Shane McGraw: Hello. Welcome to today's SEI webcast, "Software Engineering for Machine Learning." My name is Shane McGraw, Outreach Team Lead here at the SEI, and I'd like to thank you for attending today's event.

We want to make our discussion as interactive as possible, so we will address questions throughout today's talk, and you can submit those questions in the YouTube Chat area and we will get to as many as we can.

Our featured speakers today are Dr. Grace Lewis, Dr. Ipek Ozkaya. Grace is a principal researcher, Lead of the Tactical and AI-enabled Systems initiative at the Carnegie Mellon University Software Engineering Institute. Lewis is a principal investigator for the Characterizing and Detecting Mismatch in ML-enabled Systems and Predicting Inference Degradation Production for ML systems, both of which are research projects here.

Ipek is a technical director within the SEI CMU, and her current work includes developing techniques for engineering AI systems, with a focus on ML and search-based techniques, in improving software evolution efficiency, with an emphasis on software architecture, software economics and managing technical debt.

And now I'd like to turn it over to Dr. Ipek Ozkaya. Ipek, good afternoon. All yours.

Ipek Ozkaya: Good afternoon, Shane. Thank you very much. Welcome, everyone, to this afternoon's webinar, and Grace and I am very excited to have you all listening in. The topic of our conversation today is "Understanding Software Engineering for Machine Learning Systems." As you all have been observing, as well as we are, that there's an increasing amount of interest in incorporating AI and ML components into systems. This comes with its advantages, as well as its challenges. We've observed that industry and duty alike are challenged sometimes to put these systems into production. It's one thing to develop an ML model, but it's a completely different thing to really develop it and being able to deploy as well as update it as needed.

So today, Grace actually will be talking about some of the projects that we've been doing and some of the progress that we've seen in terms of research, as well as the challenges in the LSG. With that, I will just get out of the way and leave it to Grace. Grace, all yours.

Grace Lewis: Thank you so much, Ipek. That was a great introduction and perfect to get started right away on our topics for today. So as Ipek said, one of the things that we've been working on lately at the SEI is this idea of software engineering for machine learning systems. Like Ipek's already said, one thing is to develop a model and a completely different things is to get that model into production. While there is a lot of guidance out there targeted at, I would say, at the data science practice, you know, how to develop perfect models, how to increase accuracy, you know, how to evolve them and things like that, there isn't a lot in the field of, like, software

engineering. How do you take that machine learning component and put it into a system and ensure that it's doing what it has to do?

So the goal of this webinar today is to introduce you to two different projects that are looking to make progress in this area. One of them is Characterizing and Detecting Mismatch and ML-enabled Systems. This is a project that concluded in December of 2020, and we're happy to report on the results of that project. The second one is one that is just starting. It's called Predicting Inference Degradation and Production in ML Systems." It started in January of this year, and we do believe that these two projects are going to contribute greatly to this growing field of SE for ML.

So why? Why do we even have this idea or this discipline of software engineering for machine learning? Well, the thing is that machine learning components are parts of much larger systems. From a software engineering perspective, they're code. They're complex code, but they are code. If you look at the diagram on this slide, it's from one of the seminal papers from Sculley, a Google employee, and he shows in this graphic that the ML code, which is that little black box right at the side of diagram, is just a very tiny piece of a much, much larger system. In order to get that ML code to perform correctly you need to do data collection, you need to do feature extraction, you need to have a serving infrastructure, need to have monitoring. You need to have data verification. So it's really just one component of many components.

One of the very big challenges that we see with machine learning components is the fact, and this is probably what makes them different from I would say traditional components, is that they're very data dependent. Their performance in, when they're put into production, really depends on how similar the operational data is to the data that it was trained with. This is often called the training-serving skew.

So what happens is that in addition to all the things that are mentioned below, there are two things that are key. Systems need to provide a way to know when model performance is degrading. There has to be a way for the system to know that that is happening, and the second is that the system also needs to provide enough information for retraining. If the model's not performing well, a system better be logging good data so that it can be retrained properly.

So basically saying, again, what I just said, is that they--these types of systems really need to be engineered. They need to be instrumented for runtime monitoring of those components. They need to be developed such that the trainee-retraining cycle is shortened. You know, the better information you gather the better monitoring information you have, the quicker you're going to be able to turn a new, you know, turn a new model around, and also you should--it should be easy to integrate components such is straightforward so that we're not taking a long--a lot of time, we're not creating tons of glue to be able to integrate a component.

There are lots of software engineering practices that apply directly but that are not--simply not used in the data science field. So for example, from one of our recent engagements on industry, we see, for example, that for software engineers, unit testing, it's just like, you know, it's bread and butter, it's an everyday thing, everybody knows how to do unit testing. But those practices don't--even though they translate directly, they're typically not used and not known by the data science field, so that's an example of one of those practices.
Other SE practice, so software engineering practice, are going to have to be adapted and extended to deal with machine learning components, like for example, monitoring. Monitoring has to extend beyond us, you know, making sure the--just making sure the component isn't crashing or that, you know, it's performing correctly, but really looking at performance but from an accuracy perspective, for example.

So Ipek, before I get started on reporting on the actual projects, are there any questions?

Ipek Ozkaya: I don't see any questions yet, but let me ask you a question, which--

Grace Lewis: Sure.

Ipek Ozkaya: --you mentioned some of those practices like testing and monitoring does not necessarily transfer, and software engineers need to pay more attention to it from the perspective of ML systems. How about the other way around? Are there any practices software engineers need to pay attention to when they're developing? Because that combination really gets in the way and we're calling it developing the AI engineering discipline in the broader sense at the SEI, but there's that how software engineering can help ML and how--what are some of the practices that software engineers need to be aware of?

