EPISODE 1120

[INTRODUCTION]

[00:00:00] JM: Code is version control through Git, the version control system originally built to manage the Linux codebase. For decades, software has been developed using Git for version control. More recently, data engineering has become an unavoidable facet of software development. It's reasonable to ask, "Why are we not version controlling our data?"

Dmitry Petrov is the founder of Iterative.ai, a company for collaborating and version controlling datasets. Dmitry joins the show to talk about how version control works with data and his company Iterative, which builds products around the version control and management of datasets.

[SPONSOR MESSAGE]

[00:00:45] JM: Code doesn't always behave as you expect, and your team might be wasting time trying to understand why. You can empower your engineers with Rookout. Get the data that you need from live systems instantly and start shipping software better, faster and more reliably. You can visit rookout.com/sedaily today. That's rookout.com/se daily.

[INTERVIEW]

[00:01:16] JM: Dmitry, welcome to the show.

[00:01:17] DP: Thank you for having me.

[00:01:19] JM: There is a lot of tooling machine learning. There's SageMaker. Lots of other cloud tools, Tensorflow, Frameworks. Shouldn't the problems around machine learning already be solved? What are the gaps in tooling in machine learning?

[00:01:37] DP: This is a really great question. From one's idea, there are tons of different types of tools, right? For modeling different types of algorithms, different types of libraries,

implementation tools, like [inaudible 00:01:54], right? Let's not talk about GPT-3 today that was released one week ago, and today everyone is talking about it.

Anyway, a lot of tools around. But when you take a look at the work flow, at the process, how the ML process is organized, you will see that there are not much tooling around. Most of the tools focused on individual data center's experience. It's about creating monoliths about watching those experiments. But when it comes down to collaboration, how to transfer my result to my colleagues. How to capture my experiments and share this with my team? How to move my result to some production environment? Then there are not much tools around collaboration.

[00:02:38] JM: So, collaboration.

[00:02:40] DP: Yeah. The area that we are working on, it's [inaudible 00:02:44] to workflow, the lifecycle management around machine learning process. And we manage – Our tools manage data, ML artifacts, ML pipelines, ML models mostly with the focus on collaboration on lifecycle management around the models and datasets and all the artifacts, all the activities around the ML workflow.

[00:03:08] JM: What kinds of datasets need to be shared or collaborated on? At what points in the workflow? Are we talking about sample datasets or training datasets or production datasets? What are we talking about?

[00:03:23] DP: We are talking about any dataset that you use for your ML experiments. Any datasets that you use for your analytical projects. At the end of the day – When you do machine learning, at the end of the day, you need to upload all the data in a single machine. It might be the cloud instance. It might be your laptop. But you need all the data here, because most of the algorithms work on with – In a local environment.

This is a place when tools like DVC can capture your dataset. So when the result is produced, you will know what version of the dataset was used. How exactly to reproduce your result? And what is probably even more important to your team mate don't know how it happens and how to get the same result.

[00:04:13] JM: When I run a dataset through a machine learning model, I might want to share that exact dataset with somebody else to run a different model or would I want to receive a different dataset to run on my model? Give me a little bit more about the mechanics of when datasets need to be shared and what models they're being used on.

[00:04:37] DP: Okay. First of all, we can separate like lifecycle and team collaboration and your personal experience as a data scientist or ML engineer. Actually both of kind of users, types of users, need to know like what exactly was used. Because in two weeks, when your model is ready to push to some like production system or to your teammates, they can ask you a question like, "What exactly the version of dataset was used? Did you use dataset from like previous week, or month ago? Or did you use like cleans version of the dataset or a raw dataset?" All these question arise and you need to have like absolutely clear answer. This is when you need versioning for dataset and this is when you need to kind of tie the dataset versioning with your code versioning.

[00:05:32] JM: So data versioning should be tied with code versioning. Say more about that?

[00:05:38] DP: Yes, absolutely. You need to version your code, and this was kind of a solved problem in the industry, right? But the version control systems cannot version datasets. And this is what we solve. What our tool, DVC, solves. We tie it to your datasets or with your code together.

