EPISODE 1272

[INTRODUCTION]

[00:00:00] JM: Mark Saroufim is the author of an article entitled *Machine Learning: The Great Stagnation*. Mark is a PyTorch partner engineer with Facebook AI. He spent his entire career developing machine learning and artificial intelligence products, which gives him a very interesting take on machine learning products, product engineering, and academia. Before joining Facebook to do PyTorch engineering, Mark was a machine learning engineer at Graphcore. Before that, he founded Yuri.ai. He's also published lots of other content about machine learning. In this episode, we discuss machine learning subjects and his experience developing cutting edge software.

A few announcements before we get started. One, if you like Clubhouse, subscribe to the club for Software Daily on Clubhouse. It's just Software Daily and we'll be doing some interesting Clubhouse sessions within the next few weeks. And two, if you're looking for a job, we are hiring a variety of roles. We're looking for a social media manager, we're looking for a graphic designer, and we're looking for writers. If you are interested in contributing content to Software Engineering Daily, or even if you're a podcaster, and you're curious about how to get involved, we are looking for people with interesting backgrounds who can contribute to Software Engineering Daily.

Again, mostly we're looking for social media help and design help. But if you're a writer or a podcaster, we'd also love to hear from you. You can send me an email with your resume, jeff@softwareengineeringdaily.com.

[INTERVIEW]

[00:01:40] JM: Mark, welcome to the show.

[00:01:41] MS: Thank you so much. I appreciate it. Good to be here.

[00:01:45] JM: The impetus for this show is an article you wrote called *Machine Learning: The Great Stagnation*. I'd like to dig into your ideas there. You're currently working at Facebook as an AI engineer. That's obviously an esteemed job that speaks to your level of expertise. So, I'll just start off with a fairly open-ended question. What are the most acute problems in the machine learning culture ecosystem?

[00:02:14] MS: Yeah, for sure. I mean, so I think as you alluded to, like, there's a couple, but I think a lot of it boils down to personal incentives. So, what I mean by that is, we've gotten to a point today where, there's sort of been this long-standing feeling in the community, as soon as we got to larger and larger models, that – there's a paper called *Attention Is All You Need*, referring to having attention networks, and just scaling them and getting great performance. I've also seen the people say the meme, like, money is all you need. I just feel like even though I feel – I have actually two positions on this. One, I feel this is sort of an intellectually lazy position. I'll get to that in a second.

But just like upfront, what's going on is that like, well, let's say, you yourself, work at an esteemed lab, that you have lots of students. Well, what you can do is like, you can parallelize a bunch of experiments over all of your students. So, the main algorithm in machine learning is called gradient descent. So, you're optimizing some function. But graduate student descent is a term I coined in the article that essentially refers to parallelizing work across all of your students. And then as soon as like one of them works out then great. Obviously, this sort of algorithm is accommodating for lab with with larger resources and there's feedback loops.

For example, like, let's say, your colleagues are also running lots of these kinds of experiments and they're not necessarily publishing them, while you can learn a lot from them. So overall, here's the thing, I think for a long time like just because of the title of the field, like AI or artificial intelligence, it sounds like what a lot of us do is really cutting edge. But for me, it's sort of like, even though, like I'm in the field myself, I'm okay with poking fun at my own skill set. And what I started to realize what was happening was, in fact, we weren't doing much really innovative work. What we were essentially doing was scaling stuff that we knew worked well, and basically hoping for the best.

So, there's sort of like this competition among a lot of larger tech companies, where they're like, well, we trained a trillion-parameter model, or we trained a 10 trillion parameter model, and this is easy PR. I think CPU manufacturers, for example, have been doing this for a long time. They're like, “Oh, we have a billion threads.” And I don't know how many cores. And that all sounds great and good. But really, what matters more is the the task level performance, and is it like an economical decision?

So actually, I mean, like, one thing I also want to quickly bring up is that even though there are all of these incentives, I think people end up hating on large models for a lot of wrong reasons. I think one of them is people say, “Well, these things have these insane energy requirements. We're going to destroy the planet by like training larger and larger models.” I think this is nonsense. I actually wrote an article sort of looking a newer one, looking at the energy requirements for larger language models. And it's like nowhere near anything to be fearful of, and the way they work makes it so that they're actually insanely efficient. So, there's definitely a lot to unpack here. So, let me know what's interesting, and we can dive deeper into it.

[00:05:19] JM: Well, not to take you off course so early, but given that you've looked at some energy requirements stuff, do you have any perspective on the whole Bitcoin debate, whether crypto is too energy intensive to justify?

[00:05:33] MS: Yeah, I think that's an interesting point. I'm just going to restrict this discussion to Bitcoin, because I don't know about other efforts too much. But here's the way I think of it. I think a lot of people that criticize Bitcoin, they'll often say something like, well, it uses as much energy as Sweden, for example. I think that claim is meant to provide like a certain sense of shock value, like, “Oh, wow. This is really bad.” But I think what it really comes down to is like a lot of critics of Bitcoin often aren't criticizing its energy use. What they're criticizing, sorry, they're not criticizing its large amount of energy use. What they're actually getting to is, Bitcoin shouldn't be entitled to any energy at all. I think this is sort of a dangerous point of view. Because while you end up in situations like, let's say, I'm balding, right? So, I don't particularly need a hairdryer. But if you took the total sum of all hair dryers in the world, it's also a substantial amount of energy, but it would be absolutely insane for me to go and make the claim, you know, maybe we don't need hairdryers.

