SDS PODCAST EPISODE 691: A.I. ACCELERATORS: HARDWARE SPECIALIZED FOR DEEP LEARNING

Show Notes: http://www.superdatascience.com/691

Jon Krohn:	00:00:00	This is episode number 691 with Ron Diamant, Senior Principal Engineer at AWS. Today's episode is brought to you by Posit, the open-source data science company, by the AWS Insiders Podcast, and by WithFeeling.ai, the company bringing humanity into AI.
	00:00:22	Welcome to the SuperDataScience podcast, the most listened-to podcast in the data science industry. Each week, we bring you inspiring people and ideas to help you build a successful career in data science. I'm your host, Jon Krohn. Thanks for joining me today. And now let's make the complex simple.
	00:00:53	Welcome back to the SuperDataScience podcast. Today we've got an exceptional episode dedicated entirely to understanding the hardware we use to train machine learning models. There might not be anyone in the world better suited to taking us on this journey than the remarkable Ron Diamant. Ron works at Amazon Web Services where he is Chief Architect for their Trainium and Inferentia chips. These are "AI accelerator" chips that are designed specifically for training deep learning models and making inferences with deep learning models. Ron holds over 200 patents across a broad range of processing hardware, including security chips, compilers, and of course, AI accelerators. He's been at AWS for nearly nine years, since the acquisition of the Israeli hardware company and Annapurna Labs where he served as an engineer and project manager. He holds a Masters in Electrical Engineering from Technion, the Israel Institute of Technology. Today's episode is on the technical side, but it doesn't assume any particular hardware expertise.
	00:01:47	It's primarily targeted people who train or deploy machine learning models. But the episode might be accessible to a broader range of listeners who are curious about how computer hardware works. In this episode, Ron details CPUs versus GPUs, GPUs versus specialized AI

accelerators such as Tensor Processing Units and his own Trainium and Inferentia chips. He talks about the AI flywheel effect between ML applications and hardware innovations. He talks about the complex trade-offs he has to consider when embarking upon a multi-year chip design project. When we get to large language model scale models with billions of parameters, he talks about the various ways we can split up training and inference over our available devices. And he talks about how to get popular machine learning libraries like PyTorch and TensorFlow to interact optimally with AI accelerator chips. All right, you ready for this deeply fascinating episode? Let's go.

	00:02:44	Ron, welcome to the SuperDataScience podcast. It's awesome to have you here. Where are you calling in from today?
Ron Diamant:	00:02:50	Hey, Jon. Thanks. Thanks for having me. I'm calling from the Bay Area where I live.
Jon Krohn:	00:02:54	Nice. And so we know each other through Shruti Koparkar. So, Shruti reached out a few months ago. She's a AWS AI ML Accelerator Product Marketing Lead. It's a long job title. She must have to have like two business cards to fit that on, you line them up side by side. And so Shruti is responsible for marketing these hardware accelerators, Trainium and Inferentia chips, which I've been reading sponsored messages about in recent episodes. And these products blew my mind. I didn't know that AWS was working on them before, and I became so fascinated by them that I asked Shruti if we could have someone come on and do a hardware specific episode. So, she found you an amazing speaker on this, like you're a world expert in this. It's so great to have you on. And so in this episode, we're going to dig into all of the hardware that is driving the AI revolution today. So,

		such an exciting episode. We haven't had an episode that's deeply focused on hardware like this before.
Ron Diamant:	00:04:00	Cool. That's awesome.
Jon Krohn:	00:04:01	Yeah. So, let's start off by talking about CPUs versus like GPUs, and we're gonna get, get into things that are like GPUs in a moment. But let's start off just with GPU. So, CPUs, you can probably explain this way better than me, but they, like, they're kind of the workforce of a computer. They're like the most versatile kind of computing device on the machine. And so yeah, they do like for the regular kind of running of programs on your machine, they're doing most of the work, but GPUs allow, like they have, and again, you can explain this to the audience better than me, but it's something like massively parallel linear algebra computations originally designed for rendering graphics in like 3D video games, but that same kind of linear algebra is what we're doing when we're training, say, deep learning models or doing inference with deep learning models.
	00:04:56	And so in 2012 researchers at the University of Toronto, including Jeff Hinton, came up with an architecture called AlexNet that leveraged, so, it was a machine vision architecture that leveraged this huge ImageNet dataset. And by using GPUs to do some of the training for this model architecture, they were able to train efficiently. And, and that was like the first big super well-known deep learning architecture and GPUs allowed that to happen. So, anyway, I should let you talk. Fill us in on like your take on CPUs versus GPUs and why they're so important in AI research.
Ron Diamant:	00:05:34	Yeah, sure. So, so let's try to break it down, right. So, the way I look at CPUs it is that they're the ultimate general purpose machine. They're based on an architecture that existed for a long time called the Von Neumann

architecture. And the way that CPUs perform computations is that they bring data from an external memory to registers that are local to the CPU, to the hardware circuit that runs the computation. They read registers, perform a computation, and then write the results back to registers. And least until recent years, these registers were scalers. So, think about a single floating point number. The beauty of this architecture is that it can really do anything, and that's why CPUs are used everywhere that you can think of. The downside of this architecture is that there's a lot of overhead in decoding an instruction in order to just multiply to scalers together or something like that.

00:06:33 Now, modern CPUs in recent years improved that to some extent with vectorized instructions, but that's an incremental improvement over the fundamental architecture that is there. What GPUs do. So, GPUs indeed were at the beginning at least, were optimized for graphic or for graphics or for massively parallel workloads. And graphics was the first use use case that that kinda led for the emergence of this architecture. And the idea with the GPUs is that the fundamental concept is the same, but in instead of using a few very powerful cores, you use thousands of weaker cores. And then if you have a workload that is massively parallelizable, like metrics multiplications or like graphics where you compute a different pixel of an image, then you can parallelize the workload on all these slower or not as powerful cores and perform a computation in parallel and get an end-to-end acceleration.

00:07:36 But if I tried to tie that back to AlexNet that you mentioned, I think what was interesting about the AlexNet paper was that it was as far as I know, the first the first example of using a neural network with a very deep number of layers and with a very large data sets. So, basically the leverage of scale in order to achieve state-of-

		the-art results. In that case, it was, was for a vision benchmark or workload. And because they scaled the amount of computation needed so aggressively versus previous neural networks, they needed massive parallelization, and they achieved that with GPUs.
Jon Krohn:	00:08:22	I see. And it was a huge achievement. It blew all of the other machine vision models at that time out of the water, and it brought deep learning to everyone's attention. It was something like a 30% reduction in errors relative to the next best model at that time. And so all of a sudden academics and practitioners alike took notice, and by a year later, all of the top architectures in this machine vision competition were deep learning architectures. Whereas the year before there was just the one. So-
Ron Diamant:	00:08:53	Yeah, exactly.
Jon Krohn:	00:08:54	Yeah. All of a sudden, deep learning started to be everywhere. And ten years later, it really is, and it's, I mean, it's mind blowing what's changed over ten years. It's mind blowing was changed over the last year.
Ron Diamant:	00:09:06	Right. I think it was a mindset shift, by the way. I think that architecture allowed everyone to think that a neural network algorithm can by itself win against all the tailored algorithms that we built as a community over a decade of vision optimizations and so on. And that actually trailblazed the same exact achievements in other fields as well, beyond vision.
Jon Krohn:	00:09:33	Right. Totally. So, a term, some terms that you used in your description of CPUs versus GPUs which, you know, I think I would normally just kind of let the person get away with saying them, but I really want to understand what it means, and you can explain it to me. So, you talked about CPUs having a small number of strong processors, and GPUs is having a large number of weak

ones. So, what is it that makes a processor strong or weak? You know what I mean? Like, yeah.

Ron Diamant: 00:10:05 Yeah, definitely. So, so there's, there's a couple of of things to kind of pay attention to, but at the heart of CPUs and GPUs there's a hardware data path that we build, and that data path processes instructions, these instructions could be as simple as add or multiply two scalers together, and there's also control instructions or data movement instructions. So, think about loading a value from memory or saving a value to memory. Now that data path runs at a certain frequency. And state-of-the-art CPUs run at a frequency of somewhere around 3 GHz. And that basically means that that is speed in which we do calculations. Now, on top of that, the data path has a bunch of optimizations in order to do multiple computations at once. So, it can do multiplying in parallel to that load the next value. And in parallel to that, save the previous value and so on and so on.