[00:05:58] JM: Can you draw an analogy between a regular Git workflow and a workflow that has DVC and Git?

[00:06:07] DP: Yes. When you use DVC, you have a bunch of commands, which are kind of similar to Git when you say like DVC add dataset, or file, or DVC add a model. Then you have a commands tool defined to your pipeline when you say DVC run and then command, which like connects your data with your model and it might form kind of complicated pipeline or direct acyclic graph. And then this is how DVC looks from our side from user point of view. You just run commands and version commands like Git does.

However, this is how I explain DVC for end users, for data scientists, for ML engineers. However, for people with engineering background, with DevOps background, I explain DVC at a different way. I am saying that DVC codifies your data. And after the codification, Git does the actual versioning.

[00:07:13] JM: What does it mean to have my data be codified?

[00:07:17] DP: Codified means – This is a general practice. Like how people work with data today. You cannot version datasets or model or ML models in the repository, right? In your Git repository, or any version control repository. And what people do, they just create meta files with description. To run this model, I need to take data from this, let's say, S3 bucket, or like Azure directory, or from these like FTPC, or from this particular directory. And they keep this meta information in their repository. They version these meta files together with code. This is how kind of connection works between data files or actual data and your code. DVC just streamlined this practice. It has a set of kind of conventions around like how exactly this meta information should look like. How to version your data from your S3 bucket, like Google storages or random servers. It defines the rules how you version pipelines, how to create meta information about pipelines. At the end of the day, you have like a few files or maybe like one single meta file. And then you just say Git add these files, commit push, and all the information, all the meta information goes to your regular Git repository. In this sense, we are not reinventing Git. We're just like using Git as an underlying technology.

[00:08:49] JM: So the actual usage of the dataset versioning control system, I'd like to understand that in better detail. So let's say I've got a machine learning team. We ship our models to production. We ship a new model to production – I don't know, every week, or three times a week, five times a week. How is the data version control system being used in that context?

[00:09:14] DP: Yeah. The environment you described, it's a good environment, like production one, right? And when you ship model in the regular ways, you probably have – It probably is not a manual process, right? You might run some pipelining tools. You might run some data engineering tools. And data engineering tools can use DVC inside, can use DVC to ship this model, to keep track of your dataset and source code.

In our experience, we focus mostly on the development stage. When users or data scientists version dataset together with code and he or she just file this result to production team. And production team has a clear protocol how to deal with the data. How to get model using regular kind of Git flow, right? Your Git repository becomes description of your model. And from production environment engineering can say, “So, I have this repository. There is a model file. And I need the latest version. I need master head from this version and so on.” The dev environment, you get the latest version. Or you can say from production environment, “I need a version, a number with this stack or with this like Git hash.” And you’ll get version, like previous version, or version whatever you ask. With DVC, you introduce Gitflow in your model management, in your dataset management.

[00:11:00] JM: And can you give me maybe a concrete example? Like are there any companies that you’ve worked with or applications that you’ve seen where you can describe why this is useful or what kinds of problems it solves?

[00:11:14] DP: Yeah. First of all, DVC basically mainstreams the general practices of introduce meta information about your datasets, right? And you, anyway, need to version your datasets, your models, and DVC gives you like a way of doing this.

What is special about DVC, and I believe this is a core for the adaption of the DVC. We don’t introduce any new infrastructure. DVC works on top of your Git, which means you use your Git – For example, GitHub server, or GitLab server. DVC uses your storages, like S3 buckets, or FTP server with a particular address. And you don’t need any additional service or database or whatever. So it’s like a distributed by nature because it uses Git under the hood.

I believe this is why many people prefer to use DVC rather than go to some – I don’t know. End-to-end ML platform, which has all the pieces and you have additional technology stack. Additional technology stack in addition to the software engineer stack. Our core belief is AI stack, AI platforms needs to be built on top of the existing software engineering stack. You just need to extend the stack. You just need to enable the data scenarios, machine learning scenarios.

