of it to have a sort of like codify our evaluation metrics, because you kind of have to do that if you want to automate all of these stuff. Now we have kind of like a little bit of an evaluation API that we've built as well that Rob could probably say more about too.

[00:34:50] JM: So when I think of an API, I think of something that is kind of imperative interface. I'm executing an API. Right now, I'm making this API request. Computer, go do something. The Kubernetes way of things is more of a declarative, like this should be the state of the world at all times. Tell me your perspective on declarative versus imperative. I realize it's a sliding scale that's kind of semantic.

I mean, you could have made, you could have done this declaratively. You could have said like, “Railyard is a declarative way of describing when and how you want to run your models.” Instead you said, “Here is an API. You make requests to it.” Was that deliberate or like tell me about declarative versus imperative in your mind.

[00:35:40] RS: It was pretty deliberate and it was sort of very driven by the product teams and that they were very interested in calling this API, right? They wanted this API to exist because they wanted to build sort of the idea of we want to retrain these models and evaluate them and then decide whether or not to put them into production. They wanted to own that logic, because that is sort of their logic to own as a product team. They’re the ones making the decisions about, “Is this model better than the last one? Should this model go into production?” That sort of thing. So, from their perspective, what they really wanted was that API that they could call, like sort of dynamic API.

The declarative piece, it’s an interesting question. So, on Kubernetes, we use like the Kubernetes jobs interface, and there are pieces of that that are declarative. We sort of have this pile of YAML that describes how we want to run the job. So, to run a job on Kubernetes, you have this YAML description of, “I want to run this job with these resources, with this Docker file.” So that is very declarative. We build up one of these declarative specifications when we run the job.

Within that job, within that Docker container, we’re executing a bunch of imperative Python code. So, it’s sort of these pieces all working together. So we make an API call, so we make sort of this dynamic HTTP API call with a JSON specification for how we want to train a job. We build this declarative spec to tell Kubernetes, “Run this job with this Docker file,” and then we pass through this JSON specification to Python code, which imperatively decides how to train the model and what data to fetch based on this specification.

So it’s sort of like a nice blend of the two. We’re sort of using a little bit of declarative pieces in building our Kubernetes job and sort of specifying the Docker container, but we also have this dynamic piece, this API that our product teams are using to sort of train models on their terms effectively.

[00:37:36] JM: Taking a step back, how did you come upon the insight of giving this unified experience through an API? Because it's not necessarily like this is not – I just want to understand your thought process a little bit better, because this is not necessarily like A leads to B, leads to C kind of insight. This is an insight of there's all of these disconnected stuff that's going on in the machine learning developer's workflow. Let's unify it into an API that is – By the way, I was looking at a request for like Railyard. It's a lot of stuff. You have to put in a lot of different parameters because the machine learning job is very complicated, but at the same time it's very nice to have all these stuff in one place. What led to that insight?

[00:38:26] RS: I think we struggled in the beginning. I don't know if struggle is the right word. We just had a lot of hard decisions to make about sort of how –

[00:38:33] JM: So you got so many constituents. There are so many different people that you're trying to cater to with this API.

[00:38:38] RS: Yeah. There are a lot of constituents, but I think one thing that we did right and one that we had to make a decision on early was sort of how much freedom to give users in terms of the code that was being executed. So, I mentioned in the blog post that one thing we could have done is when we wrote Railyard, we were essentially mostly just using scikit-learn as a machine learning platform. I don't think – We did not have [inaudible 00:39:04] boost as a framework to that point. We're just using scikit-learn.

One thing that we could have done and one thing that we talked about, like seriously considered doing, was just building a DSL for scikit-learn, where people would specify not just how we want to fetch data and how we want to hold out that data and how we want to filter that data. Literally, how we want to construct the rest of our ML pipeline like each component, like how to transform it, how to encode it, what model or estimate do we want it for or wanted to use.

We considered that. We thought maybe we could build a DSL where folks would literally write no Python. The only thing they would have to write was this declarative JSON specification of how they wanted to build this machine learning job end-to-end so they would – All they would have to do is write some JSON and that would contain everything that they needed to run their ML job.

I'm really, really happy we decided not to do that. We sort of hit a middle ground. There are some things that every machine learning job needs to do. They all need to fetch data and they all need to run that data through some sort of funnel, right? Then at the end they want to be able to serialize their model and they want to be able to serialize some sort of evaluation data.