00:11:01 And so that's how I would describe a CPU core. When you go to a GPU core, this main data path is much simpler. It runs at the lower frequency in order of 1 GHz or 1.5 GHz. So, we're talking about two to three x slower. We don't do as many things in parallel as we do in CPU, but that simplicity is actually beautiful because we can make this cores extremely small in hardware and then pack thousands of them in a chip and get performance that way, given that the workload is parallelizable enough.

Jon Krohn: 00:11:41 Cool. Yeah. Thank you. That makes sense to me. So, yeah, so these weaker processors, they go more slowly and they do simpler operations. So, I guess that means that like, does it mean that they're like cheaper to make, but they're also maybe smaller so you can fit more of them into like a given area? Like, for example, like why wouldn't somebody want, like, would it be theoretically possible for somebody to build a GPU full of like 3 GHz processors or something?

Ron Diamant: 00:12:17 Yeah, it's, it's a good question. So, there's, there's a couple of reasons that first of all, these very complex CPU cores, very advanced CPU cores are indeed much larger area-wise. And we do have a limitation on the amount of area that we can build into a single chip. So, that's one of the considerations there. The other consideration is that when we run these CPUs at the lower frequency, they're also smaller, as you said, but they can also run at the lower voltage operating point, which make them more power efficient. So, in some ways, building these small and simple CPUs allows you to run more power efficient and allow you to, allows you to pack more cores in a single chip

Jon Krohn: 00:13:04 Power efficient and probably heat as well. Right? That's probably a factor.

Ron Diamant: 00:13:07 Yeah. Yeah. They go hand in hand. Yep.

Jon Krohn: 00:13:09 Cool.

00:13:09 This episode is brought to you by Posit: the open-source data science company. Posit makes the best tools for data scientists who love open source. Period. No matter which language they prefer. Posit's popular RStudio IDE and enterprise products, like Posit Workbench, Connect, and Package Manager, help individuals, teams, and organizations scale R & Python development easily and securely. Produce higher-quality analysis faster with great data science tools. Visit Posit.co—that's P-O-S-I-T dot co—to learn more.

00:13:47 So, all right. So, that gives us a good sense, a really great sense of CPUs versus GPUs. So, then when we talk about these kinds of deep learning training accelerators like

GPUs, there's lots of these different kinds of accelerators out there today. And in my, like small-minded way and this was one of the things that I really had a hard time initially wrapping my head around with your Trainium and Inferentia chips, is when Shruti first came to me and I had a call with her, I was like, so these are GPUs, right?

00:14:19 And she was like, no, no. Like, they're not GPUs. But in some ways you do kind of use them in similar situations to GPUs. So, I'd love for you to like, break down for us, how is a GPU different from Trainium versus Inferentia chips, as well as how these differ from TPUs - Tensor Processing Units that we hear about, as well as IPUs, these, I think, Intelligence Processing Units. So, there's like all these different kinds of accelerators that we, you can, that we can use for deep learning, and I think more are emerging all the time. So, yeah, break down for us, like the differences.

Ron Diamant: 00:14:58 Yeah, absolutely. So, when we started looking into Inferentia and Trainium, we basically stepped back and asked ourselves, how can we build a chip that is optimized for deep learning workloads from the ground up? So, let's not take any dependency on previous architecture. Let's see how we can do it in an optimized way. And our observation was that in both CPUs and GPUs there's still an energetic overhead to performing computations. And what I mean by that is that they run instructions and whether it's with the it is with a small number of very advanced cores, or with a very large number of simpler cores, they, they fetch instruction from memory, they decode the instruction, and then they perform a relatively small computation and write the result back. So, when you look at the energetic breakdown, only about half of the energy is spent on the actual computation, the floating point multiplication and addition, that we need to do.

	00:16:00	So, we looked for ways to solve it fundamentally via a different chip architecture. And we stumbled into an, actually an old architectural concept called Systolic arrays, in which you basically, you layer what we call PEs or Processing Elements next to one another. And then you structure the computation like an assembly line. So, basically there's one, we fetch one instruction decoded, but then we perform one computation, move the data to the next processing element, perform another computation, move the data to the next processing element. And with a decently sized Systolic array, that's how the architecture is called, we can get to a point where 99% of the energy consumption is on the actual computation that we need to do, and not an instruction decoding [inaudible 00:16:55].
Jon Krohn:	00:16:55	Whoa, that sounds like an insane increase from 50% to 99%. I was expecting like, kind of like a marginal increase or something like that. That's like, it's a completely different situation.
Ron Diamant:	00:17:07	Exactly. Exactly. Exactly. And we felt that this is fundamentally something that we need to bet on. And we basically structured a matrix engine on this underlying architecture. Now, interestingly, GPUs also started incorporating similar ideas over time with what they call Tensor cores. These are also Systolic arrays, just smaller ones. We can talk about them later. But the first idea that we had with Inferentia and Trainium is that we need to leverage Systolic arrays. On top of that, the other idea that we had is that it's much easier to program a small number of large cores if you make them efficient enough compared to a very large number of small cores. So, we kinda wanted to get the best of both worlds with CPUs and GPUs, so make sure that we run at lower voltage and then very high energetic efficiency, but at the same time, make sure that we have a small number of cores so that the software programming is much easier. And we did

that as well within Inferentia and Trainium, along with all kinds of other optimizations to kinda accelerate deep learning workloads end to end, because it's not only metrics operations, it's much more than that.

Jon Krohn: 00:18:21 Wow. Very interesting. So, then, okay, so this kind of gives me a sense of the GPU versus Trainium and Inferentia difference. So, TPUs, IPUs how are those different from these other options?

Ron Diamant: 00:18:38 Yeah, let's talk about that. So, so basically in recent years, recent years were kind of golden years for a chip architects because all of a sudden this deep learning workload came in and it's extremely popular, it's exploding in the amount of compute demand that it requires, and now people get to rethink how, how we should build compute units for this workload. So, there's, there's a couple of architectures that emerged over the last couple of years. TPU is one of them, and it's very similar in the fundamental architecture to what we do with Trainium and Inferentia. It's based on a Systolic array. It has a set of supporting vector engines that we can talk about later. But I would say that TPU sits on the same camp as Inferentia and Trainium, while other architectures like IPU and a few others take a different approach where, where the main idea in that architecture is that we want to avoid reading and writing data from memory at all costs.

00:19:43 So, basically that they, they build what's called a data flow architecture where you read the inputs from the memory, but then you pass inputs from one engine to the other and try to avoid writing data to memory throughout the computation. Now, there are pros and cons with that approach. The advantage is that you can build a more cost-effective system if you can do it right, because the memory system is quite expensive. So, so you can kinda oversubscribe to some extent your memory system or

		even have a very cheap low bandwidth memory system. But the downside is, I see it is software simplicity. Because now as we program these chips, we need to make sure that we take into account how the data flows from one engine to the other, and make sure that it never gets to a point where some of the compute units are idle because they're waiting on neighboring compute units to get free or something like that. So, it adds some [inaudible 00:20:51] challenges that need to be solved.
Jon Krohn:	00:20:54	Very interesting. So, yeah, so the pro with a TPU is that it can be cost efficient because of how they've optimized to try to have data flows moving into memory and out of memory. But the con of that is that there's extra complexities. And if you, if that, that could mean that some of these processing units are sitting idle.
Ron Diamant:	00:21:22	Yeah, that's for IPUs, but yeah.
Jon Krohn:	00:21:23	Oh, for IPUs? Yeah. My bad.
Ron Diamant:	00:21:26	For, no, no. The for TPU, TPUs are not, are, are not data flow architecture. They, they're, they, they have at the heart of them, they have a very efficient matrix computation engine, but they are supported by a very high bandwidth, high-performance memory subsystem to move data in and out of this efficient matrix engine.
Jon Krohn:	00:21:51	Nice. Okay. All right. So, that is a great introduction to these kind of, these popular deep learning accelerators. With GPU, TPU, IPU, there's just like one kind of brand, at least for these kinds of chips. But with AWS's, you have these two different brands, Trainium and Inferentia, for related, like they're, they're related in some, in some way as well. So, could you break down for us you know, the similarities and differences between the Trainium and Inferentia chip, which, if it isn't obvious from the name, and actually it took me a while to figure this out myself.

So, like, when Shruti first came to me, I was comparing side by side, I had on my screen the webpage for Trainium and the webpage for Inferentia. And I was reading through all the detail, all the specs, and I was like, what are, what's different about these two chips? And then when I kind of like, it's one of those things where like when you step away from the problem, then it hit me. I was like, oh, it's in the name. Trainium is for training, Inferentia is for inference. Come on. But yeah. So, what, what are the differences between these two chips?