So, that meme aside, just specifically looking at Bitcoin, I think what people don't understand about it, is again, they'll compare it to something like Visa, and they'll say, “Well, Visa does, I don’t know, a billion transactions a second. Bitcoin does like 30. Therefore, Bitcoin is useless.” But really what I think of it, it's sort of like a spectrum of, do you want something to be fast and centralized? Or do you want it to be slow and decentralized. So, at the base layer, Bitcoin provides something that's very slow, but very trustworthy, where you don't need to trust anyone. And at the other end of the spectrum, you have Visa where I need to trust Visa, and they can look at all my transactions. But I know that it's very fast.

Bringing this discussion back to language models, I think they have a similar characteristic, and that a lot of critics of large language models, they don't like them for two reasons. They'll say something like, “Well, this is just big tech that is destroying the incentive structure in academia.” So, that's point one. But then the second point as well, but these things are using so much energy, at some point, we're going to destroy the planet because of them. My point in my article was, like, these are mutually – you can't have both positions. Because the nice thing about language models is that you can fine tune them. You can use them as a base layer for whatever other model you're using. So, effectively, whatever work some big company does to make a language model really good, can be leveraged by thousands of engineers when it comes to fine tuning it, but also be leveraged by millions of people when it comes to pre training them.

So, even though if you look at, let's say, Google's latest switch transformer, which uses about three times the amount of energy as a plane trip from SF to New York. Again, in the paper, they try to say, “Oh, wow, that's a lot.” But when I looked at it, I was like, “Wow, that's actually very little”, because there's maybe like five or six such models being trained every year. Whereas if you look at airplanes, there's maybe like 40 million flights happening every year. And whereas a flight can only service the people that are on the plane, whereas a language model, it can potentially service millions or billions of people. So, they're actually insanely energy efficient and I think they also have some interesting implications for how to design energy grids, which is also true for Bitcoin.

[00:08:47] JM: The idea of building really, really large language models. I think of this as something that you can do much more effectively in industry than in academia. Because in industry, you just have access to a greater array of resources, a greater array of demand

specific tools, probably higher quality coworkers on balance. What does academia offer these days in the realm of machine learning research? Does academia really offer anything of significance? Is it a total, no offense, to the academics, but I'm just genuinely curious, like, is it effectively kind of a lost cause?

[00:09:31] SM: So, it's actually a complex question. So, let's dissect that a bit. I think that like part of the issue is that like, the first thing you've alluded to, is there's just some things that industry is better at. I think one of those things is scaling. Basically, execute, like basically using resources to have some sort of predetermined outcome. I think like industry just shines at it because you have sort of like these top down hierarchies, you have like large sources of funding, you have PR. You're not like arguing over grants and stuff. But there's effectively another class of academic nowadays where it's become very common for a lot of computer science professors to have a dual affiliation where they also work for a large tech company. So, it's very common to see someone like let's say, work at Berkeley or CMU, but then they're also a manager at like Microsoft or Amazon.

That arrangement, I find that a bit bizarre. I mean, from a professor's perspective, it's great, right? Because there's no downside. You can have the nice academic cushy job, sorry, you can have the freedom of an academic job, but then you can – and the prestige of it. But you can also have the reliable high income that you would generally get at a larger tech company. This is interesting, because I mean, most people at larger tech companies won't have that level of freedom. So, I think these kinds of academics have sort of gained the system for something that works really well in the short term. One thing I call them out in my article is that, if you're sort of engaging in this kind of arrangement, like you have to somewhat be honest with yourself and not call yourself a risk taker. The thing that it reminded me of quite a bit was, sometimes I would watch like something like CNBC finance, and you'd have like, some person sit and talk about, I don't know, some fears in the stock market, or actually, the stock market is underpriced or something.

But regardless of what they say, and or how accurate their predictions are, they just continue going up on television, and just like talking, because like, they're sort of like known thought leaders. But it's not their portfolio that we're measuring them on. It's just basically like how charismatic they are, or how compelling their words are. So again, when it comes to large model

training, or anything that involves scale, I don't think academia has a lot to offer anymore. But when it comes to, basically cheaper techniques, I think there's a lot of things that academics could offer, if they sort of changed their mindset of it. What I mean by that is, if your attitude as well, you know, machine learning is going to be solved by scaling. Great. You should stop being a machine learning professor. You should go get a job at a big tech company, and you should go scale these techniques.

But if you do genuinely believe that these larger techniques aren't the way to go, and that there's more efficient techniques that involve, like biasing your models a bit more, then I think being an academic is still great. And what you can do there is basically, instead of arguing with people on Twitter that like, “Well, my technique is the best.” You can, for example, produce a data set by yourself where your technique really shines, and basically tell people like, “Look, here's a data set I have. The large transfer models really suck on it. My technique seems decent, how come?” That's a very interesting discussion.