Grace Lewis: Yeah. So let me--actually, let me use the unit testing example as a--also as an example of something that software engineers need to keep in mind. So when software engineers, when we develop--when we develop unit tests, right, we basically say, "Okay. These are the imports that I want to test against. If the output of this particular component is 5, then that, you know, that--it passed. If the output of this component is -3 then that's a problem, that's a fail," and so we're really used to these, you know, like, these deterministic, you know, answers. Like, this model, this component, produces this output. But we really need to be aware of the nature of machine learning components, where, you know, right now, based on, you know, what it's seeing right now, that model might produce a 5 or it might produce a 7, and those are still valid answers, because, for example, what we are expecting to see is a range of things.

So for example, for a software engineer being able to express those outputs not in deterministic terms but for example in ranges or for example by talking to a data scientist, I could say, "Okay. You know what? The output doesn't have to be, you know, 5, 6, 7, 8, but I really do expect these outputs to be in this order." So just changing into that mentality that it's not always going

to be deterministic, and for example, being able to write unit tests different, is one thing--one example of a practice of that type.

Ipek Ozkaya: So I think bottom line what you're saying is some of our known assumptions will definitely not hold and we do definitely--

Grace Lewis: Absolutely.

Ipek Ozkaya: --need more empirical evidence to be able to look that, which is actually the project that you're just going to talk about, so thank you.

Grace Lewis: Exactly. Thank you very much. Okay. So let me start with this project. I said this project just ended, and I'm happy to say that it successfully ended. So you're going to hear a term, use a term, called ML-enabled system. What we're defining an ML-enabled system is a software system that relies on one or more of these machine learning components to provide our capabilities. So first of all, these machine learning components, it's basically a trained model that is wrapped as a special type of software component that we're going to be calling an ML component.

That ML component is going to receive some kind of processed operational data from one software component and it's going to generate an insight, a prediction and imprints, that is going to be consumed by another software component, and on top of that, these--there are run--there are going to be runtime monitoring tools in the operational environment that are going to be measuring the accuracy or the performance of that machine learning component. So that is an ML-enabled system. We don't--it's not a stand-alone model. Is not something that runs on a desktop in the corner. It's a machine learning component that is part of a much larger system.

And one of the problems that we had been seeing that really triggered our research in this area is that--is this problem that we're calling the problem of multiple perspectives. If you look at how these system-- <audio cuts> --eloped, they're developed by three different teams with three different knowledges in three different workflows. So for example, you have the data scientists that are typically trained in statistics, in machine learning, and they, at the end of their process, they go from raw data all the way to a trained model. That model is then passed on to a software engineering team for integration into a larger system and testing, and finally, that model is then deployed and then is monitored by an operations staff.

So these are three different stakeholders, sometimes three different languages, which is something that we observe in our research, and three different perspectives, and the problem with this is that because these teams often do--often don't work together and sometimes often don't communicate, they tend to make incorrect assumptions about other parts of the system, and we call these implicit assumptions.

So as a data scientist, I could make a very wrong assumption about the environment in which the model is going to execute. As a software engineer, I can make a completely wrong assumption about the language that the model is developed in, so those--this is what we mean by mismatch. So we define mismatch at a problem--as a problem that occurs in the development, deployment and operations of a machine learning system, and that problem happened because of incorrect assumptions that different stakeholders made up of different parts of the system and that resulted in a negative consequence.

In addition to that, we also posit that ML mismatch can be traced to there should've been a piece of information that should have been communicated and was not, and if that piece of information would've been shared, that problem would've been avoided.

So what are some examples of mismatch, just to get some clarity on--about what we mean? So for example, there could be a computing resource mismatch. So the poor system performance because the computing resources that were used, for example, for model testing were very different from the computer resources available in the operational environment. Or you can have a data distribution mismatch, which is actually very common, which is that you have poor model accuracy because the training data really doesn't match the operational data. Or you can have an API mismatch, where you have to generate a ton of glue code because the machine learning component is expecting completely different inputs and outputs than what is provided by the system in which it's going to be integrated. Or you can have test data mismatch, which is you know what, the software engineers really couldn't properly test the component because they had no idea how to do it. Or there could be a monitoring mismatch, which is basically--or a metric mismatch, which is, "Sure, we have tools for monitoring in the operational environment, but they're not really capable of measuring that--whatever that goodness metric is for the machine learning component," which is typically accuracy, for example.

So what is it that we developed in the context of this research project? So we developed a set of machine-readable descriptors for elements of machine learning-enabled systems. So basically we developed descriptors for the different elements that you see here in green, and the--they're machine readable, and you'll see why in a second, and why did we create this? So first of all, we want them to serve as checklists. So for example, let's say that two different teams or three different teams are part of the machine learning at system development. You can imagine, for example, that the data science team would create the descriptors, for example, for the trained model and the training data. It could be that the operations people, because the raw data is coming from the operations environment provide the descriptors for the raw data. The software engineers provide the descriptor for the development environment, but basically they serve as checklists. So for example, when a model--when a trained model is handed off, we could make sure that the--all the information that I need in order to avoid mismatch has been included.

Similarly, it can provide stakeholders with examples of information to request and/or requirements to impose. So for example, if I'm an organization, this is very typical in a department of defense, where I have contractors, for example, developing models, I can say, "Yeah, when you deliver your model, make sure you also accompany it with these particular descriptors."

And the third one, which is the one that we're very excited about and hopefully we get to continue working on that, which is what we want, is that we want to be--these descriptors, we want them to be able to be leveraged by tools that can do automated mismatch checking. We have to make sure that everything is in agreement, and the reason also why that is important is because if we want to be able to detect mismatch, for example, at runtime, it also dictates different system elements that also need to be present in order to do mismatch detection.

So as seen in the diagram on the right, we expect them to be used by system stakeholders, let's say, in a manual way, for lack of a better word. So just for information awareness or for evaluation or things like that, but we definitely also, the end goal, is automation. We want to be able to use these descriptors and tools, that kind at design time or runtime say, "Hey, there's a problem here. There's a mismatch."

I see that there are questions, Ipek. Go ahead, before I go into the study. I think this is a very good time.

Ipek Ozkaya: Yes. So there's actually a question from Matt. I will take them in different order, because I know about the study. I know what you'll talk about. I'll take advantage of it.

Grace Lewis: Okay. Okay.