Basically, every machine learning job has a handful of things. No matter what framework they use or what problem they're trying to solve, they almost all kind of have a few things that they have to do. So, the Railyard API really tries to address kind of those core fundamental properties, those properties that are fundamental to every ML job, and then kind of leave the rest up to the user.

So, I sort of jokingly referred to Railyard as like an arbitrary Python execution service with opinion, sort of opinionated interfaces of the boundaries. That's kind of what it is. When you write a Railyard workflow, we have these functions, we have a method like train. We say, "We're going to call the train method with your data that you have to implement." At the very least we say, "You need to tell us how to fetch your data, and then you need to write some Python code that trains your model and pass your model back out."

But, the product person is writing this Python, and they can do whatever they want in that train method, right? The only contract is we're going to pass the data in and you need to pass the trained model back out. Within the train method, you can write whatever Python you want. We've also allowed users to sort of override how they fetch data. They can customize that. I think the flexibility in terms of letting our customers kind of write the machine learning workflows that they need to write in Python and us really just defining the interfaces is part of the reason that we've got pretty decent adaption with Railyard, because we have a constrained kind of like fundamentally what they're wanting to do. We haven't constrained them to just use one framework or just write Python in the way that we want them to. They have a lot of freedom to build these training workflows. We just sort of define the interfaces for them.

[SPONSOR MESSAGE]

[00:42:06] JM: DigitalOcean is a simple, developer friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow. DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support, plus they've got 2,000+ tutorials to help you stay up-to-date with the latest open source software and languages and frameworks. You can get started on DigitalOcean for free at do.co/sedaily.

One thing that makes DigitalOcean special is they're really interested in long-term developer productivity, and I remember one particular example of this when I found a tutorial in DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence, and an attention to just getting developers off the ground faster, and they've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's the D-O.C-O/sedaily. You will get started for free with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:44:07] JM: Let's make this more concrete. So, let's say we're talking about the problem of fraud detection, detecting fraudulent transactions. I want to train a model and deploy a model for fraudulent transactions. I want to use the Railyard API to get this thing going. What is my request look like? What happens when I submit that request? How does Railyard parse it? How does it create the execution logic? How does it spin up resources? Take me through the lifecycle of a Railyard use case.

[00:44:45] RS: So, let's start – Every Railyard job starts with just a blog of JSON, right? Some JSON describing how you want to run this job. So, at the start, we'll start with describing where you want to fetch your data from. One option is we use Red Shift. You can fetch your data for Red Shift. Most of the production jobs have ended up with their data as parquet files in S3.

So, to start you'll say, "Okay, I want to fetch these 10 columns from this parquet file at this S3 path." So, there's the start as we say. This is concretely the data that I want to use to train my model. All sort of the upstream data discovery and data generation has happened outside of Railyard. You're just telling Railyard, "My data is ready. Please go get this data. Get these columns from this data." So, that's sort of the first part of the API request.

The second part is this is how I want to filter my data. So, if you're doing retraining, it's possible that you have data. Let's say we have data that represents the entire history of transactions at Stripe. Well, it's probably that you don't want to train on the entire history of transactions at Stripe. Maybe you just want to try it on the last three months. Maybe you want to try it on the last six months. Maybe you want to try it on the last year, but you probably do want to filter that data in some way.

So, that's the next part of the request, is here's how I want to filer my data. You'll see for a lot of the retraining jobs that a lot of times they're just manipulating those filters. They're like, "We want to train the same model just with a slightly different dataset, a slightly different filter set."

The next part of the request is here's how I want to hold out my test data. With machine learning, you sort of have a dataset that you train on and then you have your hold out set. So you want to actually test your model against this set, update and then it's never seen before. So, this is another piece of the Railyard API specification. You say, "This is how I want to hold out my data.

One way to do that is to just carve out a chunk of data. So you say, "All right. I want to train – Given the last four months of transactions, I want to train on the first three of those four months, and then the last month up until now, I want to test on that." That's an example. We also support sort of the out of the box scikit-learn ways of splitting data. You can split data based on like a key – You can sort of do random sampling of data, that sort of thing. So, that's like the core data request. Where is my data coming from? How do I want to filter it and how do I want to build the hold out in our test set?

The next piece is actually pretty small and straightforward, which is just what is the workflow? What is the name of the Python code that I want to execute? We call it a workflow. Workflow is kind of like an overused term in a lot of different Python applications. Airflow also calls the things that it does, workflows, or Airflow workflows as well.

But we have a Railyard workflow. That is essentially Python code that I want to execute. In the request we say, "All right, here's the Python code that I want to execute." There are a few other fields, but that's effectively that meat of it. It's like, "Here's a Python code I want to execute with this data."