Unfortunately, though, there's another bias here, where a lot of papers are essentially judged to have the need to be state of the arc, which means that iterating on data sets isn't as important as iterating on algorithms on specific data sets. I have the strong opinion as well in the reinforcement learning field like, like everywhere. But yeah, I mean, I think academia still has a lot to offer. They just have to go about things fairly differently and not compete with big tech companies at stuff they already excel at.

[00:13:23] JM: We've talked around the ideas of your paper at this point. But let's get into it head on. So, *Machine Learning: The Great Stagnation*, your article explores a lot of ideas, what are the key points you're trying to hone in on?

[00:13:42] MS: Yes, so I think there were really two main points that I wanted to address. I think a lot of people sort of stick to the first half and the first half is, is I guess, like more bombastic and where I explore various incentive structures that make it so core ML research has somewhat stagnated. So, one of these is basically a rise in complexity where because everything needs to be state of the art, like a very reliable way of making something state of the art as you take something that used to be state of the art like a generation ago, and then you just make some random changes to it. Eventually, you can get something that works and you

can reproduce, and then you can share that. And then you get famous, you get Twitter followers, people pay more attention to you, you get like big academic appointments, where you can get more students to do the same kind of incremental research.

But I jokingly said, and actually, this is not something that – even though it's a very obvious point to a lot of people that are in machine learning, I had this joke in the article where I said, like matrix multiplication is all you need. It's true. I mean, regardless of whether using transformers or recurrent neural network or convolutional neural network, they're all in some shape or form, like a form of matrix multiplication. So, I also had this joke on Twitter like a couple of years back where I said, eventually someone is going to say like, multi-layer Perceptrons is all you need. And then lo and behold, there was like a paper by a bunch of Google people saying like, "Well, yeah, with a bunch of clever tricks, you can just use multi-layer Perceptrons and you can beat transformers."

So, all of these ideas are encapsulated by this really nice, very short, like one-page essay by Rich Sutton, called *The Bitter Lesson*, which is that like, ultimately, scale wins. And as much as we like, humans like to think our ideas are really clever. It seems that in a lot of cases, they're not really all that clever. So, that's sort of like how I think the first half ends and there's a couple of more ideas. Really, one of the main ones that I've been exploring was this idea of how many angels can dance on the head of a pin. What that means is like, I went to school in a Catholic school for 12 years, and a lot of like, basically, like, scholarly religious debates at the time and Constantinople. There were people writing like dissertations about like, how many angels can you fit on the head of a pin where the arguments were, are angels material or not? Do they have some way of becoming material? People like, wrote these long essays about it. And while all of this was happening, the Ottomans just invaded Constantinople and and took it down.

So, it's sort of a warning to not be too navel gazing, and sort of acknowledge when your ideas maybe aren't necessarily very good. So, that's the first half of the article. But then the second half was really, like, after I wrote this, I'm like, "Well, but obviously, I'm still in the field, right?" So, how come? I started being a lot more intent of the projects that I thought were very compelling that weren't necessarily very popular in the core ML community. For example, like one idea I'm really bullish on is the idea of people building more game simulators. So, Unity is a game engine, for example and you can basically build any game you want. It can involve like

language. Let's say, a robot needs to talk to like a button and then push some boxes, and then it needs to go drive a car.

So, you can have like, these very complex modalities of things that people need to do, where we could say, “Well, if something does solve this kind of environment that it feels like it's pretty clever.” But unfortunately, most machine learning engineers are not game developers. So, instead of building environments that test out interesting capabilities, what they will do instead is they'll over optimize on existing benchmarks like Atari. I think Atari has been so very done. I'm sure if you throw a data center at the problem, you can solve most Atari problems, but I just don't think that's an interesting research contribution.

That's one. And then I think others are, were more on the language space. Let's say, for example, building languages where you can differentiate stuff more easily. Let's say like, in Giulia, or languages, like in Haskell, where you can potentially have like these two, three-line descriptions for complex neural network architectures. Or if it's something like fast AI, where you know, what you're thinking about is like, what are the software patterns that make it easy to build machine learning code? So, like the Gang of Four, but applied to ML. What does that look like?

So, I think all of these are extremely interesting, but I just don't see any of these ideas, gaining much popularity outside of basically a couple of niche areas on Twitter. This is sort of why I end my article with saying is that, I've basically been reading papers a lot less. What I've been doing more is basically trying to find pockets of interesting people online, and they're just trying to figure out what they're doing and that's generally been very rewarding for me. I think that's sort of like the challenge. If your expectation is, “Well, I'm going to be in this field, where you can feel like a lot of the incentive structures are corroding”, a lot of people feel it, and then you just stay in the field and you don't ask any questions. Well, surprise, a couple years later, you're probably not going to do anything that interesting. Basically, in the short term, you can benefit from, like the hype in the media and stuff. But I think in the long term, this is not a great strategy.