Ipek Ozkaya: The question is, "Can you give a very quick overview of the types of systems, organizations and environments that was included in the study, and in particular, did that include systems that allow or follow CI/CD pipelines and life cycle?

Grace Lewis: Yeah. So unfortunate--there's not a lot of--because of the IRB, there's not--I can't report on specific systems, obviously, but I would say that we had I--we had a good representation. I'm going to talk about that in the study. We had good representation from industry, both large industry as well as small basically startups. We had also a very good representation from Department of Defense, both government and contractors, and I would say very, very different types of systems, going from very massive systems, like, you know, we interviewed people from the large industries. I'm sure you can imagine what those are. But for example, we also interviewed people from startups where their product is one system, but that system has to, you know, perform very well.

There was also representation from the larger companies or that tend to be the ones that have more automated processes, and actually, the second project's going to--is going to talk a little a little bit about this, where for example, models are trained on a daily basis just because they can, and/or for example, oh. There were also small companies where models are really retrained, for example, and don't really require, because it's just like one product, you know what I mean, they don't have very complex CI/CD pipeline. But I--that's what--I want to save some of that for the--for when I talk about the study.

Ipek Ozkaya: Yeah. But all of them were organizations following modern software engineering practices?

Grace Lewis: Correct. We--

Ipek Ozkaya: Up-to-date tooling, CI/CD life cycle, an awareness of all those. So I think that's common across--

Grace Lewis: Right. Correct. And also, both for the interviews as well as the survey, one of the requirements to participate in both is that you had to be part of organization that was developing systems that fit into our definition of ML-enabled systems. If they had to be a part of a team, of a group, of an organization, that was developing those types of systems in which the components are part of a much larger system. Yeah.

Ipek Ozkaya: And I'll tell you the next question, but you might maybe prefer to answer it while you get there.

Grace Lewis: Sure.

Ipek Ozkaya: Because we do talk about programming languages and their mismatches. So the question is from Dwaine, and he asks, "Is R an absolute necessity for hardcore ML or can standardizing one pipeline improve ML mismatch?" So... And you can--

Grace Lewis: Oh, my gosh. Yeah, let me--

Ipek Ozkaya: You can answer now or you can answer when you get there, but I wanted to (inaudible) you with it.

Grace Lewis: I'll answer when I--I'll answer when I get there, but you know what? That is such an "It depends," question and I hate to give those answers, but really, it depends on skills. Now, from a robuster's perspective, I will say that Python is a much more robust language, and if it were my preferences, that's probably what I would do. But R, from a statistics perspective, it's a lot more, I will say, it's a lot more complete in terms of libraries, so it's kind of one of those--I

don't know. It's one of those, like I said, "It depends," questions. However, and I'll, even though I'm going to talk about this, we had--there were so many mismatches that we had in the study related to porting problems, and you'll see why in a second.

So for example, if you have data scientists develop a model in R, and then it's just handed over to the software engineers and now they have to reimplement it in Python, there's so much room for error. So we definitely did see mismatches such as that.

Ipek Ozkaya: And programming language is definitely, that's there the devil is in the details, and that's where we actually get all the system problems, and so we saw some of those as well. So I won't--

Grace Lewis: Correct.

Ipek Ozkaya: --derail you any further, and we'll get to some of these conversations as you explain more about R data.

Grace Lewis: Absolutely. Thank you for the questions. I really appreciate it. It's so much--it's good to have, like, those little breaks.

So now just let me talk a little bit about the study protocol that we followed. We divided the study into two phases. In Phase 1, what we did was that in parallel, we conducted a set of practitioner interviews. I already told you what type of practitioners these were, and the goal of those interviews was to elicit mismatches and consequences. So basically it was asking the same question over and over again. "Can you tell us an example of a mismatch that occurred because you made an incorrect assumption or somebody that you work with also--or made an incorrect assumption?" and we gathered a series of mismatches and consequences, I'll show those in a second, and then we followed that with a practitioner survey where we basically said, "How important is--how important do you think it is for you to know this information in order to avoid mismatch?" and so definitely out of that survey came a set of validated mismatches and consequences, and I'll show you some of the results from that.

In parallel, we conducted a multi-vocal literature review, combining both white literature and gray literature, where we basically mined these literature. It's literature for best practices for ML systems, particularly looking at pointers to, like, "This is how you should document systems. This is information that needs to be shared," so really mining them for best practices in terms of information sharing, and out of that came what we're calling system attributes, and you'll see that in a second.

Phase 2 of the study consisted of, "Okay. We have a set of validated mismatches and consequences. We have those system element descriptors or/and attributes. Now let's do a mapping between the two."

Which are the system elements that I would need to know in order to avoid this particular mismatch and do that mapping? And then we did a gap analysis where we basically say, "Okay." Sorry. I'm doing this because that's how they were in the...

So for this particular mismatch are there any, I mean, is it the case that there are no system attributes that could be able to detect that particular mismatch? And the other way around, are there attributes that came up as best practices but don't really map to any mismatches? And so what we did is based on our knowledge, in addition to other literature that we reviewed, we say, "Okay. Let's fill in these gaps," and then basically we created a set of descriptors, a set of seven descriptors that we defined in JSON Schema just because, as I said before, our end goal is really automation.

So what I'm going to report on today is this--only this part, and hopefully in the next--hopefully in a follow-up webinar I'll get to talk about the other phases. But we're going to talk simply about Phase 1, the--and the elicitation of mismatches and consequences and what we are seeing are the main causes for mismatches, at least among the people that we interviewed in survey.

So as far as interviews, we conducted a total of 20 interviews and those, from those 20 interviews, we did very--we followed very, very, know, sound practices and we came up with a total of 140 mismatches that led to 232 distinct examples of information that was not communicated that led to mismatch. This goes back to the question that was asked through here. As far as affiliation, 60 percent industry, 30 percent government, 10 percent research lab, and primary role? Mostly data scientists, and actually this is some, in full disclosure, this is one of--we believe this is one of the threats to validity, that it's very data science heavy, which is why we want to repeat this study. We would like to repeat the study with a more distributed or--and a better distribution of people, so especially so we could get more of the operations perspective.