So, the next thing is an engineer or a service or a computer will send this API request to the Railyard service. The Railyard service itself is written in Scala. It's a finagle API. Finagle as an HTTP server written by Twitter. The Scale service itself is a pretty thin layer actually. So, the Scala service validates the JSON request that says, "This thing is shaped correctly." It handles sort of all of the job execution and monitoring the job. So it will get the request. It will take this request and then package it up for Kubernetes.

So, the Railyard itself, we run a PostgreS instance. So we'll get the request and we'll test PostgreS, "Hey, we got a request. Here's when we got it. Here's when we're starting the job. The job is running." So PostgreS is where we keep this idea of a job running or complete or the job has been killed or something like that or deleted.

So the Railyard service itself is sort of Scala talking to PostgreS to keep this metadata about job execution. Then we'll actually use the Kubernetes Java API itself to build the request, to build this Kubernetes job request. So, we'll take the API JSON, we'll build this Kubernetes job request and then we'll make a request directly to the Kubernetes API and say, "Start this job with this API request in this Docker container and these resources." So that request goes to Kubernetes.

Kubernetes schedules it, right? So, one thing that Railyard does is it tells Kubernetes, "We have sort of different instance types, so we can trade models and sort of instances that are optimized for CPU. We can train models on instances that are optimized with a ton of memory. We have GPU instances for deep learning models, that sort of thing."

So, Kubernetes actually schedules this job. So, Railyard makes the job request to Kubernetes. Kubernetes says, “Okay. Railyard has to me it wants to run a job on this instance type. Do I have enough resources on one of this instance to run the job?” If it does, great. It will start up the job. It will execute the Docker container. If not, it will basically sit there in queue and wait for services to become available. So let’s assume that the job that we have resources available and we started the job.

So, we’ll fetch the Docker container. We’ll basically just execute Python. Python with a few command line arguments, including the API request itself and things like identifier, like a job identifier, that sort of thing. That basically calls in to our Python host. That calls in to our Python libraries. At that point, we will take that JSON request and say, “Okay, they’ve said I want to fetch data from here. I want to filter it like this and I want to hold it out like this,” and that’s when we start executing our Python code, where we use our Python libraries to say, “Okay, go read this parquet file out of S3. Built it into a pandas data frame,” which we’ve sort of standardized on pandas data frames and numpy arrays for the most part, but we’ll build a pandas framework. We fetch the data. We’ll carve out the hold out or test set and then we will pass it into this train method on your workflow. That’s sort of there the ML engineer’s code takes over.

So, up until this point, everything that Railyard has done, the ML engineer has written this JSON specification, sent off a request, and everything up until now has been controlled by Railyard. At this point, we call into the train method on the workflow with their hold out and their test data frame and then it’s their code running, right?

So, they build a scikit-learn pipeline, or they train an [inaudible 00:50:53] boost model and they train a fast text model or they build a PyTorch deep learning model or something like that. So they train their model and then they pass their model back out. When they pass the model back out, we have to do a few things.

So, we need to save their data. So we need to save their labels and their hold out data for evaluation. We need to take their model and we need to serialize it. So we need to serialize it in a format that we can score it, that works, that we’re able to score it in production. That is something that Railyard announced, like that our ML engineer shouldn’t have to think about at all. Serialization should be up to us.

I think one kind of the beautiful things about our platform is that when you train a model with Railyard, you can – As soon as the Railyard job is done, you can score that model in production like immediately. When your Railyard job is finished, you can immediately make an HTTP call and score your model. The moment after it's done. It might take a few seconds to load the model into memory.

So that is being served through a Scala service that we cal Diorama. So we have a scoring service where we have Scala implementations of our machine learning models. We have Scale implementations of Random Forest and [inaudible 00:52:01] and that sort of thing.

Yeah, that's sort of the end-to-end. So at the very end, we'll also report back to the Railyard service and say, "Hey, our job is done. Here's our model identifier. Here's where all of your evaluation data is. The Railyard service will tell PostgreS, "All right. Set this town to complete," and then someone can query the Railyard API and get back their model ID and then you pass to their evaluation data and that sort of thing. That's sort of the lifecycle of a Railyard job.

[00:52:28] JM: It's quite a detailed system, and one thing I like about it is kind of the boldness of having to outlay so much work, it seems, before you really have – I mean, is this something you can build a proof of concept for or is it something where you really just have to kind of like get everything done and then like let people do it. Can you give people like – Maybe you give them the API service and then like you go and like wire everything together in an ad-hoc fashion? I guess there are some ways of prototyping this kind of thing.