Ron Diamant: 00:23:02 Yeah, sure. So, so first of all, I think you're right to point out that the chips are very similar and we actually do it on purpose. We build one chip architecture, and then we tailor it for training workloads with the Trainium form factor and for inference workloads with the Inferentia form factor. So, right when we started quite a few years ago, we kind of had the realization that every successful machine learning workload will have to have at-scale inference deployment. Otherwise, you train, but you don't use the train model. So it doesn't make any sense. So, so we kind of looked through the inference problem and thought through what are the key characteristics of that problem. And there are a few items that we noticed. One is that it's what we call a scale-out problem.

00:23:59 So, with inference, you typically don't need to perform inference on a giant data center, but rather you can do inference on one device or one server. And the more successful your product is, the more customers and requests you get. But then you can deploy, you can scale out the number of servers and just deploy the service, the different requests in par. Now, another thing that is unique for inference is that we need to make sure that we can deploy them across the world, because typically you want to perform inference close to where the requests are coming from and react very quickly. You typically have a latency bound. So, we want to make sure that our inference form factor is low power and deployable in every data center in AWS. And we have tens and tens of regions where we deploy Inferentia.

00:24:53 Training, on the other hand, is a different story. It's kind of an off offline workload where you train on a cluster of many machines, many servers, and in some, in some, in some form, the data center becomes the new computer for training, for training workloads. You train on tens of thousands of devices together. So, that's what's called a scale-up workload, where the cluster is very tightly connected with one another. And we don't need to deploy close to a certain user because training grants for days and weeks and sometimes even months. So, there's no, no problem moving the deploying in some remote data center and performing the training job there and then getting the results. There's so that, that difference, by the way, scale-up versus scale-out also means that the amount of communication that we need to pack into these different form factors is different. So, with Trainium, we need to have a lot of communication capability, very high bandwidth communication capabilities in order to interconnect these chips and servers together. And with Inferentia, while we do need some chip-to-chip connectivity, and that's, that's kind of interesting, we can discuss it further. It doesn't need to be extremely high bandwidth. So, we can kinda create two form factors that are different for these, the two workloads, but leverage the same underlying architecture.

Jon Krohn: 00:26:24 Ok. So, question one is, are you able to explain to me what like form factor exactly means? And then the second, like the follow on question from that is, when you talk about the Inferentia chip not needing the same kind of high bandwidth connectivity, I hypothesize that that allows to ramp something else up.

Ron Diamant:	00:26:45	Correct? Yes. Correct. Correct. So, let's talk about both. Form factors are just packaging of the same underlying chip. So, I'll actually have a board here, maybe I can flesh it. That's one of the boards that we kinda-,
Jon Krohn:	00:27:01	Wow that's cool.
Ron Diamant:	00:27:01	Based on these chips.
Jon Krohn:	00:27:03	So, our YouTube, the like small percentage of SuperDataScience listeners who do it in the video format. Yeah, hold that up for a sec again. They get to, they get to actually see like one of these chips up on screen.
Ron Diamant:	00:27:15	So, eh, when you think about the form factor, you could take, for example, the same chip and put it on a different board with somewhat different characteristics, less input output wires and so on.
Jon Krohn:	00:27:27	Oh, right.
Ron Diamant:	00:27:28	But it's the same, same underlying chip. So, these are different form factors.
Jon Krohn:	00:27:31	Right, right, right. So, like on that, on the board that you held up there, it looks like there might be like two chips on it.
Ron Diamant:	00:27:37	Correct. That's, these are two chips. That's actually not the chip that we deploy right now. That's something that we work on internally, but but yes, that's a Form factor with two chips and a certain amount of IO.
Jon Krohn:	00:27:49	Nice, nice, nice. So, yeah, that's completely new information to me and like, super cool, because like, I kind of like, yeah, I had no idea how to conceptualize this Form factor thing. I've heard that before, but now it sounded like when I'm talking to you, it's like, yeah, there's so many questions that I've had for you today that

		I'm like, I would usually just kind of like let it like go. And I'm just like, okay, like weak compute versus strong compute. I kind of like at a high level, know, know what it means, but it's awesome to have you actually break it down for us. And so, yeah. So, this is a really cool thing too. So, with these Form factors, you have the same chip, but the way that you connect them to the surrounding board is different. Which allows you to take advantage of different, so in the case of going back to the second question that I have for you and that you're just about to answer, but I'm not letting you, is with Trainium, you had, you are specializing in having a high bandwidth connectivity so that you can have a scale-up workload supporting lots, potentially tens of thousands of these training chips working together to train a model over days, weeks, or maybe even months. And then, so with Inferentia, you can have the same chip, but they're connected to the board in a different way, which allows for-
Ron Diamant:	00:28:59	Yeah. So, Inferentia we basically reduce the amount of IO or chip-to-chip connectivity, network connectivity that is not required for inference. And that allows us to do a couple of things. The first one is to reduce costs. And that's important in large-scale deployment. The other thing is to pack more of these compute servers in a single data center rack, and then we can have more compute density within the data center. And again, at the end of the day, it all comes into enabling our customers with best-in-class performance and price performance, basically the cost per inference that they can achieve.
Jon Krohn:	00:29:40	Cool.
	00:30:24	This episode is supported by the AWS Insiders podcast: a fast-paced, entertaining and insightful look behind the scenes of cloud computing, particularly Amazon Web Services. I checked out the AWS Insiders show myself,

and enjoyed the animated interactions between seasoned AWS expert Rahul, he’s managed over 45,000 AWS instances in his career, and his counterpart Hilary, a charismatic journalist-turned-entrepreneur. Their episodes highlight the stories of challenges, breakthroughs, and cloud computing's vast potential that are shared by their remarkable guests, resulting in both a captivating and informative experience. To check them out yourself, search for AWS Insiders in your podcast player. We’ll also include a link in the show notes. My thanks to AWS Insiders for their support.

00:30:31 So, yeah, so it lowers costs, which obviously yeah, is super important to people. Especially in that inference situation, because you are going to, so when you're training a model, you might say, okay you know, we're gonna need this many hundreds of processors running for a week in order to train our gigantic large language model. But so you're like, you're kind of like, well, that's a one-time cost and I'm willing to eat it and I'm willing to pay extra to have this high bandwidth connectivity, to have that training done in one week instead of two. But when you're doing inference, all of a sudden you're like, okay, this GPU is gonna be running 24/7, all the time in order for our users to be able to make use of my LLM. And so then once you're in that situation, you're thinking about, okay, if this is running 24/7, 365 days a year, what can be done to lower costs without sacrificing performance?

Ron Diamant: 00:31:32 Exactly. But it's this Inferentia device that is going to run 24/7, not the GPU.

Jon Krohn: 00:31:37 Oh, man, yeah, exactly, yeah. That's, that's a bad habit we got to get. Yeah, exactly. Yeah, the Inferentia device. Exactly. Nice. All right, so then, well, so let me tie my flub and make it seem like I, like, there's a reason why I did it on purpose. So, I'm gonna tie it into my next question. So, some people out there might have even framed the question that I just did and be thinking about GPUs in the cloud. And so why, why was the development of these Trainium and Inferentia chips necessary? Like, what was their, out there that your customers needed, that your users needed that like the existing off-the-shelf GPUs that were designed initially for graphics processing weren't good enough for, and actually, I'm actually, as I asked the question, I'm starting to piece together that you've already kind of given us the answers, which is stuff like, like you already, you know, you talked about using the Systolic arrays to go from 50 to 99% efficiency. So, obviously that's gonna relate to a cost saving. And then similarly, when people are using a Trainium chip for training or an Inferentia chip for inference for the reasons that you already went into, like high bandwidth con connectivity or increased compute density respectively, these things are going to relate to lower costs for the customer. So, did I just answer the whole question or did I leave anything for you?

Ron Diamant: 00:32:57 Exactly. Oh, most of it, but let me, let me adjust one, a little more. But yeah, it's the underlying reason is exactly what you said. We were trying to provide our customers with the best performance and best price-performance or, so lowest cost. And when we kind of looked at deep learning, we saw a workload that is exploding in popularity. So, it was powering more and more applications, and as it was powering more applications, it was creating more compute demand. And as it was creating more compute demand, it created more investment in terms of infrastructure and research, which improves the algorithm and then powers even more applications. And there's kind of a virtuous cycle here that kinda creates a lot of momentum for AI, and I think we're seeing it in the last couple of years.