That said, it was very hard to get people in the operations role. It's, again, it--a lot of people are--which shows that a lot of people are still kind of in the model development phase and they're really haven't gotten to that operations phase where they're monitoring that MSU learning system and doing those things. So yes, it's a problem with the data, but also it's--it was simply--it's almost like it's not a known thing. Not a known thing, but how would I say this? It's operations people are not typically associated in the type of development of these systems, which we think is a problem.

So basically when we took those, all those examples of mismatch, what it came down to was this chart that you're seeing right here. So 30 percent of the elicited mismatches had to do with

incorrect assumptions about the trained model, followed by 60 percent that, incorrect assumptions, about the operational environment, 50 percent about task and purpose, and the others in smaller amounts, and what I'm going to do is I'm going to go into detail into one of these, and for each one of these I'm going to give you a breakdown of what were the subcategories and also give you an example of a quote from one of the interviews that I'm sure that many of you will be able to relate to, because we definitely were able to relate to it.

So as far as training--as far as a training model--a trained model, sorry, it was an even split between test cases and lack of test case--information related to test cases and test data, and lack of information about API and specifications that led to the most mismatch examples. So this example comes from a person in the software engineering role that basically said, "Look. I was really never able to get from the data scientist a description of what the components were, what were their specifications, were there any tests and so we could reproduce the results," so this was a typical example where, again, a model is kind of like I don't, I mean, it's an idiom but like kind of thrown over the fence, and it's like, "Okay, integrate it into the system," and there is no formal specification or even informal specification of that model that would help me integrate that model to the system.

Other examples of subcategories included, for example, a lack of evaluation metrics. For example, the operations people put the model into production, but they weren't sure exactly what it was they were supposed to be monitoring. Programming languages. This is exactly the example that I talked about before, that, for example, the software engineers had no idea that the model was going to come in R and that totally messed up their schedule. So yep, those are some of the subcategories under trained model.

As far as operational environment, this was the second-highest category. A large number of the mismatches were related to runtime metrics and data. Basically, again, the model was put into operation and either the operations folks had absolutely no idea what it was that they were supposed to be monitoring, or the other way around, the data scientists just assumed that this was something that the operations people know how to do and they were collecting all this runtime metrics and data that they needed, and so this is a quote that comes from one of the data--from a data scientist where basically, you know, the models were into production, the model produces a bad prediction, and the, you know, the, for example, the operations team comes to the data science team and says, "Hey, the model isn't working," and they're like, "Okay, we--can you show us, you know, what data you were collecting? Where are the logs?" and it's like, "Well, unless you tell us, you give us some more information, we really can't reproduce that event."

So it's this idea again, and this is the whole idea about mismatch. If this information was shared, you know, if operations and data scientists were acting with the same information, this mismatch wouldn't have occurred.

Next is task and purpose. This basically a rough way to think about task and purpose is almost like the model requirements. So basically, what are things that need to be communicated, especially communicated, for example, between project owners and data scientists, so that data scientists build the right model? So for example, 29 percent of mismatches had to do with lack of business goals, and I really love this. I really love this quote. Is one quote--it's a quote that made me laugh a lot, but we heard it, we heard the same quote from many people. It's just that this one, this person, just said it in the right words. This was a data scientist, and she said that she felt like the most broken part of the process was exactly this task and purpose because a lot of the time, you know, somebody comes to a data scientist and said, "Here, here's a bunch of data. Go do some data science on it," and they're like, "Well, okay. With what purpose?" and then what happens is that the specification of the problem is left in the hands of the data scientist and so there's a lot of churn and a lot of going back and forward, you know, "Bring me--" the, "Bring me the rock" syndrome, right. "Is this what you wanted?" "No, it's not." "Is this what you wanted?" "No, it's not." "Is this--" so there's a lot of going back and forth, and again, if these, if this information was explicitly shared from the beginning we wouldn't have these problems.

As far as raw data, just to clarify what raw data means, raw data is the raw data. It's not the training data, it's the data that--it's the data sets that come as is and that the data scientists have to transform, or the data engineers, depending on your organization, has to transform into training data, and really most mismatches had to do with lack of metadata.

So one thing is to have, for example, a data dictionary, which is the second, this, over here, that was the second most popular, where a data dictionary is kind of like the traditional data dictionary that most of us in the software engineering field know, right. This input is of this type and this is what it is and these are the values and all that stuff, but metadata was the one that is most important because it's basically, "Okay. It's great that I know what the data is, but for me to do my job, for me to do my data cleansing, for me to do my data preparation, I need to have some more information."

So for example, where is this data coming from? When was it collected? What do these missing values mean? What is it--is what I'm seeing, is that normal? Is that an outlier? Am I seeing seasonal affects from this data? So being able to have that metadata can definitely help the data scientist do a much better job and that is something that they felt that is not often shared. Once again, you know, "Here's some data. Go do some data science to it," right. So that was--that's related to some of the raw data subcategories.

As far as development environment, definitely the number one mismatch had to do, and you--I guess you guys predicted it--had to do with programming language. Not knowing--not sharing information about the programming language that was used to develop the model, or the other way around, the ML framework, tools and libraries, and for example, this was a quote from a

software engineer that said that some of the weird--they saw so many failures trying to port models from R, so for example, Python, and, for example, one--there was one specific instance that was shared with us where they were trying to reproduce the model in Python and it just wasn't--they weren't getting the same answers, and they realized that there was an underlying, very low-level difference, between the way two different libraries were handing floating point numbers that was causing the differences.

So for--so again, ideally you want to be in an environment where you don't have to do any porting, but again, if there has to be some porting done, that's something that has to be shared from the start so that people can make decisions on whether, "Yes, we do have to port and we have to put that task of porting into the schedule or we're just going to agree that this is the language that we use, period."