[00:12:49] JM: That's a profound realization. And can you compare that? Your fully integrated approach to the other approaches out there? Because I mean version control of data management for machine learning gets accomplished in a variety of other ways. Can you compare it to some of the other workflows that people endure?

[00:13:08] DP: Usually, data versioning is happening under the hood of your AI platforms. If you use an AI platform, and many companies today have built internal AI platform, or you can buy an AI platform. They do have data versioning. You cannot use a proper workflow, proper collaboration without data versioning or model versioning. And this is hidden in the platforms. And you as a user usually use the versioning and all these capabilities through like WebUI like interfaces.

In our case, DVC basically say that you don't need an AI platform. You have your Git for versioning, which is great, and everyone use it. You have your S3 buckets or whatever storage you use. And DVC just orchestrates those existing technologies and introduces data versioning to your workflow. This is how DVC is different from AI platform point of view.

From another point of view, you can take a look at data engineering systems. And some of the data engineering systems also have versioning inside, right? One, for example, Pachyderm, is data engineering system, but it still have capacity. Not capacity, but functionality of versioning data. But DVC is not data engineering tool. It doesn't have like any like schedulers. It does not recover your jobs. And it source like a very simple goal. Just connect your code in data and ML pipelines altogether and version and transfer all over these artifacts together.

We focus on data science workflow when data science work on some projects and collaborate with the teammates. And data engineering goes, like in the next step, when you do productionization of your models through pipelines through batch processing.

[SPONSOR MESSAGE]

[00:15:15] JM: Today's show is sponsored by StrongDM. Managing your remote team as they work from home can be difficult. You might be managing a gazillion SSH keys and database passwords and Kubernetes certs. So meet StrongDM. Manage and audit access to servers,

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

[INTERVIEW CONTINUED]

[00:16:44] JM: We've talked a little bit about data versioning at this point. Can you talk about model versioning? When is it important to do model versioning and how is that implemented?

[00:16:55] DP: Model versioning of course is like a core piece in the data science workflow, because usually this is the connection between your development part of the cycle between data scientist and engineers, production environments. This is how these worlds are connected. Of course, you need to have a kind of conventions of some product or how to work, how a team can work together. From DVC point of view, it just versions files. It can be models. It can be dataset. It can be like intermediate result. For example, some match exist that it will be used in the training.

For DVC, it doesn't not matter like what you version. It can version models as well as datasets. What is important, DVC has a concept of metrics, right? This is a core part of ML workflow. And when data scientist think about like models, they don't think about the files, right? They think about metrics. What metrics I got in the modeling process. And metrics is also the artifact that DVC versions and tied together with code and models.

Second thing is hyperparameters, because in many cases, this is what's driving your success in modeling. This is also important to remember like what types of parameters, hyperparameters

you use. This is how DVC came from managed models, not just like a dataset. You've got a file, which DVC can version in a problem and corresponded metrics as well as hyperparameters.

[00:18:41] JM: So tell me a little bit about the engineering. What have you had to build in order to enable dataset versioning and model versioning?

[00:18:53] DP: Yeah, this is an interesting question. In terms of engineering, I think the core part of the engineering, what the decision around if we should do like AI platform style tool? Like create a server and put meta information into the server and maybe even data, or we should do just the kind of lightweight way with codification.

And what we found is that the codification can work when we can just codify all the artifacts and put them into your regular. It was an interesting architecture decision. And this decision put us in a kind of like interesting spot where you don't like any additional infrastructure in your team. You can version data. You can version your models, everything around just using your regular workflow.

What we have built, just orchestrator. DVC is kind of an orchestrator around your Git and around your S3 storages. This is how it works. And it is like distributed by the nature just because it uses Git as an underlying technology, because Git is distributed, and so DVCs.

[00:20:15] JM: Now, are there any places where Git is not an ideal underlying medium or ways in which you paper-over some of the features of Git or build on top of it and extend Git? Tell me what about Git as a version control system is not appropriate for a data version control system.