00:33:44 So, we realized that this is a workload that our customers care about a lot. And we had very good offerings for our customers based on GPU for this workload. But we kind of stepped back and thought about the AWS scale, where we are servicing millions of customers, and whether we can use that scale in order to provide even more benefits, both in terms of performance and cost for our customers. And that's when we realized that at this scale, it actually makes a lot of sense to get into a chip development project. It's not a small investment, but at our scale, it does make sense to make this investment in order to improve performance, cost efficiency, and energy efficiency even more for this workload. And then basically allow that allow a more optimized infrastructure for our customer, offer more optimized infrastructure to our customers, which in turn enables them to innovate even more. They can train larger model, they can train for lo longer time and so on. And it creates even more momentum to the cycle that they described before.

Jon Krohn: 00:34:54 Nice. So, that virtuous cycle that you were talking about there between hardware and AI I think you've called that an AI flywheel effect in other places. So, do you want to dig into that a bit more? I guess like, the idea is that when you know, as, there's more advances in AI, so starting with AlexNet now 11 years ago, you know, more and more advances in AI where like, you know institutions like AWS are able to realize, hey, like there's a huge market here and it's gonna be even bigger in the future. Like, we're gonna have more AI in more devices, data sets are gonna get even bigger and models are gonna get even bigger. And then when you create those chips, you get these kinds of efficiencies like going from 50 to 90% of the energy being spent on math operations.

00:35:45 And so then that means that a startup like mine, like Nebula, we can then use an Inferentia or a Trainium chip and we can train the same model at a fraction of the price. And so it allows more companies like mine to be able to afford to create bigger, more powerful models in a wider range of applications, which then means that AWS and other companies, hardware, other hardware designers will say, oh, look it, the market's get bigger than ever and it's gonna get even bigger. So, let's invest even more money in more chips and so on. Right?

Ron Diamant: 00:36:17 Yeah, I think that's exactly right. And I think the, what, what I try to explain to myself at least with this flywheel flyway method of modeling things is whether there is a spike in usage and that spike will kinda tame down over the next couple of years, or whether this is going to keep increasing. And I think there is a good reason to believe that AI is going to keep increasing in popularity and in compute demand over the next couple of years because of the cause of the flywheel effect. So, so basically, I was kind of walking through it before, but, but we've see in recent years that AI is used in many, many applications. So, we obviously have chatbots or AI assistants, we have voice generations, recommendation systems. We have image tagging And that's, that's just a fraction. It goes, I can go on and on, and all the different use cases.

00:37:15 So, as you get more use cases, obviously there will be more demand from the compute infrastructure to just do more AI-related compute. And because there's so much demand on the compute infrastructure, it makes sense for both academics and industry players to invest more in optimizing that compute, that workload. And that involves tooling and it involves finding different methods to do things more efficiently. It involves trying to find a new algorithms. And that investment unlocks innovation. So, one, one recent example that we've seen was RLHF or Reinforcement Learning from Human Feedback. So, we've seen that get popularized over the last couple of years and improve the quality of these models such that they can get used in even more applications. So, and so you can see why this kind of fits into itself and creates even more usage and that usage create more innovation and even more usage. So, I think we can make a decent argument why AI will keep increasing over the next years or maybe even decades. And it's worthwhile investing big time, investing with big projects in order to make it more efficient. And that's, that's what Inferentia and Trainium is all about.

Jon Krohn: 00:38:36 Really cool, and great example there with the RLHF. So, that ties into how we've talked about this on the podcast before, but this is for listeners that aren't aware of it this reinforcement learning from human feedback allowed the huge jump in capabilities from GPT-3 to GPT-3.5, and then especially to GPT-4. And it was this idea that by, with, by using user feedback, so OpenAI was able to use thumbs up, thumbs down in their platform in order to have this proprietary dataset of the, of sentiment effectively on what kinds of responses users are looking for versus not looking for. And that's what allowed GPT-4 in particular to be so intuitive feeling about anticipating the what you're looking for. And so it's like, it's wild to me with GPT-4, for example, that I won't tell, like, you know, I'll ask a question and it'll give me a response and maybe the response wasn't exactly what I was looking for, and I won't say like you know, that's wrong. This is what I'm looking for. I just, I ask like a tangential question and it seems to intuit that it was wrong. Like, it was like, oh, I'm sorry I now see that you were looking for this other thing. And so let me tell you about that. And I'm, I'm like, whoa. Like.

Ron Diamant: 00:40:07 Yeah, it's amazing. I agree.

Jon Krohn: 00:40:08 So, yeah, so really cool example, and it ties in very nicely to my next question because and kind of builds upon your point, Ron, which is that with RLHF and this kind of amazing utility that we get from GPT-4, it means that more people than ever, whether it's in their personal life or their professional life, can make good use of this large language model. And so yeah. And so, you know, that means that people who are making decisions about investing in hardware will be like, okay, there's obviously gonna be way more growth in the future, and these models are gonna require lots of compute.

00:40:49 So, yeah. So, I guess, so specifically, I, you know, I had this question for you about how, you know, I now realize it's almost like it's a really dumb question. Like, the question is, does the popularization of generative AI increase demand for high-performance chips? And obviously the answer is yes, like. So, so maybe I should get to the next question, which is a little bit more nuanced. And so you might have a more interesting answer to this one, which is I've talked on this show about techniques like Low-Rank Adaptation, so Parameter-Efficient Finetuning specifically, I talked about this in episode number 674. And so these techniques like LoRA are designed to allow us to take a big LLM and insert relatively rare matrices within the overall architecture. So, you add a tiny bit of parameters to your model, like it could be in the area of like 0.5%. So, so if you had a I'm probably gonna butcher this math off the top of my head while I'm speaking, but like, if you had a model with 10 million parameters, then 0.5% would be just 50,000.

Ron Diamant: 00:42:10 Yeah, I think yeah, I think you're right. Yeah.

Jon Krohn: 00:42:12 Yeah. So, so you add in like that kind of that or something by out of power of 10 close to that but yeah, you add this relatively small number of parameters in and then you just tune those. And so people are trying to come up with these kinds of algorithm optimizations for training in order to dramatically reduce the cost of training these LLMs. So, you take a pre-trained LLM that you can download open-source, and I've had tons of episodes about that recently stretching back to the LLaMA episode in episode number 670 and a bunch of subsequent Friday episodes. So, you get these big open-source LLMs and then you insert this very small number of new matrices to train. And then yeah, we use this Low-Rank Adaptation approach to very parameter-efficiently train these models.

00:43:05 And there's other kinds of optimizations like Quantization. So, using you know, instead of using full precision floating point numbers, we use, you know, half precision or smaller. So, I guess do you think that like these kinds of optimizations could mean that there will be like a somewhat smaller demand at some point in the future for hardware? Or is it basically that like, you know, over, because we're seeing such enormous improvements in capability by scaling up models to be very large, that these kinds of algorithmic optimizations like LoRA and Quantization will just somewhat reduce, but on like, but not substantially reduce this explosive demand for hardware?

Ron Diamant: 00:43:54 Yeah, so I, first of all, I, I'd say that since the release of the LLaMA model and weights what was, what's happening in the, in the community is amazing. The amount of innovation and the pace of innovation that we're seeing is absolutely fantastic. And all these ideas like LoRA and QLoRA are coming into the kind of the front stage and improving the efficiency of these models. That's fantastic. I think that there will be, there, there will be some use cases where you can run efficiently on the end consumer hardware and it'll be efficient enough. And, and these cases will probably be the cases where the model can be small enough and the rate of interaction with the model is relatively slow, and then you can run it on a mobile device on a laptop, and it's still fast enough. When you go to mass deployments when you need to, where you need to service millions of requests a day or sometimes even more than that, I think the physics of generative AI models will, will come into play and eventually these models are extremely, extremely memory hungry. So, it'll be more efficient to do them on the cloud with dedicated hardware that is efficient for that.

00:45:13 So, that's one case where I think cloud backing will still, will still be useful. And on top of that as far as I can tell right now, there will be small models that are, that sacrifice a bit of the quality of the results may maybe very minimally for performance and for cost, but there will be cases where we want to achieve the maximum highest quality results. And then you want to deploy a 200-billion parameter model or something like that. And that's hard to do on consumer-grade hardware.

Jon Krohn: 00:46:28 The future of AI shouldn't be just about productivity. An AI agent with a capacity to grow alongside you long-term could become a companion that supports your emotional well-being. Paradot, an AI companion app developed by WithFeeling AI, reimagines the way humans interact with AI today. Using their proprietary Large Language Models, Paradot A.I. agents store your likes and dislikes in a long-term memory system, enabling them to recall important details about you and incorporate those details into dialog with you without LLMs' typical context-window limitations. Explore what the future of human-A.I. interactions could be like this very day by downloading the Paradot app via the Apple App Store or Google Play, or by visiting paradot.ai on the web.