It's actually also why I recommend for a lot of junior researchers, when I wrote the article, one of the most common questions I get asked was like, “Should I do a PhD?” And my question to that, sorry, like my answer to that was like, “Well, like right now you can make really good money just

basically PIP installing stuff. So, why not take advantage of that? Build a safety net for yourself, but use the spare time and the safety net, to find other stuff to be interested in and to work on and don't over optimize for something that has a lot of hype right now."

[00:19:33] JM: To take a bit of a devil's advocate approach to your title. Are we actually in a great stagnation? Because I see what what's happening is you basically have a confluence of some different elements. So, number one, almost nobody can write machine learning applications. Plenty of people can write front end web applications. The tools are very easy, much easier to conceptualize what you're doing. But you take machine learning applications, it's a lot harder. Two, the tooling for the few people who can write machine learning applications has accelerated so fast – it's developed so quickly relative to the surface area of applications that we need to build, that we can build, the opportunities that are available to us. That basically all the machine learning engineers are tied up doing application development with stuff that they know is going to work. So, it's sort of like, who needs to who needs to innovate that much right now, when we've got so much ground to cover with the preexisting technological tools? So, I would argue that we're not in a great stagnation.

[00:20:42] MS: I actually agree with you. I mean, it depends on what you define the scope of stagnation over. Really what I was saying in the article was score ML has stagnated. As in, if you think of like the core field, as in people doing sure enough bounds, or proving convergence proofs or people innovating on new architectures. I think all of that work may not be as useful as people think it is. But of course, like I think when it comes to like deploying these models, I think there are sort of almost like a wild west aspect to it, right? Let's say you want to think about experiment tracking, there are dozens of startups you want to think about. Managing pipelines, there are dozens of startups, you want to think about. Dashboarding, again, dozens of startups.

I think all of these are – these in of themselves aren't necessarily interesting academic problems. But they are interesting problems. I think there's no clear a solution to them. Even for me, this is something I struggle with at my day job. I'm just trying to think about, what are the kinds of tools I want to use? How do I make sure my work is leverageable to the maximum amount of people very easily at scale? These are all hard problems. Again, I just don't know of all that many academics that work with the stuff like that. Outside of a few exceptions, obviously, like people like Matei Zaharia, from like Databricks and stuff. They think about that all that stuff.

But there are more people, I guess that would have a hacker leaning, as opposed to a mathematician lean. I think that's again, another area where interesting stuff is happening. So, like languages is one aspect. The tooling is one aspect. The best practices for the that tooling as another aspect. All of these areas are still exploding as far as I can tell and increasing at an exponential pace, actually.

[00:22:18] JM: So, do you see this article as simply making a set of observations or are you trying to encourage a call to action?

[00:22:31] MS: Yeah. It's hard for the article to sort of be treated in a vacuum. Because after I wrote it, like, I realized that was just so common, like a lot of so many people reaching out to me on Twitter or LinkedIn DMs, saying, "Well, what should I do?" I agree, like, I've noticed the same thing. So, what should I do for my career? And as far as – I spent a lot of time, I was on vacation at the time, too. And I spent, like most of it on my laptop, answering people's questions. A bit acting like a therapist where I was going through people's personal situations and trying to think through what they were doing. But then I quickly realized that a lot of the themes are tied to generally how to manage risk in your career.

So, I had another talk at USF about it, which is like about managing career stagnation that I think is very relevant. Again, after that talk, a lot more people started reaching out to me. So, that's why now like, I manage a Discord channel called the robot overlords, which was a spin on – the name is a spin on an E-book that I wrote about robotics and ML about two years ago, where I basically help people. I guess, the what we call is like, becoming like a sovereign researcher is the main theme. What we mean by that is, how can you basically build skills that help you get like a few money, so that you can engineer that kind of life that you want to have? And not necessarily being too caught up by incentive structures that you see around you?

So, basically, it's like, how can you be very aware of the kinds of incentives in your field, be very aware of them. Gain them if you need to basically build a safety net for yourself. But once you've built that safety net, I think that's sort of when the interesting part of life starts and, basically how to leverage that, whether it's writing research independently on your blog, whether it's building a SaaS startup, whether it's writing E-books. I think these are all very interesting directions for people. It's also why I think, like COVID, sort of presented an interesting

opportunity for a lot of engineers where, in the past, you can imagine, let's say you weren't particularly charismatic. It would be hard for you to get leverage and convince a whole bunch of people to do something that you want to do at work. But like, if you're a good writer, and you write like a compelling proposal or write a compelling weakness for your product or say like, “We should fund this thing, and here's why”. I think it's very democratizing, because people will judge you for the quality of your writing and the quality of your ideas, and not necessarily for how senior you are in the company, or how highly you're paid.

