Determined Podcast Series

Tianqi Chen on Efficiently Deploying Models on Specialized Hardware

Podcast Transcript

Craig: Hi, I'm Craig Smith and this is Eye on A.I.

Last year, I had Evan Sparks and Ameet Talwalkar on the podcast to talk about Determined AI, their startup which aims to provide cutting-edge deep learning software infrastructure for everyone. Since then, in line with their mission to democratize ML infrastructure, they decided to open source their deep learning training platform in hopes that it will help machine learning engineers to more easily build better deep learning models.

Ameet, a professor at Carnegie Mellon University and a leader in the emerging machine learning systems research community, is joining me in a series of five episodes to talk to some of his friends and colleagues about various aspects of the machine-learning pipeline, from data preparation, model development and training, to hardware management and deployment.

We're now at the deployment end of the pipeline and in this episode, we talk to Tianqi Chen, also known as TQ, about his latest project, an open-source compiler called TVM that helps data scientists optimize their model's performance on specific hardware.

I hope you find the conversation as informative as I did.

Craig: Why don't we start, Tianqi, by you introducing yourself. where you studied, and your work history, and how you came to, this TVM project?

Tianqi: Yeah, sounds good. My name is Tianqi Chen. You can call me TQ. And I was a PhD student at University of Washington for six years and my major research area is on machine learning and systems. So, the idea is to build systems to scale out machine learning algorithms and make them deployable on all kinds of different hardware devices. And right now, I'm currently serving as a CTO at a startup company called OctoML and I'm moving to Pittsburgh to join Carnegie Mellon University as an assistant professor,

Craig: Great, great. Well, congratulations on that.

Tianqi: So, to touch more about the particular thing we want to talk about: I have built a few key machine learning systems before that are widely used, including XGBoost, which is one of the data science toolkits that's widely used by a lot of data scientists.

Ameet: You started XGBoost before grad school, right?

Tianqi: It's close, like at the beginning of my grad school, as opposed to before I joined UW. And the MXNet is another major deep learning framework. And now, my major focus has been on TVM, which is an open source machine learning compiler.

Ameet: Can I just jump in and say, any one of those projects would have made for a grand slam project as a graduate student. To do two of them would have been, I don't know what's bigger... But to do all three of those is sort of unheard of.

Tianqi: From my perspective, I will always want to see my research being landed and being used.

Craig: And TVM stands for what, Tianqi?

Tianqi: Usually people just say TVM as it is, but it was originally from the compiler name called LVM, and as an analogy, we try to create something for Tensor. so, T stands for tensor, and VM usually stands for virtual machine. In this case, we are saying that we want to build a compiler that's able to compile Tensor operation-based programs on different kinds of hardware backends.

Nowadays there's a lot of machine learning models, and a major category of machine learning models is called deep learning. And the models that run, say, voice recognitions, image recognition, speech machine translation, all those models can be represented as a Tensor operation program.

Craig: What is end to end philosophy and how does TVM relate to this idea of end to end philosophy?

Tianqi: In a lot of cases where we are building applications, there are always application boundaries. So, the idea is that, say, the hardware vendors will just give you a hardware and then people build software libraries on top of that, and then people build an application on top of it. However, more recently we started to see a trend of more need for end-to-end, because hardware designers are starting to design domain specific hardwares for, say, running image recognition or speech recognition. Meanwhile, machine learning researchers are also designing new models that try to fit to those specialized accelerators. For example, one important case is Google's transformer model, which more recently there's a variant called GPT3, which is really popular.

And that model got started mainly because it actually runs very well on all those GPUs and TPUs. So, in some sense, in order to get to the next mile in AI, people are starting to try to co-design the machine learning models and the software and hardware backend together.

Ameet: TVM, is it more focused on the training or it's more focused on deploying these models on proliferating hardware, on edge devices, and so on?

Tianqi: Actually, TVM originally was mainly focused on deployment. The goal is that, because there are a wide spectrum of hardware devices nowadays, it's really hard for humans to go and manually optimize machine learning models on each type of device. So TVM aims to provide an automated solution that takes a model from TensorFlow or PyTorch and automatically generates a deployable model. However, more recently, we also started to optimize training-based workloads as well.

So, the steps are, you want to build a machine learning model to do something. You collect the data. You prepare the data. You build the model. You train the model. And the last step is deployment. And you're going to deploy it on a server, or on an edge device like a cell phone, or on some hardware that doesn't even exist today. The hardware needs a set of instructions, a piece of software to unlock the power of that hardware. And today that's very manually intensive. So TVM is using machine learning to automatically create that software to unlock the power of that hardware.

Craig: Can you explain how it does that? How it searches for the optimal program to use, and how it creates an executable module?