[00:53:03] RS: Yeah, it's a good question. There's kind of this – Sort of this cliché in tech that you want to do is like build the skateboard, and then the bicycle, and then the car. But I think in practice it's often very hard to do that. I think in this case, we did a pretty okay job.

So, we kind of started with a Python codebase. So when I talked about ML engineers kind of logging, SSH-ing into an EC2 instance and training a model, we had something that looked a lot like a Railyard workflow as a Python library already. A lot of these Python libraries were already in place. They were just executed by people from the command line.

So, we sort of had to structure those. Rather than having a workflow, there was a bunch of sort of imperative Python. We had to make sure that we could build that workflow from some JSON. But a lot of those Python pieces were in place. There were some work to sort of refactor them into something we could execute based on like a declarative JSON specification, but it wasn't too bad. So, a lot of those things are already been written.

Then the next thing we did is like – Actually, the very first version of Railyard didn't interact with Kubernetes at all. The very first version of Railyard had – So I was talking about the Scala service that is the HTTP API and talks to PostgreS. The very first version of Railyard was just that Scala service running on an EC2 instance using Java's process manager to kick off a Python process that would run these training jobs. We ran like that for months.

The very first version of Railyard is just this Scala service using Java's process management to run Python processes and get results that way. So, there was no Kubernetes component for quite a while. That was sort of us walking. Then sort of at the end of like this first big project sprint, we finally were able to get to Kubernetes integration and move off of sort of what we call the in-memory runner, the in-memory runner being Java kicking off Python processes.

But it lead us shift the first version of Railyard months before we would have otherwise and sort of let us validate that it was going to be a useful thing for folks and then it was going to be sort of – And let us sort of iterate on it early on without having the complexity of Kubernetes.

Now, the downside for this was this first version, the in-memory version, we're sort of still running on EC2 instances where if we needed more memory, then we had to spin up new instances. We didn't have the flexibility. We didn't have like the scheduling capability of Kubernetes. We didn't have the ability to run on different instance types. There are a number of things that Kubernetes gives us now that are really nice to have. But, yeah, that first version was just a Scala service that was like kicking off Python processes in-memory right there on the same [inaudible 00:55:51] the server is running on.

[00:55:53] JM: We got to begin wrapping up. I've definitely only scratched the surface of the questions that I wanted to ask. As we conclude, I want to get both of your perspectives on the industry trends that are exemplified by this work that you've done. So there's a bunch of different burgeoning areas that we've touched on from kind of the exploratory versus deployment fashion, like the exploratory Jupyter Notebook side of thing we didn't really get into versus the deployment frictions versus your reframing of the problem into giving an API surface to something that was a bunch of disconnected parts. There are so many different areas that we didn't really dive into in as much detail as I would like.

I guess, Kelly, from your side of things, I'm hoping to get a perspective on data engineering management and the roles of different people in an organization. How they work together? To wrap up, I'd like to get Rob's perspective on the individual data engineering practioner and how things are evolving there. So, Kelly, why don't you go first?

[00:57:07] KR: So, you're kind of asking about just different roles on how people interact.

[00:57:10] JM: The state of the world, not just at Stripe, where you've got a high-functioning organization with a bunch of unicorns speaking more generally about the industry. Thinking about your random software engineering shop that does machine learning work in the Midwest somewhere. Just a set of tools that are available, industry trends. What is changing in the life of teams?

[00:57:39] KR: I think there are a few things that are happening. One is just in terms of like which tools and frameworks? There's sort of like this massive explosion of things that people can try, and I think that's both kind of like libraries as well as sort of more like managed products. I think there are a lot of different things people can explore.

I think you can divide a little bit into like kind of people, like you're talking about someone in the Midwest who wants to do a little bit machine learning versus like kind of large scale production deployments. I think, actually, the managed space is pretty exciting for the first group where you have things like SageMaker on AWS and just like a lot of ways that someone can get started and get something working a lot faster than they used to be able to without having to know kind of all these many layers through like the application and machine learning all the way through machine learning infrastructure all the way down to compute. I think that's actually like pretty cool.

I do think you have the side that's more like really scalable, customizable, for companies that really focus on have teams on this where different open source projects that people are like delving more into. I think we're a little bit more on that side where we want a lot of control over how we do things and we care a lot about the scalability. So we're kind of –

[00:59:00] JM: Right. SageMaker wouldn't work so well for the nuanced that you want to bring to your infrastructure.