I think there's a very democratizing aspect to all of these. And so yeah, definitely it was a call to action. But it wasn't a call to action to people who identify themselves very strongly as academics. In fact, I had friends that identify purely as academics who read the article and got very mad at me. They just told me like, straight up you don't know what you're talking about, this article is very sloppy, and et cetera. At first, I took the feedback to heart and I was trying to get to, what are they really saying? Was I sloppy in my thinking? But after going over the article again, and again, I was like, “No, it wasn't very sloppy, but you're just sort of threatening a worldview.” And that's always dangerous, right? In my case, I feel I'm generally pretty okay with realizing, maybe the skill set I invested a couple of years in isn't very useful. That's something I'm psychologically okay with. But I do recognize that it is a painful thing to hear.

However, it helped me attract the kind of people that have these kinds of similar thoughts, whether it's through my Discord channel or on Twitter, or just generally people that want to give me feedback on my future articles. So, even though the article did ruffle a few feathers, I think it helps me meet some of the most interesting people I've ever met in my life. It's also why I'm so bullish on writing online anyway, because, again, if I compare it to something like networking at a conference, it's very low signal. You basically talk to people because you're in a similar proximity, well, maybe because you're grabbing coffee in a similar place. Or maybe they said something you thought was compelling at a talk. But I think when it comes to sort of professional flushing out ideas, I think that the internet is a much more powerful force than online conferences.

[00:26:46] JM: As I mentioned, you work at Facebook. I don't know, if you notice, I just wrote a book about Facebook. I spent two and a half years writing a book about how Facebook engineering works. So, I have a lot of familiarity with the culture of the company, and how

unique engineering is there and how unique people think about it. I actually did not focus much of the book on artificial intelligence, or machine learning. So, I don't know a ton about how the machine learning engineering teams work there. But given that you're pretty familiar with the industry as a whole and you've been at Facebook, I know, you're only been there three months. I know you're you're fairly new, so I wouldn't ask you to let go super deep on anything. But I would love to hear in broad strokes, because I know how unique Facebook is. But I would love to know in broad strokes, how the Facebook engineering approach to machine learning and AI differs from the rest of the industry?

[00:27:41] MS: So, what I'm thinking about making the comparison, I think a lot of it will apply to just regular – like software engineering at Facebook, without ML. With ML, they are fairly similar approaches. So, I think for me, at least the big shocks when I first joined Facebook was one is the mono repo. I know people have mixed feelings about it. But I personally really like it because I feel like it unblocks me very quickly. I don't know how something works like, well, I can just go to the source code. I don't need to go argue with a PM about something and go figure out how this thing works.

So, I think the main challenge though, is that because like Facebook doesn't have a cloud offering, is that you ended up having like infrastructure internally that doesn't look like what people use externally. Generally, isn't an issue for most Facebook teams. But when you think of like, let's say, my team, which is the PyTorch team, like this is an issue because like most of our customers have – we'll use something like AWS or Azure or GCP. I mean, that's not what we use internally, so that's not what our tooling looks like. So, you ended up having to be aware of both sort of the internal and the external world because you have internal customers where you want to increase adoption of PyTorch. But you also have external customers where you want to increase adoption of PyTorch and make everyone's life a lot easier.

But generally, like I would say, just looking at the culture, there's definitely like an emphasis on speed. I think the people I've met at Facebook have been some of the hardest working people I've met. So, there's, of course, I can emphasis on getting stuff done very quickly. But then the other thing that I really, really love is that I often noticed, sometimes I would see someone who would be very active in a lot of internal groups, writing very interesting specs, and they're behind a bunch of interesting open source projects. But then maybe there's like out of college, like for

two years, and I found this amazing because like generally in other places I've been on. It seems that the more senior you are, you're supposed to be the person that orchestrates plans. And then basically more junior people or like ICs have to basically go execute. But one of the really pleasant surprise at Facebook is that it's very common to see someone who is an IC, who can be extremely high impact, and just like basically write specs that determine the outcome of what you know hundreds of people
specs that determine the outcome of what you know hundreds of people will work on. And they're not going to have a single person report to them.

So, I think that's really compelling and that there's like a big respect for talented engineers. Of course, like, they also generally have to be good writers. You have to write something that's compelling for other people to read. But the fact that that's even available to you that you can just post something interesting on a group and share is great, because other companies I've been on, generally, if you want to propose something that's a bit different, you run it by your manager, and then they run it by their manager, and then you invite a bunch of more people, then you have another kickoff meetings. So, this is all very costly. I think it's very time consuming to orchestrate all of these meetings. So, just being able to write down your ideas and say, “This is what I think is the right thing to do”, and then have people agree or disagree with you directly on the document, I think has been a profound cultural change that I think I've personally really, really loved.

[00:30:54] JM: Facebook has, obviously on one side of the framework wars, or I don't know to what extent it's a war. Again, this scenario I'm not super familiar with. But I know that Facebook has the PyTorch side of things. Google has the TensorFlow side of things, when it comes to frameworks. Is this one of these winner take all in the long-term scenarios? Because there's other technological paradigms where it's been winner take all like. React, basically, one on the front end. Kubernetes won in container orchestration. Is that the case with machine learning frameworks? Or is it going to be like TensorFlow and PyTorch for a very, very long time?