00:46:31 Totally, yeah, all those answers made perfect sense and they were in line with that, with what I was expecting, but it was nice to hear like that level of detail from you. And then you also, you hit on a really great point there, which is that like things like LLaMA, Vicuña, Alpaca, GPT4All-J, DALL·E 2.0, like all these kinds of open-source architectures that we can download and then use LoRA to fine tune to our needs. That in and of itself, even though you're like, okay with LoRA with training, we're like, okay, there's not that many parameters that we need to train, but you then have, like you're saying, this explosion in people being like, oh, I'd love to do it, which increases demand then yes, your point right there, which I completely was absent on, but at inference time, it doesn't matter that I had a small amount of weights during training.

00:47:12 So, even if I, so like, if I was right about that math going from the 10 million down to the 50,000 I still at inference time, I can't do inference with the 50,000, I need to use the full 10 million. So, so yeah, so at inference time, you know, you're still, you're still gonna have all that demand and then hopefully your application takes off and hopefully lots of these, you know, more and more of these applications are gonna take off. So, there's gonna be more and more demand for the Inferentia type chips. And then just as you say, a chip like Trainium is gonna be in demand because yeah, people, you know, we're constantly pushing the state of the art with these huge scaled-up models with hundreds of billions of parameters. And those are the most powerful general-purpose ones. So, that'll continue to happen and people will continue to use those as, you know, a starting point for distillation or proving. But yeah. Cool.

00:48:03 All right, so that's wild. So, we just got through, like, I had five different kind of sections of topics to go over, and we just got through the first one. That's so awesome. And I love it. Like those, you know, I had it as the first section for a reason. I thought it was like, you know, some really exciting stuff. I mean, we still have obviously lots of other really exciting topics, but yeah, it was awesome to be able to dig with you Ron, into so much of these technical things related to hardware. So, my second topic area for our listeners is, as a chip designer, what is your long-term vision when you architect a new chip? So, when you talk about something like a hundred million dollar project, obviously that is happening over an extended period of time, so you have to make predictions about what people are going to need many years from now in their devices. How do you go about doing that kind of long-term design?

Ron Diamant: 00:49:00 Yeah, that's, that's a, that's a tough one. And it's a, it's one of the areas where I think experience actually matters, where you kind of see chips that succeeded and chips that didn't succeeded as much. You kind of get a good hunch of where to focus on. And just before I answer, I would also add that the way that we build chips, the first, the first copy of the chip costs about 20 million to manufacture, but then every, every other chip, every other copy of the chip after that costs tens of dollars or, or maybe a hundred of dollars. So, that's, that's a very unique setup, especially when, when described to software folks, because the cost of a mistake is brutal. If you need to fix a mistake and remanufacture the chip, that's an extra 20 20 million right there.

00:49:53 So, you really need to be confident about two things: about the specifications of the chip that you generated, and also about the level of testing that you did so that there's, there won't be a silly bag that causes us to remanufacture the chip. So, with that kind of preface, I think what you said is exactly right when building chips is a, of decent complexity, is a process that takes about two years time. It's not a quick process. And then we build servers and deploy them into the data center, and people use them for quite a few years, three, four, or five years. So, when we start a new chip project, we try to anticipate what's going to happen over the next five, six, or seven years. And that's extremely hard to do for almost any workload, but especially in AI where things change in a super rapid pace, right?

00:50:48 And it's, it's very tempting to try to guess what changes will come in three or four years. But I think it's, I honestly think it's impossible. I think no one knows where AI will evolve into in the next five years or so. So, one of, or two techniques that we try to deploy or that I at least try to deploy every time I go and build a new chip is the first one is to look at the workload and try not to target the current popular workload, but rather try to decompose it into primitives, generalize the primitives, and build a machine, a chip that is very efficient in offloading or accelerating these different primitives and let our customers kind of play Lego with these primitives and create the new workload five years out from now. That's one the ability to break down to primitives and try to generalize them.

00:51:48 The second one is something that I personally took from Charlie Munger, actually. So, I love following Charlie Munger. I find him supervised and yes, this set of mental models that he uses for a bunch of decision-making, the decision points that you need to make. And one of them, I especially like, it's a, it's a mental model called inversion. And the idea is that whether on the, on the professional life, by the way, or on the personal life, if you have a question that is super hard to answer, try to invert the question and answer that inverted question. In many, in many cases, it's much easier to answer the inverted question, and then it guides you through how to deal with the more complex question.

00:52:37 So, if I kind of try to tie that back to what you asked, we're trying to build a chip that will last for five, seven years or something like that. And the question that we're asking ourselves is, what's going to change over the next five years? And we need to prepare the chip for, what kind of future-looking features should we build in? And I would argue that if we invert the question, it can give us a lot of clarity. So, what's not going to change in AI over the next five to seven years? That's actually a simple question to answer. So-

Jon Krohn:	00:53:10	Yeah, like linear algebra operations.
Ron Diamant:	00:53:11	Linear algebra operations are here to stay. Different data types are here to stay. Like we saw FP32 and bfloat16. I don't, and it sounds intuitive now, but when we just started, folks were arguing that we need to build chips that only do int8 and nothing else, int8 math and nothing else.
	00:53:32	And we were saying, no, that's, that's it too significant of a bet. Right now people are using all kinds of different data types. So, we figured we need to allow our customers with this flexibility to choose the data type that is optimal for their use case. And by the way, even more intuitive things like that. So, there's no chance in the world in five years people won't to want high performance. Everyone wants more speed, right? Everyone wants lower cost. So, we try to design the chips around the things that we know are going to be needed, and then as I said before, try to generalize the primitives and not kinda overfit if I use ML-lingo to one specific workload.
Jon Krohn:	00:54:14	Nice. That's a great example. Yeah. And so Charlie Munger, for a bit of context for listeners who haven't heard of him, he is Warren Buffett's business partner at Berkshire Hathaway. And so probably most people have heard of Warren Buffett. A few years ago he was the wealthiest person on the planet largely on the back of his big holding company, Berkshire Hathaway. And Charlie Munger is Warren's right-hand man, and I think he's a hundred years old now, if I-

Ron Diamant:	00:54:38	Something like 98 or a hundred something in these lines.
Jon Krohn:	00:54:42	Yeah. And-
Ron Diamant:	00:54:42	Still sharp as a knife.
Jon Krohn:	00:54:44	Yeah, that's what they say. People like friends of mine were posting from like the most recent Berkshire Hathaway annual general shareholders meeting, photos like with Charlie Munger and just saying, yeah, yeah, how sharp he is. And yeah, he's like, he's famous for like how much he reads and how widely he reads. Cool. All right. Yeah, so his inversion approach yeah, makes it easier to answer difficult questions. And you gave a perfect example there where like, yeah, when you're trying to figure out what will change, answer the question, what will not change, and you're, you're absolutely right. That did make it easier to understand. And I'm gonna try to remember to apply this in my personal and professional life. I hadn't heard it before. So, thank you very much for that. So, yeah, I mean, I guess there's like a lot of different aspects to chip design, lots of different things that you need to weigh against each other.
	00:55:39	So Serg Masís, who is our researcher for the show and who has been on the show in the past so he has an awesome episode on explainable AI, that was episode number 539. And yeah, so Serg had some additional questions around terms that I don't know anything about. I mean, so one of them is memory, I know that, but then there's also like accelerator interconnect, dynamic execution, collective communications. So, like how, yeah, I mean, how, how do you weigh all these different things? And I don't know, maybe we don't want to get too down in the weeds on this, but yeah, if you think there's something else kind of interesting additional to say related to design, that'd be cool.

Ron Diamant:	00:56:22	Sure. Yeah. So, I think if we can kind of touch briefly on these different topics. So, accelerator interconnect or sometimes called the chip-to-chip interconnect is how we interconnect these chips with one another so that they can exchange data quickly with one another. Back in the days, it used to be gradients, in today's workloads with gigantic models, it actually tends to be a temporal state or activations that move between the different chips. It even at some for, for the very large models, we actually even perform one giant matrix multiply, a feed-forward network on multiple chips, and then we kind of take a matrix, kind of partition it column-wise or row-wise, spray it across chips and then assemble the result back. So, that's then via chip-to-chip interconnect So, just high speed IO at the end of the day.
	00:57:18	Collective communication is the set of primitives that is used to transfer data in an orchestrated manner between the chips over this chip-to-chip interconnect. And I think you touched on dynamic execution as well. Dynamic execution is something that is challenging to some chips, especially highly parallelizable chips. And the idea is that if we kinda use software terminology as an example, if you have an IF condition in the middle of the code, then you get dynamic execution based on the data that is being processed. You decide whether to execute one piece of the code or the other. Now, in CPU, it's actually, it's, it's not very, very hard to do. It's just a branch instruction. That means that the next instruction is fetched from a different location. In highly parallelizable machines, that's actually harder to do because think about many cores processing data at the same time. And then some of them do need to branch and the others don't need to branch, and all of a sudden they go out of sync from one another. So, we need to build some mechanisms in order to deal with that efficiently.