[00:20:39] DP: Oh, yeah. This is really, really a good question. Yes. First of all, of course, Git has a lot of limitations and assumptions around the workflow. And initially it was not designed for data science workflow flow, like data specifically. When it comes down to data management and data transferring from like your environment to your instance in cloud, it works just great. You clone your repository. You pull code, you pull data, and everything works well. The problem is experiment management.

You're probably familiar with data science workflow? They run like tons of different experiments. Sometimes you can have like ten experiments per day, sometimes 20, sometimes 3,000. There are ways to automate this experimentation process.

Today, DVC is based on semantic of commits, right? Every experiment is a commit. It can be in a branch or it can be in your like existing branch or a new branch, whatever. But making the commit, it's quite a big – I would say heavy action if you're experimenting a lot, right?

Just to give you like some analogy with software engineering, it's fine to make a commit with your source code, but not in development phase. For example, your unit test fails, right? You don't fix it and commit it right away, right? You probably fix one then run one more time. It fails again. You don't commit again. You don't commit. You make another fix. You run it, and finally it's green. And then you commit.

In machine learning, you do kind of similar way. You run one experiment, second, third, like fifth and have to do it again. You can say, "Okay. It's good. I'd like to share this result with my teammates." But you still need the previous ones. Data scientists usually reluctant to do like as many [inaudible 00:22:41]. Today we are working on kind of lightweight experiments. How to enable experiments, ML experiments without actually committing them into the Git, but still storing them? Still preserving them. And we will be storing this information in your data remotely, like S3 remote or some other storages kind of outside of Git, outside of Git workflow. This is how we try to – Like our complementation that Git introduces.

[00:23:16] JM: Tell me more about what you see as the bottlenecks in machine learning workflows and how did version control fits into that workflow. We've talked about it a little bit, but could you just go deeper into why it's necessary to have a data version control system just to reiterate the usefulness.

[00:23:38] DP: It all comes down to maturity of your process. How repeatable it is? How manageable it is. How can I return to my previous model? How can I get my result from three months ago? So all of those [inaudible 00:23:56], they're very simple. However, there are very, very few teams who can weekly answer this question, who can quickly get to your previous result. Who can quickly find your ML model, which didn't work well like last month, but it's still

interesting for some reason. It's really hard to find your previous results. And all these information is getting lost all the time. And this is why ML process is very fragile. It's really hard to kind of introduce well-managed process around just because there are no kind of good collaboration tools.

This is why you need a proper data versioning. This is why you need a proper kind of a product of working between your data team and DevOps team, or engineer teams. This is when you need tools to support your workflow.

[00:25:00] JM: The classic problem in machine learning development is that you have a data scientist who's doing some ad hoc testing and work in a Jupyter Notebook. And eventually they need to get their Jupyter Notebook into production. They need to test it on a larger quantity of data. And there are a lot of different ways that this tooling workflow proceeds. How does a data version control system assist in this kind of workflow bottleneck?

[00:25:34] DP: Yeah. This is a huge problem, right? It's like kind of a disconnect between your data scientist and your ML team. Sorry. Not ML team, but engineering team. How data version helps you, it can show you not just a notebook that was used, but it can bring you a model which was the result of your training as well as dataset, which was used for getting the final result. With data versioning, you can capture all the artifacts around. And your engineering team will know exactly like what happened. What dataset was used? What model was Git. And this simplifies the process of productionization, right?

So the problem that we are solving is improving cooperation with one simple goal. It's to decrease time to production. Time the team spends on moving your data products in, usually, ML models to productionize environment, to real systems.

[00:26:44] JM: How is the experience of a data scientist different from that of a software engineer working in machine learning?

[00:26:52] DP: Data science workflow is very different touch. If you go like deeper – Because it looks like it's the same. They're sitting in front of laptops and pushing the buttons. But if you go a little bit deeper, you'll see workflow is absolutely different.