[00:31:34] MS: So, the funny thing about this point is like, let's say if you were comparing TensorFlow and PyTtorch, like five years ago, they looked very different. But I would argue like nowadays, if you're comparing something like PyTorch, to the new Keras API. They look pretty darn similar. If you consider that a lot of people are using some top-level frameworks, like let's

say stuff like hugging face, which are language agnostic, then it seems to me that in of itself, a language, isn't that much of a moat. What matters is the community around it. Basically, are people building interesting stuff around it. Is there mindshare? The people have like a feel like there's an easy path from research to production.

But I think, no one internally, I haven't seen anyone internally use the term frameworks war, because I think it's like – I mean, I don't necessarily think it's like that zero sum and then the libraries borrow so much interesting ideas from each other, that I don't think that they represent sort of a fundamentally different shift to how you do machine learning. I think the comparisons get more interesting, let's say, if you're thinking about like newer frameworks, like Jax, which are pretty different. But again, a lot of the best ideas do make it back to the top-level frameworks like TensorFlow and PyTorch.

But again, I think there's this easy bias to think of these things as being adversarial. But then like PyTorch, is supported on TGPUs. And the reason it's supported on TPUs is because customers have asked for it, and then the teams actually work fairly closely with each other. So, I think it's easy to imagine that there would be an adversarial relationship, but from what I can gather from my short time at Facebook, that just hasn't been the case yet.

[00:33:23] JM: I have done more shows in the last couple years, far more shows, frankly, about data engineering, than I have about machine learning. My sense is that there's a pretty interesting division of labor between these different departments. How do you see – and I think of it as sort of like, pre-DevOps world, maybe? Pre-DevOps, you had kind of an uncomfortable division of labor between the people who are operating systems and the people who are writing new code. I wonder to what extent today, there is that sort of divide between the data engineering infrastructure people, and the people that are building the machine learning models and applications?

[00:34:15] MS: That’s a great question. So, the short answer there is, I think, by identifying yourself as a data engineer, you're basically just signaling to the market that you want to be paid half of what machine learning is doing. The reason I say that is because, if I were to look at like –I had this meme, even as well on the stagnation article, but like, let's say data engineering is a very thankless job. If everything works fine, you're invisible. So, no one's thanking you when

everything's perfect. But when stuff breaks, it's very public. You get woken up in the middle of the lie. Maybe there's data that you can never recover, maybe you're losing money. The inferences are not working. Something happened with – and it's a very complex system. You're dealing with like a billion services in AWS. It's not like Kubernetes adds another layer of complexity on top of this stuff.

So, it's just like very, very complicated. I think because it's so complicated, people imagine, "Well, oh, but this machine learning thing, I need to know this complicated stats and stuff." But the reality of it today is that, if you want to use something like a pre trained model, all you need to do is W get a model model file, and then load this model with a framework like PyTorch, and then just call and model that forward on your inference and get an inference. So, again, I think it's one of those things where I hear this a lot as well from junior people, and they go, "Oh, but I want to be the person like writing the algorithms." One of my thoughts to that is like, "Well, it's not really that much involved in it." We talked about stuff like graduate student descent, you take something that exists, that you PIP install it, you change a couple of stuff and it works. And then you blabber on for eight pages about how it's better.

So, obviously, that's not true for all research. But a lot of research, I think in ML can be reduced to a tweet where you say, "I took this architecture. I changed x, here's the plot, it's great." So, this could be a very common format, instead of like the regular eight pages that we use quite a bit. So, I guess, my advice to data engineers is often like, I think their skill set is actually more difficult and more involved than the core ML skill set nowadays. I think, you just need to know just a bit of ML to be dangerous. And what that is, in my opinion, is learning how to load the pre trained model. I think, just functionally. But when it comes to like the math of ML, I think if you understand how gradient descent works, and how to derive like square functions, that's 80%. How to multiply matrices, like that's probably 80% of what you need to be able to read most ML work nowadays.

[00:36:46] JM: If you were to point to the most underdeveloped part of the data engineering stack, what would you point to?

[00:36:56] MS: So, that's interesting, like because I don't think of any part of the stack as being that underdeveloped. What I mean by that is that, if you want to elastically scale models, there's

stuff that exists. If you want to do distributed training, like at massive scale, like the Megatron work for Nvidia, you could use something like Deep Speed from Microsoft. If you want to do like experiment archiving, weights and biases is great. If you want to do AB testing, stuff exists. So, for me, it's not necessarily that I think of parts as being underdeveloped, it's just I think there is an overwhelming amount of choice, where it's not entirely clear what's 10x better. So, I just think it's a question of people basically settling on a bunch of best practices, and deciding, "Okay, well, this is what I think, for example, is the best way to serve models."