Going to the next category, operational data, this is the data that the model actually encounters at one time, and most mismatches had to do with lack of data statistics, and this goes back to the training-serving skew, which is that the, you know, data scientist trained the model and for one reason or another the model wasn't trained with data that really represented the operational environment. So number one was data statistics, knowing--having much more information about the statistics of the actual operational data that the model is going to encounter, and I guess tied for a second is this idea of, oh, sorry, of data pipelines and data sources.

Another thing that we saw as problematic, and this was constant in the--in some of the interviews, was this idea of not knowing, like, what part of the data pipeline is outside of the model and what part of the data pipeline is inside the model? And this is something that shows up again with respect to training data, and so for example, having information about data statistics, about what exactly do the data pipelines look in the actual operational environment, and, for example, what the actual data sources are. That is definitely some information that should be shared, because that allows, for example, data scientists when they're building their models, to make much better decisions.

The next category was training data, and this is something that showed up both in the operational data and the training data, but most of them had to do with just data preparation pipelines. The fact that, for example, when the model is going over to, you know, hand it over, when the model is handed over, all the data preparation quote is either often not handled or was done in an ad hoc way and therefore has to be reproduced in the development environment, in this case, and so for example, this particular quote, it's kind of--this quote is interesting because it touches on a different--on several mismatches, but basically they said that a group developed their architecture for a whole machine learning pipeline and--but the pipeline went a little bit into future engineering as well. So that, that boundary between, "This is the data pipeline and this is where the modeling starts," was a little bit too much into the right, so, for example, and that

precluded them from doing, like, different alternatives, for example, into future engineering, and they were kind of like locked into an architecture.

I guess a very specific quote, but it shows why it's important to have that kind of like that very clean separation between, "This is the data pipeline; this is the model component."

And (inaudible)--

Ipek Ozkaya: Can I stop you for some questions here, Grace?

Grace Lewis: Yes. Yes. I was going to stop right here. This is perfect.

Ipek Ozkaya: Okay. Because there's very good discussion and questions going on in the chat.

Grace Lewis: Nice.

Ipek Ozkaya: I don't want you to miss it. So one is Shrikanth asks whether, if we can repeat when an ML mismatch is in particular? Is it between practitioner opinion and empirical evidence. So let's clarify again for the audience so that they can follow the rest of it as well.

Grace Lewis: Absolutely. So what we define as machine learning mismatch is a problem that occurs when you're trying to put a model into production, and that problem occurs because different stakeholders made incorrect assumptions about other parts of the system. For example, as a software engineer, I made a totally incorrect assumption about the language that the model was developed in, or as a data scientist I made a incorrect assumption about what was the expected accuracy for this model. Or as an operations, part of the operations staff, I made a very incorrect assumption about what was the data that I was supposed to be logging in the operational environment? So that's one part of mismatch.

And the further explanation of mismatch is that for our study the idea was that we would, every example of mismatch, including the ones that I just mentioned, could be traced back to a piece of information that if shared would've avoided that mismatch. So it's not a--it's not mismatch between empirical and practitioner, but rather, mismatch between different stakeholders that participate in the development of this ML-enabled system. Hopefully that answers the question.

Ipek Ozkaya: And as our audience will also find out as we (inaudible), the idea is that you can actually formalize and maybe automate parts of it so that--

Grace Lewis: Yes.

Ipek Ozkaya: --you can be more consistent among the information share.

Grace Lewis: Absolutely.

Ipek Ozkaya: So next question is from Matt, from sunny Pittsburgh, Pennsylvania, and he asks, "Did you find situations where the fact that this was a system with an ML component added additional requirements on the ML model that weren't obvious until the system was deployed?"

Grace Lewis: Repeat? Repeat what you said after ML compo--I missed the last part of the question. I'm sorry.

Ipek Ozkaya: Let me repeat it all. Did you find situations where the fact that there was a system with an ML component added additional requirements on the ML model that weren't obvious until the system was deployed? It's really going back to what you could observe in deployment versus what you could gather during the requirements phase.

Grace Lewis: Yeah. So here's one of those--actually, yeah. That's a very good question. So here's one of those. Going back to the very first question that Ipek asked me that said, you know, which practices change or the which practice have to change or which practices can be used the same and all that stuff. So requirements engineering for these types of components, actually--it's actually quite an active area right now, obviously because of the, you know, the interest in machine learning, and I would say that from a requirements engineering perspective, all the techniques that we use to gather requirements are exactly the same, right, because in the end, going back to what a machine learning component is, it is a piece of code. It's just the--a special piece of code. But what changes are the questions.

So let me give you an example of that. So for example, a question that we might not be used to asking during requirements engineering would be, "What is the risk of having a--what is the risk of the model producing a false positive? What is the risk of the model producing a false negative?" That is a question that we normally wouldn't ask during requirements engineering, but for this type of system it's very important to ask because, again, as software engineers, we have to live with the fact that these models are going to degrade over time, that they're very dependent on data, and that any changes in the data are going to change the performance of the model.

So that's an example of something that--where this requirements engineering. Now, a specific thing that you wouldn't know until after the model is deployed--so the examples mostly, to be honest, mostly had to do with monitoring, and this goes back to why we think it's so important. We, no, the model was performing. It was doing great. All of a sudden it started producing a bad prediction, and the data scientist had no data to work from because nobody thought about logging this particular information or this particular combination of input, outputs.

So yeah, there are lots of--I can think of many examples but I don't want to get into them, but the whole point behind what we're trying to do is to make all that information explicit so that that doesn't happen. If different parts--if different stakeholders are feeling out different descriptors for the system, all that information is being shared and so a data scientist could say, "Ah, I understand now. This is what the operation environment is," or the software engineer can say, "Okay. This is how the model was tested. This is all the testing that has already been done. This is how it was evaluated," et cetera.

Ipek Ozkaya: And some of the mismatches that were very worse between--the mismatch between trained model and the operational model could get into requirements practices, but some of the data science practices could also actually already be able to detect those. So there's--

Grace Lewis: Correct.

Ipek Ozkaya: --aspects of both that are actually involved. But I agree, it's an excellent question. The next question is one that is both near and dear to both your heart and my heart, Grace, because it's related to architecture, so...