[00:59:05] KR: Yeah, although I think if it were like 5 years ago and we were just starting out, maybe that would be kind of useful for us. But I think we've kind of passed through a little bit the point and we've built so much of our own already that's sort of more customizable and tailored to our needs. But we are kind of looking at like what are all the frameworks. We have people now using fastText. We have people using PyTorch. We have people using PROPhet. So we're kind of looking at like what are all the capabilities that we want to integrate to our platform versus like here's a managed platform that gives you something like pretty good for like not much.

[00:59:35] JM: Very interesting. Okay. So, Rob, as somebody who's building these kinds of tools that Kelly alluded to. You've got your hands deep in the much. Any predications of perspectives on how the building of machine learning infrastructure will change in the near future?

[00:59:56] RS: I think that Kelly – To think about 5 years ago, I have a machine learning model. How do I deploy it? How do I actually score this thing in production? The options were pretty limited. When you're talking about SageMaker, we realized we used to like everything about SageMaker kind of wondering like, "Why didn't we use SageMaker?" We started the Railyard project six months before SageMaker was announced.

So, a lot of these tools, like if you look at sort of Google and Microsoft's Azure and AWS offerings now, they're really quite good. I think for a lot of teams, they're probably going to end up sort of using one of those solutions for serving models. You can also build this yourself. You can stand up a Python service and train a Python model and sort of serialize your model using the built-in scikit-learn stuff. That's an option for a lot of companies with big enough team.

I think the hard part is for a small company, we're lucky that we have a machine learning infrastructure team. We have a team that can build these things. For a lot of companies, the data engineer and also the data scientist and also the machine learning infrastructure person and also sometimes like the product application person and the analytics person. I think it's a very common case where the first data science hire in a given company has to do all of these jobs at once.

I think for them, these managed services I think are going to be a huge, huge help, because it just takes away some of the things that they have to think about. So in the same way that we built Railyard and our machine learning infrastructure's internal at Stripe to help machine learning engineers and data scientists to think less about these things. I think sort of the managed services are helping the small companies think less about them too.

I think that it's interesting, TensorFlow has a really good serving solution now. I think we're going to see more things like this. There's been pretty interesting research out of Berkeley talking about machine learning serving systems, that sort of thing.

If you're starting a company and you're like, "All right, I want to pick a database and sort of like a message queue." You kind of have these big, great open source solutions. You could choose Mongo, or PostgreS or one of these open source database systems. You can say I want to use Kafka as a message broker, which is like a wonderful, huge open source project.

But as far training models, there are a bunch of great training frameworks. It's very straightforward to get started with scikit-learn and get started with PyTorch. But in terms of actually like serving those models in production, that's one area where there hasn't been a ton of open source work yet, and I'm expecting to see more of it.

I think like TensorFlow serving and Kubeflow and that sort of thing. It's like steps in that direction, and I expect to see more things like that in the future sort of more out of the box open source solutions for serving models in production. That's sort of where I see things headed.

[01:02:45] KR: Well, and I think one of the things that kind of circling back to what we talked about at the beginning, like machine learning infrastructure is sort of challenging and interesting partly because it's sort of like this integration over everything else your company does where you're like using all these data and all the other infrastructure, and that makes it like a little bit harder to kind of standardize across companies.

I think in the managed space, you'll have people where just those components become more standard. So everyone actually is using the same set of things and they can build on that, but then for larger companies, you kind of have to figure out like how do you glue together all of these different components that you may want to use and make it kind of like cohesive and work together.

When we talk to other ML infrastructure teams, I think some things that's like everyone has their own version of this thing, but it's a little bit hard for you all to use exactly the same framework, because you're building on like slightly different infrastructure underneath.

[01:03:35] JM: There's certainly not a historical example that's easy to map to. If you think about like Netflix being the first big company to move to the cloud and adapting EC2 servers, the abstraction of an EC2 server, not very complicated. It's a virtual machine. Everybody knows that's what they want. With machine learning infrastructure it's like a lot of different people want a lot of different things. There's a lot of subjectivity. Hard to know what the end state is going to be if we talk about something like a standardized set of things, even if you compare it to something like the container orchestration space. In the container orchestration space, you had Mesos and Docker Swarm and Kubernetes and they were doing stuff that looked pretty similar. The API surfaces were kind of different.