But to your point, as a whole scope, I think that just in general, deploying models to production, I think is underdeveloped. But not underdeveloped in the sense that stuff doesn't exist, but just like too much stuff exists and it involves just constantly – it reminds me a lot of the early web days. So, before sort of React came out on top, this was maybe, I don't know, like 2013 or something, I got very interested in web development. I was like, "Okay, I want to learn a bit more about this to deploy stuff." I remember, it felt like a new framework was coming out every week. So, I'd be like, "Okay, well, I should be using Angular." And my people were like, "No, no, you should use react." That's like, "No, no, you should use Viu." And then, "No, no, you should use Svelte." This kept happening, like almost on a weekly basis, where I felt like, "Why am I bothering to learn about any of this, if whatever technology I'm going to learn is going to be obsolete in a couple of weeks?"

So, I'm getting a very similar feeling from the ML ops space. But I think it's good in the sense that, you know, having lots of competition is good, because it means that people have lots of ideas about the best ways to do ML in production. But I would hope that over time that people can somewhat converge to a bunch of good best practices where you think, "Okay, well, I want to deploy a model that does some inferences at scale and see all of the dashboards really easily." At least that core use case will be very obviously dealt with by a couple of companies or a couple of open source projects. But I don't think we're there yet. Again, I don't think it's necessarily a bad thing. I think competition is good in this respect.

[00:39:26] JM: What are the newer AI tools? Well, in your in your article, you mentioned Hugging Face. And then you mentioned Hugging Face is one of the more transformative companies related to AI infrastructure. I think the world of AI infrastructures is pretty interesting, because I think going to market with AI tools is really, really tough for for a number of reasons.

One, they're really hard to develop. Two, it's really hard to hire a team. Three, you are kind of competing with commodity cloud providers solutions. Four, you're competing with open source projects. There are all kinds of things that make building a successful AI tooling company quite tough. What are the big category winners in the space? The companies you're excited about?

[00:40:09] MS: Let's talk a bit about Hugging Face because I've met a lot of people over the years and I would ask them this question like, “Why do you think Hugging Face is so big? How can they capture so much mindshare?” Their number of GitHub stars, like rivals, for example, TensorFlow or PyTorch. It's insane when you think about it. It's like a couple of PhD students can build something that rivals what billion – like the largest companies in history, can build. That's amazing.

So, I actually wrote an article about this called the rise of Hugging Face where I tried to explore a bit more, why I feel like Hugging Face became so big. The question is maybe a bit more convoluted than just it's a platform company, because I don't think it's a platform company. I think it's a platform company and the community company. So, what do I mean by this? Traditionally, if you look at the business models of ML, the ones that we're all familiar with, just regular from the SaaS days, I guess, is maybe you're a service. I call your API and you give me an inference. This has generally not led to a good moat in ML. There are efforts, for example, like Microsoft and Azure Cognitive Services. As far as I know, that's been a flop just because people like the fact that they can change the thing they're working with. They don't want this gigantic black box and like to be privy to this company. They want to feel like they're doing something. Then there are the consultancies. Almost all ML startups that I'm aware of that fail, because they sort of never get out of the consultancy phase, including my own startup, by the way. I was doing like an RL serve like a reinforcement learning service. for game developers. It's just really hard to get to a point where everything is self-serve that you have, like the right primitives. In your code, everyone thinks stuff is very easy. That's very hard.

Then you have what I call like the media companies, which is like open AI, for example, where you build really beautiful and amazing demos that inspire people, but you're not really selling them anything. So, effectively, you're somewhat competing against companies like the New York Times in this respect. So, when I recently saw that open AI, I decided to also become a venture fund, I wasn't surprised at all, because I think it's a great position to be in if you're a media

company, because you get basically the smart people naturally gravitating towards you. If smart people are using your API, you know who's doing what, and you can basically leverage them and fund the right people by having knowledge that may not necessarily be accessible to the open market.

Then there are platforms. So, platforms are, you know, again, what we've talked about, it's like stuff – you can you can build stuff on top of it. So, I think weights and biases is like a good example of this. But then I think Hugging Face was something different, and that I think it was a community-based company. Here's really what I mean by that. I think that for a lot of tech tech people, we have a similar hustle at this point where maybe you work at a big tech company. But you know, you're dreaming of making it big at a big tech startup. Maybe you move to Seattle, or **[inaudible 00:43:02]** you have family there, because work is there. So, you're isolated, you're probably not that religious. So, you don't have like a community you're plugging into. So, you're coding at work and then you know, to sort of get away from the cycle, you're coding after work.

I think open source communities gives like this strong sense of identity to people, where they feel like they're contributing something to something that's greater than themselves. Actually, if you look at it, I think the structure of many strong community driven open source startups reflect the structures of a lot of religions. For example, like the first Git commit, well, that's the founding document. That's like the Bible. But then, you have the main contributor, they're the evangelists. They're the ones who decide what gets added to the history. They have means. “Oh, like hugging faces, all you need. PIP is all you need.” I think this is insane. Because traditionally, if you look at SaaS startups, like they invest so much time in marketing. They'll spend money to go talk at conferences. If they want to recruit people, they'll interview like maybe 100 people to hire like one person. Whereas with Hugging Face, let's say if they want to hire someone, like, for example, like they had a high profile hire in South Vancouver, for example. And the reason they hired him was because the guy was already contributing so much to the library. So, if you want to recruit people, all you need to do is to look at your Git contributor graph, and just hire the top people, offer them a base salary. Tell them like, “Look, it's okay that you can work remotely and that's great.”