Grace Lewis: Yay.

Ipek Ozkaya: And there's a couple plus-ones for this, which goes, "Did you see any mismatches as of expected quality attributes versus as architected quality attributes? Example, performance latency scale needed to be handled but was only built for half of that." So, like--

Grace Lewis: Oh, absolutely. Absolutely. Oh, there's--okay. I'm going to, like, leave it to two, but because we--I don't want to run out of time before I get to my slides, but for example, and again, this is--data scientists are great. They're great people to work with. They are super-knowledgeable. I would never know all the things that they know. But they're not trained in software engineering. They don't--they typically don't know what quality attributes are. They typically don't--yeah. They don't understand. They are very good at building their model, about meeting, you know, 95 percent accuracy. Once they reach 95 percent accuracy they're very happy. They say, "Okay. The model. Here it is." But for example, you can imagi--okay. Here's a situation. You develop a machine learning model, 95 percent accuracy. You put it into production, and all of a sudden, the data rate, the data rates for how the data is coming into the model, the model just isn't--it's just not capable of performing--ah, see, here's the thing. Performance the way that data scientists use it and performance the way we use it as software architects is very different.

So the data scientist talks about accuracy, right, so let's just use accuracy. So, for example, all of it, you know, this, it's just it's not performing. I mean, it just can't handle data that's coming in. So all of a sudden, now you have to replicate that model four times, so you create--you put in a

load balancer, you create four instances of the model, and then what happens, which is very interesting, is again, models are very data dependent, right. So if you start load balancing, different data starts going to different instances of the model and now you have four identical models that are performing in the data science sense very differently, and anyway.

But those are examples of things that we have to keep in mind of were there any--would anything have been done different by the data scientist if the data scientist knew, for example, that the model was going to be replicated four times? Or would the data scientist would have done anything different if the data scientist knew that the data was coming in at this speed instead of that speed? So yeah, there are many like that. I think the main one is performance and scalability. Those are ones that are very dear to our hearts as software architects, but it's just not something that the data scientists envision. They just produce a model and they produce an awesome model. But thinking about how to put it into production is a whole different conversation

Ipek Ozkaya: And the monitoring aspects and how the data (inaudible)--

Grace Lewis: Oh, yeah.

Ipek Ozkaya: --and the department aspects, there's-- there were--so I guess in short, yes, there were quite a number of those that were identified and we're actually actively working on both doing case studies, working with our organizations, as well as our research. So once we have more substantial information accumulated we'll definitely come out and do another webinar and address this for sure in detail.

So Grace, in the interest of time, let's move on, and--but there are great questions coming.

Grace Lewis: Oh, thanks. That's great. So now, obviously, we came up with the mismatches. Now we needed to validate it, and basically, we conducted a survey where the main question was, "How important is it for you as a practitioner to be aware of this information in order to avoid mismatch?" We had 31 responses to our survey. Not as much as we wanted to, but then again, we were very specifically targeting people that were involved in the development of machine learning-enabled systems, and because this is kind of a new discipline, outside of the large companies, it's hard to get people that are working exactly in this area, and it was great. I mean, the results were great.

As you can see in the diagram, the importance of sharing information for every single category was mostly rated between "Important" and "Very Important," which was good for our study because it showed that the information that we had gathered at least was representative of information that these survey respondents thought should be shared.

Now, in observation, and this is--this shouldn't be surprising to anyone--but what is important varies per role, and those three are--so three, they're very interesting examples. So for example, knowing the programming language. It's funny because those of you that are here, for example, that I imagine maybe come from the software engineering domain, Software Engineering Institute, right. So for example, knowing the programming language used in the development environment, or sharing that information for software engineers, 80 percent, for operations a hundred percent, but for the data scientist only 69 percent. Knowing what computing resources are available in the operational environment, a hundred percent for both operations and software engineers. Data science, 88 percent. Evaluation metrics--I am so sorry.

For the data scientists, a hundred percent, because this is what they need to do, or to have, in order to, you know, to do their job, whereas for software engineers, for example, was only 80 percent. But again, for operations a hundred, because this is something where these two perspectives are the ones that are most involved. So I think even though it was a small-scale survey, this particular observation to me is very important because I think it makes what we're doing even more relevant, the fact that for different roles different things are less--or more important but that you need to look at them together, because they're important for somebody.

And so that finishes the results of Phase 1 and this is what we did in Phase 2. This was already concluded, but we're not reporting on this. So after we had the validated mismatches and the consequences and the different system element, the system element descriptor attributes from the multi-vocal literature review, we did a matching between the two. So basically have this huge spreadsheet where we had all the mismatches here as rows, and then we have the different descriptors on the top as descriptors and their attributes. So this is--where is it? Good. This is trained model, this is training data, raw data, operational data, task and purpose, and what we did was a formalization.

So formalization, what this means is what would indicate that there is a mismatch? So for example, in this particular case there's a mismatch when if you add A1 and A2, that's greater than A4, or there is a mis--there's a mismatch when these two are the same. Actually, that probably should've been different. Or there is a mismatch between when if you do a chi-square test between these two attributes, it's not--it doesn't fall within threshold. So that's what I meant by formalization is understanding which of these attributes could lend themselves, going back to what Ipek said, to automation.

So we did that. So then we did a gap analysis. I briefly talked about this before. So for example, I mean, this is a very small example, but for example, this attribute right here, it doesn't contribute to any mismatches, so we did analysis as to why, and is there maybe a mismatch that wasn't mentioned in the--particularly in the interviews, but maybe came up in the literature view as something that we should be watching for? And the other way around. For example, this one right here. There is a mismatch, which, I guess we could call it 5, that--of the attributes that

came out of the reviews. It didn't really--it doesn't seem like there's a way to detect those, so maybe that's a example of one of those that is probably not--we can't formalize that particular one. But it was important to know which ones we could and which ones we couldn't.

So after we did that, we did descriptor formalization, and basically what we created was a JSON Schema, and where we have--there's a JSON Schema descriptor for each one of these, for each one of the seven categories, and then the attributes are encoded using a JSON Schema notation.