First of all, they work with data a lot. They need to transfer the data. Sometimes it's not a small dataset. Sometimes it takes time to get this. Then they, in many cases, they do data modifications. They hug data as often as they hack code. And all these hacks needs to be recorded, right? Versioned. And they do experiments. Experiments, they're kind of similar to software engineering. If you do like small changes around your like unit tests and see like results. It takes more – Usually it takes more iterations. And some of your failed results also needs to be recorded. This is the crucial difference between software engineering and data science. Because if your experiments fails, it doesn't mean it's bad. It means that some people just didn't work behind it. You can still return back to these experiments in the future. Datasets firsts, experiments second. And the third, it's not easy to automatically evaluate the process. In software engineering, you got test paths on your CICD system and you're like, "All right, I'm good to go to production. It's good to deploy."

But in machine learning, it's different. It's metric-driven. And if you get, say, 0.7% accuracy improvement and 0.5% of more true positives. Is it a good result? Is it a bad result? I don't know. You need to understand the scenario behind this model, right? You need to understand the goals and the history of your experiments to answer those questions. There are no simple answers, like passed, failed. You need to be like metrics-driven and you need to make the decision based on some knowledge. This what makes ML also different.

[00:29:18] JM: We've talked about data version control. Now, to talk about your company, which is Iterative.AI, you have data provenance, which goes hand-in-hand with data version control. If you have your data and your models version controlled, then presumably you can do provenance. You can know when something happened, why it happened, and you have machine learning model management, which enables that as well. Tell me about what the company offers and why people would use it? What kinds of abilities does it give to a machine learning team?

[00:29:55] DP: Yeah. If you're talking about the company, we are, I would say, a usual startup with open source products, right? Today, there are a bunch of startup which have million like open source products and create some offering on top of this. We do the same. We have DVC, which is like fully open source. You can use the tool for data versioning for provenance, data

provenance. We have recently build a CML project, which is continuous machine learning. And CML integrates [inaudible 00:30:27] systems. Like GitHub actions or GitLab CICD.

The idea is simple. Just to utilize your existing IT infrastructure, your existing CI systems in order to make them work for machine learning workflow. And in additional to those two tools, the open source tools, we offer more business-specific products. And usually they're related to security and compliance. It's not something that individual needs. It's mostly about like business needs. Everything that individual needs, we give through open source for free. And if business needs some – For business needs, we have a special offering. This is not what we invented. This is a regular approach for like open source companies, right? There are dozens of those today and more.

[00:31:22] JM: If I'm developing a machine learning model, how am I leveraging Iterative.AI? What are the ways in which my – This platform is going to be useful for me in deploying and managing the models?

[00:31:42] DP: Yeah. First of all, you call this platform, right? But we don't call this platform. We build ecosystem. This is what I would it. We build separate tools, which can be connected or can be not. Apps to users. And we have DVC. People use DVC sometimes without any of our tools. We have built like CML. It's the same. People can use CML. People can use CML with DVC or without. And the same for our other product.

For example, if you're a small team, probably DVC and CML, it's enough for you to capture your data and models, to version them properly, to manage them and move them around properly. And CML, it can be enough for you to connect the data science team to engineering team through CICD process. But as a company, you might need something more. For example, you can say, "Hey, but like how much resources we have spent for this particular product? ML model. Who was working the most on this product? Who spent most of our computational resources? Who has access to this particular dataset? This kind of scenario, there are business scenarios. Small teams do not these scenarios. They don't care about this kind of organizational complexity, if you wish. And this is what comes. This is when our business offering can cover.

[00:33:16] JM: There's a variety of newer machine learning experimentation tools. There's a company called Weights and Biases that's going to be on in the near future. There's ML flow, which is a project out of Netflix. Tell me about how the system of tools that you are working on compare to these other experimentation tools.

[00:33:37] DP: Good question. Yeah, there are a set of tools for experimentation as you mentioned, like Weight and Biases, MLflow. MLflow [inaudible 00:33:45] Netflix is from Databricks. [inaudible 00:33:48] ML. The old system, ModelDB, and there are like maybe 3, 4 more tools.