Tianqi: actually, what we were trying to do is we are trying to build something like an AI or machine learning based solution that mimics the human programmer mind. So, when a human programmer starts to optimize a program, what they will do is they will first evaluate the potential ways to write the same program. For example, when we are trying to write, say, matrix modification or some operations, there are different ways that we can do that. And there are also different ways that people can, for example, temporarily copy a certain part of data into a special cache in the hardware so that it runs faster.

There will be a space of billions of possible programs that people could kind of arrive. However, some of those programs run faster than others. And traditionally, it's human experts who use their domain knowledge and intuition to actually craft such a special program for the tensor operations. What TVM tries to do is we try to formally define that search phase and the machine tries out different kinds of things. In the meanwhile, we're also using machine learning itself to learn, to predict, if we try this program out how fast it will be. And usually that prediction runs faster than actually running on the target hardware. So, in that case, by using automatic search, we'll be able to mimic what human programmers might do. And sometimes because the machine has more energy and resources and time than a human, they can sometimes find better solutions.

Ameet: The GPUs themselves are powerful hardware, but it's not the GPUs alone that unlocked the ability to train and eventually deploy. deep learning models faster. It's also kind of the low-level software so maybe you could say a little bit about that

Tianqi: Yeah. So, we are talking about the ability to use all those powers, right. Why hardware has in theory those potential capabilities, the software stack is actually one of the very problematic parts for a lot of use cases. In particular because we want to offer all kinds of machine learning models onto those hardware devices. You know,

people will need to rely on software stacks. For example, one of the most successful hardware companies for deep learning is Nvidia. And they have dedicated a lot of engineering resources to create a software stack specifically for Nvidia devices for a small collection of machine learning models.

So, we are trying to automate that process, including for, Nvidia devices and other hardware devices, we want to be able to automate that process as much as possible. So, we can easily recreate a new software stack that's specialized for a particular model the user might be interested in.

Ameet: One analogy is I used MATLAB a lot back in the day and maybe I'm dating myself a little bit. But the linear algebra operations in MATLAB, the ability to invert a matrix or compute an SBD were really, really powerful because they were these low-level primitives that it called upon, that had been incredibly optimized. And for those in the audience, who don't know exactly what an SBD is, it doesn't matter so much. It's more that you have these big matrices of data. There's a bunch of different operations you want to perform on them. And these low-level libraries, they don't just say, "Okay, let's perform this operation." They say, "If your matrix is tall and skinny, do it one way. If it's symmetric, do it another way. If it's sparse, do it another way." So that's like classical, linear algebra primitives. Similarly, for deep learning, again, TQ's the expert here, but my understanding is that, say, for convolutions, even in GPUs, there's not one way to execute on a convolution - there are 5, 10, 20. And the point is, it really matters. The naive way of executing a convolution on a GPU could be orders of magnitude slower than the more efficient way to do that.

Tianqi: Yeah. I can to speak to that because, 10 years ago, when I was an undergrad student, I was working on deep learning, and there wasn't a good software stack then, so I had to basically spend one month to just write GPU code, to offload my machine learning model on to that piece of hardware. Today, at least for Nvidia, they have libraries for that. So, it's much easier to actually offload those machine learning models on top of this hardware. But there are also emerging models we need to support. Which, the hardware vendor may not be able to catch up very quickly, because you have a new model coming in. There's new variation of convolution and other operators. And there are also possibilities if you want to use another piece of hardware that do not have a software stack.

Craig: You were saying when we spoke earlier that with Nvidia's software stack, TVMs can get a model to run even faster than their software stack. Is that right?

Tianqi: In some cases, yes. So, to be clear, we also get contributions from Nvidia, because they are also interested in the automation capabilities of TVM. When we are talking about machine learning models, usually there are benchmarks that people use to show off how good their software stacks are. On those standard benchmark models, usually we cannot do as good, or we can just do maybe 1.2x as what Nvidia's or other companies can do, because that is the benchmark that humans go and keep optimizing for. However, if you go out and talk to, say, your industrial partners, your friends who work for a different company, they will tell you that the machine learning model they are interested in does not necessarily belong to those standard benchmark categories.

And in those cases, because of the objective of those big company employees, they are getting paid to optimize the benchmark. But they are not necessarily getting paid to optimize the models that the particular company is interested in. So, in that case, we find TVM gives you even more dramatic speed up specialized to that particular model. We can see speed ups from range of 2x sometimes on some interesting special models that can even be like a 5x speed up versus what the original software stack can provide.

Craig: Yeah. And then, do TPUs come with a software stack as well?

Tianqi: Yes. So, TPU is this specialized hardware built by Google, and TPU does come with the software stack. However, that software stack is still internal to Google.

Craig: I see. Do you need to build your own software stack if you're using a TPU outside of Google?