Jon Krohn:	00:58:30	Very cool. Yeah, so thank you for those introductions to those concepts. And yeah, so I guess like as you're weighing all these different kinds of things you're kind of using this inversion approach to say, okay, you know, we know that people are gonna want these to be cheaper, but, but more powerful. And so yeah, how can we engineer all these different aspects together while delivering those outcomes?
Ron Diamant:	00:58:55	Right. Yes. It's, it's exactly, I forgot about the how, how do we weigh these different things together? So, the first thing that we always try to do, and that's actually, I find it to be critical, is to try to simulate the workload on the hardware as we build the primitives. And that allows you, us to kind of think top down rather than bottoms up. So, instead of kind of trying to build all kinds of engines, and then at the very end look at how we can kind of force them to run the workloads that customers care about, we start with a couple of workloads, try to make them different than one another to kind of have a large span of different usages of the hardware and try to simulate it on the hardware we're thinking about building and trying to see how things will play out.
	00:59:41	So, this allows us to get a better mental model on how to partition the device and what capabilities the chip needs to have. And then the next thing after that is, so we talked about the inversion approach. So, basically if we talk about dynamic execution, are we sure the dynamic execution will be needed, right? And I don't, I know, I'm not sure to be honest, but or I wasn't sure four years ago when we start, when we built Trainium 1. But if we look, but if we invert this question, are we sure that we won't need inversion? The answer is absolutely not. We we're not certain they won't need a, sorry dynamic execution. The answer is not, we're not certain that it won't be needed. So, we build this capability and we build enough flexibility into the hardware in order to deal with that.

	01:00:40	And maybe one, one more sentence on that. What we do for these kind of these kind of problems or requirements is actually kind of interesting. On the one hand, we want the chip and the hardware to be as optimized as possible, right? To perform a computation at wire speed, to do, to do it at very low energy consumption. And so on. On the other hand, we want it to be very flexible. So, we need to balance these two, and it's, it's kind of a tough task. And what we end up doing typically is that we take the main data path, the part that needs to be the number crunching, and we highly optimize it by hand, we make it as efficient as it can be. But then we attach a small microprocessor, think about a small CPU, to that data path, and that processor can make all kinds of control decision, like an IF condition that we, that we said before, or like deciding how data moves from one data path stage to the other in order to get all the flexibility that we can. And in some way, we get both the flexibility and the efficiency by combining things this way.
Jon Krohn:	01:01:52	Wow, that's wild. So, like, so flexibility means that the device can handle like a wider range of operations, for example. Is that kind of what flexibility means?
Ron Diamant:	01:02:04	100% right. And, and I have an interesting example on that one. So, when we built our first deep learning chip Inferentia 1 that was before transformer existed. We started building this chip a year or two before that. So, we, two years before that. So, the most popular workload back in the day was, was resonant. You could also everyone knew about LSTMs back then. So, and there were a couple of hardware vendors that were building Convolutional Network Optimized Machines, because that was the highly popular workload. And we actually got recommendations to do the same. Basically Convolution Neural Network is the future. Go build only that. And again, we use the same, the same techniques that I, that I told you about before to not do that. Basically, we don't,

we don't have conviction that CNN is the future. Let's not tie ourself to that to that workload only.

01:03:03 So, we decomposed things to the primitives. And you can think about one of the primitives of a neural network is a non-linear function, what we call activation. And back then ReLU was starting, actually ReLU, sigmoid and TanH was very, were very popular, these three. So, it was, it would've been very simple to just hardcode these three, a non-linear function into the device. And whatever new non-linearity comes our way, we cannot support. But again, for the same exact reason, and exactly what you said before, we knew that we need to make sure that we can support new instructions and operators that we don't even know about today.

01:03:46 So, that's how we try to decompose things. We try to make the non-linear engine a very flexible engine that can handle any scale or type of processing, including all non-linearities. And as we deployed the chip transformer ate the world came to be and became super popular. And for the, for the one, two years after our deployment transformer was actually our number one workload, and it was a workload that we didn't even know about when we built the chip. So, that actually is a good data point that speaks to the flexibility, the right flexibility point that we built into the chip.

Jon Krohn: 01:04:24 Such a great example. I love that. And it makes it so crystal clear to me what flexibility is. So, yeah, so instead of building a convolutional neural network-specific architecture, you wanted to be able to support a wider range of operations, which ended up being critical when transformers started eating the world. Awesome. All right. So, transformers require, well, maybe don't require, but the best use of transformers we've seen today involves a lot of scale. So, you're having a large number of transformers in one architecture as well as using a lot of data then to train that architecture following things like the Chinchilla scaling laws that I've talked about previously in episode number 676. And so scale is really important, model scale, data scale. So, when we scale as you mentioned earlier, we might use tens of thousands of accelerators like Trainium accelerators, for example, for training our model. So, obviously we're distributing compute across many different machines, many different resources. So, how does this work like maybe like in a bit of technical detail. I know, I know that there's like different kinds of parallelism that you can probably speak to in a lot more detail than I could, like data parallelism, model parallelism, tensor parallelism, and pipeline parallelism.

Ron Diamant: 01:05:50 Right, yes. So, so yes, scaling a model is, is actually an art a system optimization art these days. There are a couple of different ways of doing it, but one class of ways to scale out a model is called 3D parallelism. And 3D parallelism involves both data and model parallelism, and actually two forms of model parallelism. So, maybe I can kind of try to describe them one by one. Data parallelism is the simplest form of scale-up. You basically, if your model is small enough to fit on a single Trainium accelerator, which in the LLM cases it's not true, but let's start there. So, if it's small enough to fit in one, we basically train the entire model or we do the, sorry, we do the forward and backward path of the model on each one of the accelerators, and then each training device computes the gradients, and then they exchange a gradients between them to average and get to a total gradient update value.

01:06:56 At that point, we run the optimizer, which updates the values with the new values and all accelerators get the new values and perform forward-backward path again until we converge at some point. So, if we look, let's say four or five years back, that was the most popular form of parallelism, data parallelism. Now, when the models become larger than the memory of a single accelerator, this is no longer an option because in order to do what I just described, we need to fit the model on a single accelerator. So, the first thing that we do is typically called tensor parallelism. And tensor parallelism basically takes every matrix vector, matrix-matrix operation, and shards that operation either on the row dimension or on the column dimension. And then if we have, let's call it 16 Trainium devices that are doing the same computation with a tensor Trainium, degree of 16 per the number of accelerators, each one is going to get one over 16 of the weights of each layer, perform a portion of the computation, and then we use the chip-to-chip interconnect that we talked about before in order to assemble the result from that matrix-matrix multiplication, and spread it across all the chips.

01:08:16 So, by doing that, we basically effectively increase the HBM or the memory that is attached to one accelerator, to the total HBM of 16 accelerators. So, think about instead of 32GB, we can get 512GB, and we can do it with very minimal overheads. You can, the amount of overheads the tender parallelism induces, if you do it correctly, if you kinda overlap between communication and computation, is in the order of 5% or 7%. So, we can do it extremely efficiently. That's a first form of parallelism. So, now we have 512GB of memory with 16 of these devices. But what happens if the model is gigantic? If it's 1 trillion parameters, then it doesn't even fit on 512GB. And when that happens, we actually, and actually it doesn't even fit for smaller models than that for 200 billion parameters, it already doesn't fit because we need to store both the weights and the grade.

01:09:17 So, so at that time, that's where we introduced what's called pipeline parallelism. And what that means is basically that we split the layers on groups of 16

Trainium devices that are doing tensor parallelism. So, let's say 16 Trainium devices will do the first 10 layers, the other ones will do the next 10 layers, and so on and so on. And then we make sure to pass the data between each groups of accelerators, but we need to do it on the forward path and the backward path while keeping the weights and gradients local for each group of 16, which makes for certain bubbles of idleness for the accelerators that we need to solve in some way. And there are all kinds of techniques in order to solve that. But I'll, I'll pause here to give you a chance to follow up.