[00:33:55] JM: Yes, I'm sorry. I mistake Metaflow for MLflow. That's the one that we covered recently from Netflix.

[00:34:04] DP: Oh, Metaflow. Right. Metaflow. Sorry. Yeah. Yeah, okay. Let's discuss Metaflow separately. Let's talk first about experimentation. And Metaflow, it's a big special I would say. So, first of all, yes, Weight and Biases. MLflow from Databricks, and ModelDB and some other tools for experimentation. What they do, they capture your result of your experiments mostly online metrics. When you run your model, you can see like what exactly is happening right now with your training process. So you can stop your training in some particular step, if you don't like to look around metrics.

And it also can preserve the history of your experiments when you can take a look at the metrics, right? This is important in the development stage. This is important for data scientists during the development process mostly. What DVC does, it captures the end result of your workflow. End result of your experiments, and you use DVC usually when you are ready to transfer this result to your team, or to production system, or somewhere outside of you.

So they kind of more focus on development experience. We are with our tools more focused on lifecycle, on lifecycle management. This is the fundamental difference. And you can find a lot of teams who use our solutions together with experimentation tools, like DVC with MLflow, DVC with Weight and Biases. So they're not kind of competitive tool. They complement each other. But like they're experimentation.

MLflow is a little bit different, because it has a component of data versioning. It has a component of pipelines. This tool, it more look like DVC. The difference is they tie to your Amazon infrastructure. This is would say is the biggest difference. We work with GitFlow on top of Git, and they use own kind of sets of convention on kind of Python APIs to do the same work. Python APIs on top of your AWS infrastructure. We do Git codification and whatever storage you prefer to use.

[SPONSOR MESSAGE]

[00:36:41] JM: As your infrastructure grows, you have SSH hosts and Kubernetes clusters in staging, production and maybe even on edge locations and smart devices. You want to implement the best practices to access them all, and that can be time-consuming. Security tools can get in the way of engineering work.

Well, there is a tool called Teleport. It's open source software. It's written in Go. It's a drop-in replacement for open SSH. It also has native support for Kubernetes. It's built by Gravitational. Teleport provides identity-ware access using short-lived certificates with SSO, session recording and other features, and it ensures compliance and audit requirements.

Teleport also prioritizes developer and convenience, because it's built by engineers for engineers. You can give it a try by going to try.gravitational.com/sed. There are links to downloads, documentations and a GitHub repository. That's try.gravitational.com/sed to try out Teleport from Gravitational. Thanks to Gravitational for being a sponsor. Again, if you want to try it out, you go to try.gravitational.com/sed.

[INTERVIEW CONTINUED]

[00:37:57] JM: So to return to the focus on dataset management, when I think about a model that I push to production. Once it gets to production, there're real API calls being made to that model. Things from my website that's going on or from my mobile app that's going on, and I'm wondering about the live data, the live data in comparison to the training data. When I'm accepting live examples, do those examples get saved indo the data version control system or

are we just logging the results of those inferences or whatever the machine learning model is doing?

[00:38:48] DP: Yeah. You are talking about online serving, right? About production system.

[00:38:53] JM: Yes.

[00:38:54] DP: So with DVC, with our products, we are focused mostly on the development stage and how to transfer this result to production stage. We don't do like productionization. We don't do serving. And this is actually like one of like core belief behind our company, our tools. We believe it should be like ecosystem of the tool. When each tool solve a particular problem, it solves it very well. We do data model versioning. We do connection with your CICD systems, with training infrastructure. And serving and productionization itself go separately. We are connected with those systems.

For example, we have – Recently, DVC was integrated as a part of Seldon, which is serving ML models storing system for Kubernetes. We have integration with Cortexlab, which is also Kubernetes-based, AWS-based, ML serving system. But we don't do serving.

[00:40:04] JM: Let's say I develop a machine learning model and it's going great and I'm training it and I roll it out to production. And then I realized, "Oh, actually this thing is classifying all the dogs as cats, or classifying all the stop signs as green traffic lights. I need to roll back my model. What does that rollback procedure look like and how is it engineered under the hood?