Oops. And so our vision. I just want to--we've said this many times, but our vision is automated mismatch detection. Like we said, we already--all those attributes have been codified into machine-readable JSON Schema because we want to be able to create tools, us and the community, that use it and say, "Okay, because I--using this information, I was able to create a tool that at development time, for example, as opposed to deployment time or runtime, was able to detect hat there was a mismatch."

There are so many forms of automation and so many tools that we can envision, going all the way from what I would call the most simple of the tools would be one that basically is able to read in the JSON Schema and is able to provide a user-friendly environment so that people, different stakeholders, can fill in the different descriptors for all these, and then, for example, have an Analysis tab that, you know, you say, "Analyze data," and would say, "Hey, here's the mismatches that I identified," and some warnings. Especially for those that couldn't be, like, fully automated.

Or you can have something more complex where you have something called a distribution monitor that is reading in data from the training data descriptor and is doing a chi-square test to compare distributions between the operational data and the training data to be able to detect that there's a training-survey skew.

So with that, Ipek, and before I jump into the next project, do you have any questions? And please, give me a time check.

Ipek Ozkaya: We have about 13 minutes or so, so I think we're doing fine on time, and the next set of questions are actually interesting because it heads into some of the existing tools and the frameworks and what it can do. So let me ask a couple of those.

Duane asked, "Do platforms help reduce the risk of such mismatch such as Azure ML Studio?"

Grace Lewis: So I'm not familiar with Azure, Azure ML Studio, but I'm going to assume, and you can correct me if I'm wrong, that this is an example of an AutoML tool. If that is the case, meaning that it's a tool that is going to help me with model development, I think--huh. That's a

good question. So I'm a software engineer and not a data scientist, so keep--just let's keep that in mind as I answer this question.

I think those tools are beneficial for the data scientist and for producing a--the best version possible of a model. So a lot of those tools, I believe, are more beneficial to the data scientists than to the software engineers or the operations staff. However, when those tools are integrated into end-to-end tool chains, like for example where I can say, "Okay. I'm going to produce this model. Now I'm going to wrap this model as a--" I don't know, "--a Flask app, and now I'm going to put that inside a container," or something like that, I think those definitely, definitely help with the mis--with some of the mismatches, because, for example, once--if you define that in your organization trained models are going to be handed over as, for example, containers, or models within containers, that means that you have taken the time to define a very clear API, because basically you want to be able to use that, use that container basically as a service.

So definitely there are things within tool chains that can help with that, but I would say if Azure ML Studio, I'm not familiar with that tool, is similar to, I would say, like, to more like TensorFlow or, for example DataRobot, I think those are more beneficial for the data scientist and help in creating really, really good models.

Ipek Ozkaya: And Michael did clarify that it is similar to ML model development.

Grace Lewis: Okay.

Ipek Ozkaya: So I think you're right on, like, yes, those tools have a place to help data scientists. But the overall goal of this is end-to-end system development where ML components is a part of, so it only can go so well, because if you're not talking from the same sheet of music, you will have similar mismatches.

Grace Lewis: Correct.

Ipek Ozkaya: The next question is somewhat in a similar lane, looks at--goes after data. Michael asks, "Did having democratized data sources, such as data lakes, lake houses, improve an ML mismatch frequency?

Grace Lewis: I don't know the answer to that because I don't think we got into the details of the different--at least in the interviews we didn't get into the details of how things were deployed. However, and I'm making a huge assumption here. Maybe I need a mismatch descriptor, but I'm making a huge assumption here that those types of data lakes, of what I would call data--what would I call it, like, data discipline? I think those would definitely help with mismatch because you, when you create a data lake or when you create a data store of some sort and you--and it's organizational, enterprisewide, you definitely are more careful about how you document

it, how you specify and basically providing metadata. So I think that--I don't think that data lakes by themselves reduce mismatch, but if you have policies as to what goes into a data lake, I think that definitely can reduce, and how it's documented, that can definitely reduce mismatch.

Ipek Ozkaya: And (inaudible)--

Grace Lewis: I'm getting a 10--I'm getting a 10-minute time check, (inaudible), so--

Ipek Ozkaya: Ten minute. So let's continue, right. That's--

Grace Lewis: Okay. And we--and this is very short, because again, this project literally just started. Actually, it didn't even start November 2020. It started January 2021, but it's Predicting Inference Degradation in Production ML Systems. Somewhat different from what I just talked about, but definitely also in the vein of software engineering for machine learning.

So what was the motivation for this project? It's a fact of life that has to do with--just with the way, you know, ML components are, but the inference quality of deployed ML models changes over time, and this is because of the difference between the characteristics of training data and operational data. It's just--and it's a fact of life, and we're calliting--calling it inference degradation.

The way that industry deals with inference degradation, especially large organizations, is they just do periodic retraining. So what happens is that they're not really monitoring to see if inference degradation is happening. They just evade it. They just say, "Okay. If we constantly retrain, we don't even have to worry about this," because basically you're deploying a new model every day. There are some stories about there that the large companies deployed, like, multiple times a day. That sounds, you know, crazy to me, but I believe it. But what we're trying to say here or the motivation here is that while that might be great for large organizations that have lots of data, where they can actually do this and they can do it periodically training, there are so many more organizations where that is extremely risky.

So for example, looking specifically at DoD applications, you see that there's a spectrum from enterprise AI to what they call operational AI, which is what deployed in the field, and when you have AI applications that have fast tempo and where the implications of failure is significant, I--we think that it's extremely risky to do a periodic retraining and redeployment strategy if you really don't have, like, a very, very strong infrastructure in place.

To back that up, the problem that we see is that inference degradation is really--it's hard to detect in a timely and reliable mannader--manner. So for example, if you look at this graph, let's say that this is over time and this is inference quality. So basically what this is showing is that the

inference quality of this particular model is degrading over time and this red thing right here is our threshold. Let's say this is our threshold for accepted--acceptable inference quality.