When I look across the machine learning space, it seems like there's way more variability than there was in the early days of container orchestration. I just bring that up not to open a can of worms but just to say that it's so hard to predict and it's very hard to draw any kind of historical analogy to the way that frameworks have developed in the past.

[01:04:48] RS: Yeah. I think like a really good example of this is if you look at the machine learning frameworks themselves, especially in Python, they've pretty much standardized to kind of numpy-like arrays or pandas data frames, things that look like matrixes.

TensorFlow has its own Tensor interface. PyTorch has its own Tensor interface. There's a numpy interface, but they all look pretty similar. If you look at PyTorch's Tensor interface, they sort of follow what numpy does quite closely.

The real problem has been given that you have a trained model, is there a common format that you can serialize that model to that something downstream could read? So, model serialization? How do you serialize the model?

[01:05:35] JM: Sorry. What does that mean? Why is model serialization important?

[01:05:39] JM: So, if you want to be able to serve the model in production, you need to be able to save model to a format that you can reload somewhere. So the example of like a simple Python services, if you train a model in Python, you want to be able to save that model to some sort of serialization format and then reload it into scikit-learn as an example. You want to have scikit-learn be able to save it to a file and then reload it from a file.

Scikit-learn's solution for that traditionally has been Python's pickle module. Pickling Python classes. There's never been – So we talked a little bit earlier about sort of having a declarative specification for something. There's never been until recently a great declarative specification for machine learning models. Now, TensorFlow did a great job of this out of the gate. So you could take any TensorFlow model and you can safe it to protobuf.

So the protobuf is like a declarative specification of that deep learning graph effectively. The rest of the industry outside of TensorFlow has never really had a standardized format until about 8 or 9 months ago when Facebook and Amazon and Microsoft and a bunch of other companies got together and have been working on this format called ONNX, which has been in my mind the most successful attempt to thus far at having a standard format for serializing ML models. Pretty focused on deep learning models, but also has hooks to serialize our normal model types.

So, at Stripe, we have in the past – We've rolled our own serialization. We have our own custom JSON serializers for models. We think we're probably going to start moving towards ONNX as like a standardized serialization format. Microsoft has been doing a ton of work.

Talking about the scoring side, Microsoft has opened sourced sort of a scoring service for ONNX. So I think if we can see a model serialization format that has sort of a declarative specification for here's an ML model and then can anybody can implement against specification. Given this declarative specification of a model, you should have the same scores out of that model no matter – Modulo some floating point around the air. You should have really similar scores out of any service that scores that model, right? I think that if ONNX get traction, if like a standardized model serialization format can get traction, that's what's going to unlock more open source solutions for model scoring.

[01:08:01] JM: All right. I got to do a show on ONNX. Rob and Kelly, thank you for all your time. Really interesting subjects.

[01:08:07] KR: Thank you.

[01:08:09] RS: Yup, thanks a lot.

[END OF INTERVIEW]

[01:08:13] JM: FindCollabs is a place to find collaborators and build projects. The internet is a great place for communicating with each other. Why aren't we building projects together? Why is it so hard to find other people that will work with you on a new project reliably? Why is it so hard to find cofounders to start a small business with?

We started FindCollabs to solve these problems. I love starting new projects, software platforms, musical songs, visual art, podcasts, and when I create these projects, I want to share them with people. I want people to be able to follow my progress, because as they see that I'm dedicated to consistently delivering quality work, maybe they'll be inspired to join me, and it doesn't have to be on a financial basis. It could be just to collaborate and to make artistic progress.

You can see my profile on findcollabs.com by searching for Jeff Mayerson. You can see the different projects I've made; games, podcasts, open source software. Some of these projects are incomplete. Some of them are well-developed, and this is how innovation and invention happens. Projects start out in an incomplete state, and if you can prove to the world that you work on projects with dedication and consistency, you will get other people to join you. Whether you want to hire them or if you just want to collaborate casually.

You can see my profile and many other people's profiles, projects that they're posting on FindCollabs, and I would love to see your projects there too. You can check it out by going to findcollabs.com, F-I-N-D C-O-L-L-A-B-S.com. You can post a project that you're working on no matter how incomplete it is.

The best projects in the world start with an inspiring vision, whether or not they have code attached. If you like to compete, the FindCollabs Open is for you. The FindCollabs Open is a $2,500 hackathon with prizes for the best React JS project, the best machine learning project, the best game. There are lots of prizes, and you can check it out by going to findcollabs.com/open and post your project.

Thanks for being a listener to Software Engineering Daily and I hope you post your cool ideas on findcollabs.com.

[END]