[00:40:38] DP: Yeah. This part, and this is like the most crucial part for most of the business. This part depends on your serving tool, right? ML model serving tool. What we can do and what we do the best is to provide your interface to model store. From production system you can say, "All right, this model doesn't work, but I need to return back to the previous model from my master branch," for example. Or I need to return back to some model from this experimentation branch, which supposed to fix this problem.

This kind of interface, this kind of like API that we provide and DVC provides. This is about like covering the gap between production system and development system. Because without

systematic approach, how it usually works, you go to data science team and say, “Hey, we got a problem in production. Can we return back our previous model?” And data scientist is like, “Yeah, sure. I have code. So, no problem. I have data. I assume this version was used. Just give me maybe like 5, 7 hours to retrain and I will get your model back,” right? So you don’t want to be in this situation when you need like 7 hours to [inaudible 00:41:58] to that. And this is why all these like lifecycle management are super crucial for business.

[00:42:05] JM: What’s the hardest engineering problem you’ve had to solve in building the data version control system?

[00:42:13] DP: Interesting. We got hundreds of those. On top of my head, today, we are working on experiments, for ML experiments. And we try to avoid, overcome the limitation that Git provides with the commits. I think this is one of the hardest problems we are solving during this – At least last few months. This is one of those. Optimization is always a big problem for us, always. So we spent so much time to optimization, and we’re still pushing hard on this direction. Because when you work with one million images, it’s never fast. And we have introduced like bunch of different algorithms to optimize this experience, because you don’t want to wait for like hours until your data moves or some changes happens.

And what makes DVC, development [inaudible 00:43:16] engine is user interface. We try to build tools which introduce kind of minimum overhead for developers. We need to understand like what engineers expect. What data scientists expects and how to cover their needs. Sometimes a very simple thing is while a lot of development behind, and this is what makes it like very complicated. How to put like a right message, for example. Error message. And how to understand what exactly what happened. The same thing actually with compilers. My first job was to build compilers. You know what? You know like in compilers, half of the code – Maybe not half, but like a big portion of the code related to like how to handle errors properly. How to put like a write [inaudible 00:44:13] messages for users. Kind of similar here with DVC.

[00:44:21] JM: How do you dog food the data version control system? Are you working with machine learning yourself in your spare time, or do you just consult with the community? Tell me about the process of developing the tool from real-world training.

[00:44:38] DP: Yeah. This is an interesting question. A good portion of requirements for the product came from my personal experience. Because I was trained as kind of mathematician. I studied applied mathematics department. I did some image process like 20 years ago even before deep learning. I was data scientist at Microsoft and I have seen how AI platforms are organized, how workflow is organized in mature companies, in mature teams. And I built tool-based of this kind of my experience. But at some point, it's maybe enough for the first version of the tool maybe for some iteration. But it's not enough to kind of the whole joining of the product, right?

Today, a good portion of requirement and feedback goes from our community. And this is a pour of open source. When hundreds of users every month came to you and ask for something. Like I cannot solve this particular scenario. What I do wrong? How tool can help me? Or I have this like issue. I don't understand what's going on here. It gives you like a lot of other materials to build a better product. And a good portion of requirements today – I would say bigger portion of the requirements today came from users, from our users, open source tool users.

[00:46:14] JM: Is model explainability a solvable problem? Or has it been solved yet? Because explainability seems like something that's going to be important if you're trying to figure out when to rollback and to what place to rollback to.

[00:46:29] DP: Yeah. I have actually have two point of view to this problem. First of all, of course, it's an important problem. Because otherwise, model is a block box for you, right? You need to understand what is going on. First of all, to understand the decision behind, and second, to improve your model, right? When you can explain why this is happening, you can use this information to build a better model. This is very important.