So what happens is that if I retrain too frequently, like if this is my threshold and I'm retraining right here, I'm wasting resources that I might not have, you know, training and deployment retrain model when I didn't have to do so, and also just the normal redeployment risks. Every time you try to redeploy something it's just an inherent risk. It's just like a fact of life. If I retrain too late, that means that in this period of time I was operating with a suboptimal model, and so by the time I detect it and by the time I redeploy the model I might be like way over here. So that's also a problem.

On the other hand, inference quality or drift, in this case, isn't always nice and, you know, nice curve. It sometimes could be something like this. This is an example of abrupt drift. So what happens in this particular case is that I would like to know, as the operator of this model, I want to know when this happens, because this bad of a--of such a sudden drop in inference quality might mean, "You know what? I need to take this model offline. I really can't continue operating this way because it's extremely suboptimal."

So what we're saying is that if you don't recognize inference degradation in a timely manner, there are so many bad things that can happen, especially when you're dealing, for example, with situations like this. So what this particular project is trying to do, and we're doing it right now, is that we're want to develop a set of empirically validated metrics that are good predictors of when a model's inference quality is going to degrade below a threshold due to different types of data drift.

The metrics are not only going to help us determine when a model really needs to be retrained as opposed to just relying on periodic retraining, but also tell us, like, if I know that a model takes, let's say, I don't know, 10 hours to train, then I better find out at least 10 hours before so that I have time to retrain the model before it goes into a suboptimal situation. Because this is a one-year project, we're specifically going to restrict our project to looking at CNNs, which are very commonly used in DoD applications for object recognition. If we're successful, we hope to be able to expand to other types of models.

So the approach that we're going to follow is basically we're going to create a test harness and data sets to baseline existing drift metrics, and here are some examples of existing drift metrics, BCPD, chi-square, and, again, KL divergence, and we're going to be able to--we want to test the ability of the single metrics to predict inference quality over time for models based on CNNs. So basically we have a data set that right now we're doing some data perturbations to it to introduce, you know, data drift into that and see how long it takes for these particular metrics to detect that drift has occurred.

Once we have a baseline of existing drift metrics, what we want to do is we want to start combining them and developing complex metrics to see if maybe a combination of metrics is a better thing to have because that way you can detect different types of drift. Because it could be that, for example, KL divergence is good for detecting one type of drift, but BCPD is better for detecting some other type of drift, and obviously we're going to validate these metrics with respect to the baseline.

So, and this is my final slide, Ipek, so our goal is really empirically-validated SE4ML practices. That is the goal of the group within--in which I work. So in this particular case, developing these novel metrics and also the test harness and making it available as open source is going to, we hope, is going to improve the practice of effective retraining of machine learning models.

The mismatch, the descriptors that we talked about before, we are--we envision them and we hope to continue doing so, that they serve as checklists as these types of systems are developed, and that, for example, these descriptors are starting to be used as kind of like requirements when people develop different types of parts of the system, and obviously our goal, especially for the first project, and obviously for the second, because eventually we do want to be able to integrate these metrics into a tool, is that we want to be able to do automation, and when it comes to mismatch, automation for determining, for example, if it wanted detect mismatch at runtime, which are different system components that be part of a system or automated mismatch detection, it could be design time or runtime, and if we take it further into the second project, imagine having these metrics built in to monitoring systems to be able to inform different stakeholders that it's time to retrain a model, and with that, Ipek, I am done with my presentation for today.

Hello?

Ipek Ozkaya: All right. Sorry. Hello. I'm going to take one last question, and then also end with a comment. The question comes from Mike, and the question is, "Would you envision this automation as part of the build pipeline or even earlier?

Grace Lewis: Even earlier. Definitely even earlier, because like some of the mismatch, for example, if as a data scientist I am aware of the computing resources that are going to be at the oper--in the operational environment, because that's one of the descriptors, I might make different decisions or make different tradeoffs saying, "You know what? I want to get a model that's 99 percent accurate," but it's going to be impossible to get that accuracy if that model is--if it's--if I don't have--if I'm not going to have the resources at runtime to execute that model. So definitely way, way, way before, or for example, if I know what the API is of the different components of the upstream and downstream components of the model, again, as a data scientist, I can take this into consideration when I'm building my data pipeline and my model, and it goes the other way around. If I know the language that the data scientists are using and I'm a software

engineer, I might make different decisions as to what language I choose, and also, for example, schedule decisions, depending on whether I decide to port that or not. So definitely way, way before. We see these being used both at design time and runtime.

Ipek Ozkaya: Yeah, so there's definitely an end-to-end perspective and an opportunity.

Grace Lewis: Correct.

Ipek Ozkaya: So I think in the interest of time, let's stop here. Thank you very much for updating us all on the progress on the projects and some of the what's upcoming.

Grace Lewis: My pleasure.

Ipek Ozkaya: We will definitely come back and do more of these, both from us and other members of our team, and I'm going to end with a comment that Pieter Botman put on the chat, and I apologize if I'm not doing justice to these names, and the comment is, "'I'm a software engineer, not a data scientist.' Well said, Grace. You speak for most of us." And all the data scientists, we talked, did actual say, "I'm a data scientists, not a software engineer." I don't try to put code. It's not extensible, it's not modifiable, so it's really how these different disciplines come together and define a new discipline. We are calling it AI engineering, software engineering for ML systems, and really find the challenges and bridge the gaps. That's really where all this work is headed to. So thank you very much, and I think with that, we're on to you, Shane.

Shane McGraw: Grace and Ipek, great discussion today. Thank you very much to both you for sharing your expertise, and we'd like to thank you all for attending today. Upon exiting, please hit the Like button below the video window and share the archive if you found value. Also, you can subscribe to your YouTube channel by clicking on the SEI seal in the lower right corner of the video window. Lastly, join us for our next livestream, which will be on February 10th, and our topic will be about the "SolarWinds Hack: Fallout, Recovery and Prevention," with Matt Butkovic and Art Manion, and any questions from today, feel free to send them to info@sei.cmu.edu. Have a great day, everyone. Thank you.