Jon Krohn:	01:10:03	Nice. Yeah, that's great. I think I followed all of it. So, data parallelism is the simplest, I think conceptually where you can fit a model onto a single device. So, then you can just you know, if you want to have 16 accelerators each with a model loaded on it you simply flow different data points into each of those 16 devices. And then you can accumulate your gradients across those different devices and allow you to just end up with one model in the end. We need tensor parallelism when the model doesn't fit onto a single device. And this allows us to scale up to 512GB of weights across different accelerators, but sometimes even that isn't enough. And then we need to go to a pipeline parallelism where in that case we're, we're additionally breaking up the training into different layers?
Ron Diamant:	01:11:10	Correct, yes.
Jon Krohn:	01:11:11	Yeah. Yeah. So, just tackling some of the layers at a time, because we can't even infer over the whole model. Yeah, yeah. We can't do inference over all of all, all of the layers, even in just a part of the model.
Ron Diamant:	01:11:27	Right, yes. And there are others. That's only one class of parallelism by the way. There's another class of parallelism that stemmed from a paper called a ZeRO - Zero Redundancy Optimizer. And there are a lot of

similarities, but the main difference is that instead of doing pipeline parallelism, as I described before, where we split the model on the layer dimension between different groups of accelerators, instead of doing that, we still, we still shard the model on the, on the layer dimension, but we read the weights remotely from a different set of accelerators, but run the entire model or on a single set of devices. So, slightly different, more, more network demanding, but similar in approach in some way.

Jon Krohn: 01:12:20 Very interesting. So, yeah, so super cool. So, this gives us a, yeah, a good sense of how models can be split up depending on their size. From my research notes from Serge he also pointed out to me that there's something called the Trainium Ultra Cluster. And so this allows us to connect over 10,000 training devices in a single cluster. So, what kinds of challenges come up when we're dealing with that kind of enormous scale, and what kinds of models are making use of that enormous scale?

Ron Diamant: 01:12:56 That's a fantastic question. Yes. So, so first of all it was up to 10,000 Trainium devices. It's up to 30,000 Trainium devices today. So, we keep increasing it. Let's start with the type of models that use it. When, when you're training a giant LLM when, when talking about 200 billion parameters, or maybe even more, maybe more towards the 1 trillion parameters you have to train on a gigantic number of devices. We're talking about tens of thousands of devices in order for your training to finish in a reasonable amount of time. What we don't want to do is train for six months, and then we cannot iterate. We just wait for training to complete. So, that's the reason that we build these ultra clusters. We want our customers to be able to get this really extremely powerful, powerful high-performance computing cluster, and train in reasonable time complete and move to the next model or the next iteration of the model.

01:14:01 And as you said, this comes with some challenges, especially the interconnect between, between devices. That's the reason we actually built an Amazon internal network protocol, called Scalable Reliable Datagram, which allows us to automatically do traffic steering. So, when we identify hotspots within the fabric, let's say a switch that got congested or anything like that, we automatically can kind of bypass the congested point and make sure that the devices still get maximum bandwidth between them. So, so first of all, we deploy these unique algorithms in order to automatically recover from failures or, and from hotspot. And the second thing that we do is that we build these infrastructure with an extremely optimized network that has zero oversubscription. So, basically you can, you can send a packet from one training device to the other with a guarantee that you'll get the maximum bandwidth between them. And no matter what other traffic is running in the cluster, you'll still get the maximum better. And that's super critical for these workers.

Jon Krohn: 01:15:13 Wow, yeah, it's fascinating to be involved in that space. So, this, you've given us a great sense of the hardware side of things in order to be able to have these extremely large models trained efficiently. Going a little bit more to the software side the kinds of popular machine learning frameworks that we use to build these models are PyTorch and TensorFlow. So, in order to be able to use these popular, these popular deep learning libraries on your hardware, on Inferentia and Trainium chips, we need like special considerations. Like, you know, like you, you spend all this time configuring aspects of the chips and the boards and the way that everything clusters together so that it's, it's, you know, it all can be optimal, but in order for our software, our machine learning frameworks to interface with that optimally, AWS has created the Neuron SDK, software development kit. So, yeah, I mean, maybe you can just tell us a bit more about that and why this is essential for making the most of your specialized hardware accelerators.

Ron Diamant: 01:16:27 Awesome. Yes. So, so Neuron is absolutely critical for making Trainium and Inferentia successful. And, and the reason it's, it's kind of funny, I kinda, I'm trying to think about the answer and one thing keeps coming to my mind. We have a, the original Annapurna CTO, who's now a VP and distinguished engineer in Amazon keeps telling us, I swear I hear it once a week, or something like that, don't be chipheads. And what he means by that is don't think only about the chip, but rather how people will consume the chip, how the software folks that need to run their workload will experience that chip. So, at the very first day of building Trainium and Inferentia, we started thinking about the software integration into PyTorch, TensorFlow, and now JAXs also, and trying to make the experience for our users as seamless as possible.

01:17:22 So, one thing that again, is very tempting to do and have seen folks do it, is to create what's called a model ZOO. So, basically you have a set of optimized models that you implemented for your hardware, and customers can choose from these models and run them on your hardware. But in my, in my opinion, that's, that's absolutely the wrong way to go, because customers don't want to be limited to a set of 20 models. They want to allow their teams to innovate and come with their own models and their own operators and their own extensions, and still have it run as effectively as possible on the heart.

01:18:02 So, for that, we have a couple of things that we built with the Neuron stack. The first is the Neuron Compiler. And the Neuron Compiler is quite a significant and a sophisticated piece of software that basically does two things. It does lowering an optimization. So it starts by capturing a graph from the framework level, whether it's PyTorch or anything else. And then it encodes the computation graph that you described, whether it's transformer or anything else, into what we call an intermediate representation. So, that's sort of a file that describes the entire computation. And each and the, and the file is is constructed as a graph with nodes that perform computation and dependencies or edges that go from the output of one operator to the input of another one. After doing that, we do something that is basically the heart of the neuron compiler, which is what we call the Tensorizer sometime. We sometimes call it also the middle end of the compiler. So, we basically take each and every operator and describe it in loop formats.

01:19:21 So, for example, if we have a convolution or a matrix multiplication, we can describe it as a nested set of loops, where at the very innermost loop, we get some scale operations that do a multiplication and addition. And that's, that's how you can describe matrix multiplication. And then once we have things described in this way, we can apply a set of, of very significant optimizations that will target the hardware better. So, just to give you a sense of things that we do, we would fuse loops together and by fusing loops together, we actually, it's kind of hard to envision, but we actually minimize data movement and memory footprint. So, let's try to kinda visualize that. If you have a set of loops doing a matrix multiplication, and then a set of loops doing a nonlinearity, a GELU activation on the result, when the make the, when the loops are not fused together, you'll do the entire matrix multiplication, write the result to some memory location, and then enter the second loop, which reads everything, and then, and then performs the non-linearity, the GELU activation.

01:20:34 If we fuse the loops together, we can basically get to a point where every single tile of matrix multiplied that gets completed immediately goes through the non-linearity, and only then gets written to memory. And by doing that, we reduce what we call the working set, the amount of memory that we need to keep cached at the time. And by, by doing that, we keep the data local, we reduce the amount of bandwidth that we need from the memory and improve performance overall. We kind of try to speed things up to not describe the entire stack, but at the very end of this Tensorizer stage, we take the inner-most loops, and we kind of collapse them into hardware intrinsics, basically instructions that can be executed by the hardware with the single instruction.

01:21:18 So, if I kind of try to tie to the beginning of our conversation, we had the large systolic array that can do metrics multiplications very efficiently. So, if we, so we take, let's say that the matrix, the systolic array can handle 128 by 128 matrix multiplies. So, we create an inner tile from the loop with 128 by 128 dimension, and we lower that to a single hardware instruction, and we eventually perform scheduling and allocation to maximize communication and computation at the same time, and target the hardware in the most efficient way. I'll say one more thing, and then I'll, I'll pause again. So, that's one, that's one kind of theme of making it as easy as possible for customers to target our hardware. At the end of the day, they need to add two lines of code that basically instruct the compiler, what are the edges of the graph that you want to target the device that you want to compile, and everything else happens behind the scenes.