Another thing is that like, let's say, Kaggle, for a long time became the CV padding thing where initially people were like, “Well, Kaggle reflects real world data science skills. But now it seems

like almost everyone I see on LinkedIn is a Kaggle master. And I don't think that's a coincidence. Because if you tell people like, “Look, if you become a Kaggle master, that's a good way for you to get the six-figure tech job.” Then everyone's going to try to become a Kaggle master. Now, I think it's like, “Well, if you contribute a data set or a new birth architecture to Hugging Face, then that's going to help you and then you can get like a really high paying job.”

So, I think there's so many externalities to being a community driven startup, that you have a mindshare, where people want to see you succeed. They want to contribute to you, and this is I think, like something an aspect of like, reflectivity that's really become very obvious in recent years. So, reflectivity, by the way is just this idea that the more people believe in something, the more likely it is to be true. I think we've seen three strong instances of this in modern times. So, one is Tesla. People keep saying like, “Well, it's not a car company. It's an internet company. It's a tech company.” And that's what justifies the valuation. Same for Bitcoin, people keep saying it's a bubble for like 11 years, but it keeps increasing in value. Same for GameStop. Obviously, like, you know, GameStop is worth what it is, but if enough people say it's worth something, then the stock goes up. The company can sell shares. Now, they have a whole bunch of money. So now, they're actually more likely to succeed in their mission than a company that doesn't have that.

So, very similarly to Hugging Phase. I think, by virtue of people believing that this company is going to succeed, the more likely it is to succeed. You can see this, by the way, in the recent press release announcements, they had this very – I laughed so hard, and I saw this line, because the CEO literally said, something along the lines of machine learning shouldn't be in the hands of like the few big tech companies. It needs to be democratized over the entire community. I read the statement, I'm like, “What corporate company would ever issue a PR statement like that?” That was kind of punk. It was like, “Look, we're going to bring down the big tech companies. This is us. We're the community, like popular revolution, we're all behind this.” This is a lot more compelling than we unlock business value for your business. I think this is something that again, when people try to value Hugging Face by not putting the community equation in it, it's just going to seem like a bubble, when I think it's a very strong community, and maybe like a cult. I think a lot of more tech startups are going to be like this. Let's say, Replit is another example of this, where you contribute to it. The founder will like personally reach out to

you. They'll advertise your project. He'll help you get funding. So basically, giving people a sense of meaning when using product tools, I think is hugely underrated.

[00:47:23] JM: The Replit point is really interesting. I guess the the whole idea around building a brand and a cult around a project can drive so much usage and adherence and stickiness. So, as we near the end of our time, you wrote this article, this was back about a year and a half ago. What's changed since then? Do you have any new insights that if you were to rewrite *Machine Learning: The Great Stagnation* today, you would have enumerated?

[00:47:58] MS: So, for example, do some slow edits over. For example, I did briefly mentioned biotech. There are some means that I added. For example, the one on like comparing data engineers to ML engineers. But actually, I think that the article somewhat stood the test of time. That it's true. I think, that not much has changed. I still stand by the bitter like, Sutton's *Bitter Lesson* is true, as in scale beats out over complex ideas. That it does seem like as far as new algorithms are concerned, that we've run out of ideas outside of, you know, basically casting all of your problems as a supervised learning problem. I think that was a profound way in which a lot of engineers now think like, "Can I cast my problem as a supervised learning problem?" Reinforcement learning is actually one example of this. Given observations, predicted reward. Self-supervised learning is another example of this. Given surrounding tokens, predicted existing token.

So, this framework, I think, of thinking of problems as supervised learning problems, I don't think is going to go away. I don't think we've seen the full impact of it yet, because it's a very profound way of thinking about problems. I think, though, I still stand by as far as core ML goes, I still think there is a stagnation. But I think the outskirts are blowing up and increasing at an exponential pace. And so, I think people should be more okay with doing machine learning and something.

If you told me, choose anything to work on, I think, probably, for me, the most exciting thing right now happening in machine learning is what Unity is doing, which is helping more machine learning engineers becoming game developers, because I think games are the most compelling data set of all, because they just generate free data. So, I encourage more people to look into that and more people to build, online content, online communities, and just be okay. ML has

grown up. This is not a bad thing. It's been incubated in academia for decades. Some ideas were very powerful, scale and casting your problems of supervised learning. It's okay. Even if there's not more stuff – it's like doing research in classical mechanics right now. I'm sure there's some problems left. But they're not like these big problems. I think the big problems are all at the intersection of how do we use ML and I do encourage more people to just look at them instead of being a bit too attached to what their graduate student advisors were working on.

[00:50:31] JM: Cool. Well, Mark, it's been a real pleasure talking to you. Is there anything else you'd like to add in conclusion?

[00:50:28] MS: I think that's pretty much it. Thank you so much, Jeffrey. I appreciate it.

[00:50:32] JM: Wonderful.

[END]