From another point of view of for business point of view, it might be important, but I believe in the long run we shouldn't focus on this like too much. Why? Maybe it's like controversial statement. But why? I think when you use data products in your business, your workflow should be metrics-driven. You shouldn't ask like why we rejected this long for this particular person? It seems like a person relevant. This is not questions that you need to ask your modeling team. If you decided to be data-driven, you should be data-driven. You should ask a different question. You should say, "Today, we got like 23% rejection rate on our loans. Why it was increased?" You

shouldn't ask about particular guy. You should think about the process. And the entire process needs to be metrics-driven.

When you process metrics-driven, when management believes in the metrics and they know how to manage the metrics, explainability, it's still kind of important thing, but it is not like a core part of the process. You don't need to go deep anymore if your process is well-designed.

[00:48:37] JM: All you need to know is, is this thing working properly? If it's not working properly, I can roll it back.

[00:48:44] DP: Yes, yes, exactly. Exactly. When a manager goes too deep into models, it's usually kind of a red sign. They shouldn't ask these questions, like why we adapted this model. They should be that metrics-driven. Because otherwise, if you don't believe your metrics, if you don't believe your models, like should you do that? Why don't you return back to your kind of more manual process when you can say who exactly reject them, right? And have a talk with him or her.

[00:49:14] JM: We've done several shows with Pachyderm, which is another data version control system. How do you compare to Pachyderm?

[00:49:22] DP: Yeah, Pachyderm, it's a pretty advanced system and it has data versioning functionality. But I see Pachyderm as a data engineering tool. Pachyderm has a scheduler, right? You can run job periodically. It has like very sophisticated result orchestrator on top of Kubernetes. And I would say it's about productionization of your results. It's like Airflow on Kubernetes with data versioning capabilities. This is how I see Pachyderm. You need Pachyderm when you go to production when you periodically retrain your models or process datasets.

While DVC, it's more about development and connecting development with your raw data on one side and production systems to another side. So it's kind of different stages of your lifecycle.

[00:50:18] JM: To close off, how do you see the machine learning space evolving in the next five years?

[00:50:25] DP: First of all, I believe we as industry will be more data-driven, right? We just described this like workflow and decision, like who reject us along? I believe we will trust more to the numbers and it's important to build processes around this like quantitative models, numbers and processes. This is a huge kind of like a cultural change and this is what I would expect from the industry.

Second, I do believe that data and machine learning is going to be like one essential part of software engineer. In the future, I don't believe we will have like data scientists or some like title like this. We are going to have like special types of engineers. And if you take a look at mature companies, like Google, Microsoft, in many cases, they prefer people with engineering skills to do modeling.

But what is important, like mental models have been different, right? You cannot just train an engineer to build some models. So those people has a little bit different kind of mental model in development. Yeah, so the gap between data science and engineering will be like smaller data becomes like an essential part of the workflow of the engineering tools and cultural change related to data-drive approach in doing business, trusting the models, building the process around numbers, quantity of numbers.

[00:52:08] JM: Dmitry, thank you for coming on the show. It's been great talking to you.

[00:52:10] DP: Yeah. Thank you, Jeff.

[END OF INTERVIEW]

[00:52:21] JM: Open source tooling is generally preferable to closed source tooling, because with an open source tool, you're going to know what code is running. You're going to know what the community is saying about that code. And you're going to have flexibility. But scaling open source tools is not that easy. You're going to have to spend a lot of time managing and maintaining that open source software. And the alternative is to use closed-source software, which will be scalable, but you won't know exactly what code is running. And you won't have an easy migration path.

Logz.io is a scalable and fully-managed observability platform for monitoring, troubleshooting and security, and it's all based on open source tools like the ELK Stack and Grafana. Logz.io is open source software at the scale that you probably will need if you are a growing company. Sign up for logz.io today by going to logs.io/sedaily and you can get a free t-shirt. You can go to L-O-G-Z.io/sedaily, create an account for logz.io and make a dashboard. Once you make that dashboard, you will get a free logz.io T-shirt.

Thanks for listening to Software Engineering Daily, and I hope you check out logz.io. That's L-O-G-Z.io/sedaily.

[END]