01:22:20 The other key capability that we bake into the hardware and the software stack is what we call custom operators. You touched on [inaudible 01:22:29] before and about the innovations of doing 4-bit inference and so on. That's exactly the reason that we built this mechanism that we call custom operators. At its heart, it's just a set of deeply embedded vector processors that are inside our training

		compute units, and they have access to the other, the other engines in the caches. And the nice thing is that they can run your C code without our involvement at all. So, you can compile the entire graph, but then let's say you have this magic operator that you want to deploy, and you don't even want to share it with us. So, it can be int4 to int8 dequantization, it could be zero decompression, or it can be anything like that. So, you can build this operator and target our hardware and run it in conjunction with the rest of the graph that our software stack compiled. So, these two together are very powerful in my opinion.
Jon Krohn:	01:23:32	Nice. Yes. Such an amazing explanation, and it's mind-blowing for me, the depth of knowledge that you have, in particular about deep learning operations. I mean, I guess to some extent you'd, you'd have to, given what I now know about your role, but coming into this interview, I had no idea that as a chip designer, you'd have to know this level of detail around exactly the operations that are happening in a given deep learning model. So, yeah, super cool to hear all that. And yeah, I think another cool thing about this Neuron SDK that you've just been describing is that it, it does also involve collaboration with community, especially with the PyTorch team. So, I don't know if there's anything about that that you want to specifically mention or if I, that kind of just covers it.
Ron Diamant:	01:24:16	No, it's absolutely true. We love collaborating with the PyTorch team. We actually collaborate between us, Meta, with PyTorch, and Google with XLA. So, the three companies are collaborating quite tightly with one another. And we one of the things that we try to do together is to make sure that the entire stack is integrated well together. So, I didn't talk about that, but the front end to our compiler is called XLA. And XLA is an open consortium these days where our attempt or our goal is to make sure that all hardware devices have the

same entry point, so that integrating different hardware devices from any framework will be much easier. So, that's one. The second one is that we try to contribute back to PyTorch, especially with distribution techniques just like we described before, such that these distribution techniques just like ZeRO that I, that I mentioned with the library called FSDP in PyTorch, we contribute back to that library such that the scale-out will happen as efficiently as possible on the Trainium and in Inferentia targets with minimum effort required from our customers or, or actually the goal is for it to be zero effort from our customers.

Jon Krohn: 01:25:42 Nice. Very cool. I actually wasn't aware of until you mentioned it just now, and I just did a quick Google search to learn more about Google XLA. So, it's this accelerated linear algebra library. And so it's cool to hear that this is kind of a standard across hardware devices to allow, yeah, these, these machine learning particularly deep learning linear algebra operations to be run you know, across, across all these different kinds of devices. Very cool. I had no idea about that. So, to sort of start to wrap up to get into your career a bit, you are a physicist and electrical engineer by training, and before Amazon you worked for other chip makers Zoran and Annapurna, which as you mentioned earlier Annapurna was acquired by Amazon. So, how did you get involved in deep learning-specific chip design? Like as I was talking about a few minutes ago, you clearly have this huge depth of knowledge in the space. Was AI always an interest of yours or, you know, how did this come about?

Ron Diamant: 01:26:54 It's kind of funny. Since I remember myself, I kind of had two major points of interest technology-wise. One was math and algorithms, I always was drawn to this area. And the other one was building things, building compute systems and so on. And I was fortunate enough that my career kind of evolving that direction as well. So, I used to do security products or cryptographic accelerators. I moved to doing compression and error correction code for certain chips. And in all, in all of these cases, we basically combined a deep familiarity with math algorithms and how to build them effectively in harbor. So, when we when I joined the Annapurna labs I kind of took ownership of the math-related acceleration engines that we had to build.

01:27:51 So, we had a cryptographic engine. We worked on a DPU, a Data Path processing Unit, or data center processing unit, which later, it was the first in the world by the way, and it later evolved to be the AWS Nitro product that is pretty well known today. But then as we identify this AI flywheel and the, and the need to service it and provide our customers with the best possible performance and price performance, it was kind of natural for me to take a crack at understanding the math algorithms and tying it back to hardware. So, I was requested to take on that effort. And, and I'm very fortunate to have done that because it's, it's a blast. I enjoyed it a lot.

Jon Krohn: 01:28:32 Nice. Yeah, it's certainly an amazing place to be, as I'm sure a lot of our listeners would agree. And we are all grateful to have you, Ron, working on these kinds of innovations to allow our side of the flywheel to be accelerating as well, and to be able to build lots of cool real-world applications using your hardware. You have over 200 patents across a wide range of areas, security chips, compilers, AI accelerators. What do you see happening for you next? Is, I imagine there could be a huge amount of depth in this AI area, and you're just kind of like, gimme AI all the time for as long as I can, or, yeah. What do you see for yourself going forward?

Ron Diamant: 01:29:15 So, so I'm having a blast with our AI teams. I especially enjoy working in Amazon. We have a bunch of super talented teams across a wide range of domain expertise,

		compilers and servers, and chips, and firmware. So, I, for now, I don't see anything, anything other than continuing to make this that this as, as efficient and as optimized as possible. And time will tell what's next after that.
Jon Krohn:	01:29:44	Very cool, Ron. All right. So, before I let you go, a question that I ask all of our guests is, do you have a book recommendation for us?
Ron Diamant:	01:29:51	It's a, that's a tough one for me because most of the books that I read are technical books. But there is a book that-
Jon Krohn:	01:29:57	You can, you can give us a technical one too, you know, like you know, whatever you think is best.
Ron Diamant:	01:30:04	No, that's, that's fine. I think a book that I read recently and I found it to be illuminating is a Richer, Wiser, Happier which talks about investors including Charlie Munger and Warren Buffett. I just find that these folks are, are they, they're, they're operating in such a complex field where you're probably even the most successful ones are wrong 40, 45% of the time. Cause you need to bet about the future. And getting into their mindset and their mental models and how they deal with making these decisions is, is super fascinating to me. And in many cases there, these are also folks that you can learn just tricks on how to live a happy life from, so I love that book.
Jon Krohn:	01:30:53	Sweet. Well, great to have Charlie Munger back in the conversation. And yeah, Ron, this has been an extraordinary episode for me. I've learned a ton about an area that I previously knew a little about. I'm sure our listeners loved hearing from you as well. How can they follow you to, yeah, you know, after this podcast, if they want to hear more of your thoughts or, you know be in the loop on other podcast appearances or YouTube videos or whatever that you make, how can they do that?

Ron Diamant:	01:31:22	Sure. So, I'm on LinkedIn and I'm on Twitter. More active on LinkedIn to be honest. And yep. Happy to hear from anyone interested.
Jon Krohn:	01:31:30	Nice. All right, thanks so much Ron. This has been a fantastic episode, as I already said. And yeah maybe at some point we can check in again in the future and hear about the hardware innovations that are happening now.
Ron Diamant:	01:31:44	Thanks for having me, Jon, it was a blast.
Jon Krohn:	01:31:51	What an intensely educational conversation that was for me. I hope you took a ton away from it too. In today's episode, Ron filled us in on how GPUs with optimization for massively parallel workloads enabled deep learning to bring about the modern AI era. He talked about how specialized AI accelerators like TPUs, IPUs, and his own Trainium and Inferentia chips, can greatly exceed GPUs capacities and efficiencies because they were designed from the ground up specifically for deep learning. He talks about how the same microchip can be connected to a circuit board in different ways so that one, like the Trainium chip can be optimized for high bandwidth connectivity while another accelerator like the Inferentia chip can be optimized for greater compute density. He talked about how Charlie Munger's inversion exercise guides him on the multi-year journey of chip design as well as many other life decisions of his.
	01:32:43	He talked about how data tensor and pipeline parallelism enables massive models such as LLMs to be trained, even if they have trillions of parameters. And he filled us in on how the Neuron SDK integrates AWS AI accelerator hardware into the popular PyTorch and TensorFlow libraries. As always, you can get all the show notes including the transcript for this episode, the video recording, any materials mentioned on the show, the URLs for Ron's social media profiles, as well as my own

social media profiles at superdatascience.com/691. That's superdatascience.com/691. If you live in the New York area and you would like to engage with me in person, not just online, on July 14th, I'll be filming a SuperDataScience episode live on stage at the New York R Conference. My guest will be Chris Wiggins, who is Chief Data Scientist at the New York Times, as well as a faculty member at Columbia University. So, not only can we meet and enjoy a beer together, but you can also contribute to an episode of this podcast directly by asking Professor Wiggins your burning questions on stage.

01:33:47 All right, thanks to my colleagues at Nebula for supporting me while I create content like this SuperDataScience episode for you. And thanks of course to Ivana, Mario, Natalie, Serg, Sylvia, Zara, and Kirill on the SuperDataScience team for producing another deeply fascinating episode for us today. For enabling that super team to create this free podcast for you we are deeply grateful to our sponsors. Please consider supporting the show by checking out our sponsors' links, which you can find in the show notes. Finally, thanks of course to you for listening all the way to the very end of the show. I hope I can continue to make episodes you enjoy for many years to come. Well, until next time, my friend, keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience podcast with you very soon.