Tianqi: for now, because Google does not open up their internal instructional sets, we can only use TPUs through Google's software stack at the moment. However, TVM does support a wide range of other hardware similar to TPU, for example, more recently, Arm is contributing support to Arm's NPU. So, inside TVM, we also have an open source TPU-like accelerator called VTA, which is our prototype used to test the capabilities of the solution to supply those specialized x-rates.

Craig: Okay. Could you walk us through how TVM works?

Tianqi: Yeah. So, at a high level, what TVM does is they take machine learning models from other model frameworks into a high-level representation. What we call high level functional representation or computational route.

The idea is that the computation steps in each of the image processing or speech recognition models using a graph-based format where each of the nodes will correspond to an operation that we perform, and the edge of a node corresponds to the data that flows through that computation. And then TVM will perform some of the high-level optimizations to transform the computation into some equivalent form that is faster. In the meanwhile, for each of the nodes we also try to generate specialized code on a target hardware. In that case, we are using the exploration and learning based strategy where we will try to, based on high level description, generate billions of variations of possible programs that we might want to explore, and then use machine learning to guide which program we want to try out on the actual hardware. The idea is that we want to smartly select potential candidates to send them to a hardware for benchmarking. And then we would try to pick the best one based on their actual performance on that target hardware.

Ameet: Can I follow up on that? So, if I understand correctly in some sense, you're kind of doing a search where if a certain space is the set of programs that are generating different programs, with idea of hopefully generating a program that you evaluate and find that it's better than what you had before and you want to keep optimizing for that.

Tianqi: Yeah, exactly.

Ameet: The search base, without any structure, is really, really large. It's a set of all possible programs, which is enormous. How do you search through that space? I assume you don't randomly sample. There's some structure that you're exploring.

Tianqi: Yeah. So random sample is definitely one potential strategy that we can use.

However, like you said, we cannot search over the space of all the possible programs. Right? Because that's like exponentially large. So, what we do instead is we are trying to develop a set of equivalent transformations that we know that if we transform the program this way, for example if we reorder the loop in a certain way, the program is going to be semantically equivalent. we call this, scheduled primitives. So, this is not a new idea, this idea that we borrowed from a previous compilation community. So, there will be a collection of scheduled primitives, and we will be able to order them together.

And by performing all those set up primitives, it will give us a transformative version of the program. So, our problem becomes search over, "What is the best sequence of the transformation primitives?"

Ameet: Do you need to check whether your transformations are leading to an equivalent program or do you kind of know already, just based on how the different instructions are set up, that kind of just know in advance, which transformations are feasible without having to check.

Tianqi: Yeah. So, for now we are trying to build transformations that are always correct. You are right. We can also build it another version that verifies the correctness so that we can apply more like fuzzy transformations.

Craig: Can I jump in with sort of a layman's questions to slow things down a little bit? How do people do this without TVM, number one? And two, can you just, for listeners that aren't familiar with the use of the word, what you mean by primitives?

Tianqi: So, in terms of like, normally, what do people do? Traditionally what people do is people will just call into libraries. So, the idea is that we will rely on hardware vendors to provide the optimal, or a good set of programs that implement the tensor operation, such as convolution and matrix modification. Ten years ago, I kind of had to do it by myself, but now there are libraries built by hardware vendors to do that.

To answer your second question, the primitive operators here are meant to be the most basic operations that we can call into to implement a step in the machine learning process, and that step of convolution operation that tries to take a low level of representation such as address of the picture and combine them together.

Ameet: And the idea is that, for a new piece of hardware, there might be some primitives, which are convolutions, pooling, matrix, basically the standard operations that comprise a deep learning model. And, while the hardware vendor might provide some default support for these different operations, you can overwrite that default. You're not trying to get a different answer. You're just trying to get the same answer twice as fast, or a hundred times faster, or whatever.

Something that I work on a lot is hyperparameter optimization and more recently neural architecture search. The problems aren't the same, but there are, I think, really interesting overlaps between them. And I think it'd be interesting to hear your thoughts on that and I can share mine as well.

Tianqi: Yeah, definitely. I think there are a lot of interesting overlaps and maybe, some of the methods can be borrowed over as well. For example, when we started trying program optimizations, we did also use ideas like exploration, and exploitation and tradeoffs in our optimization. And also tried a few methods from the Bayseian optimization community. And a few interesting differences in terms of characteristics. First of all, we are searching over structured program space, while traditionally in hyper parameter search, most of the parameters are less structured, in the sense that, you just try to pick the numbers, as opposed to pick the best program. However, we are also parametrizing our search in the sense that they are also parameters that look like hyperparameters that we want to pick from. So, there are definitely similarities there.

The fact about a structure search is that we will want to build, say, cost models or models that predict how good the programs will do based on the program structure. so that, it's kind of aware of the domain. A second interesting perspective is that, we want to be able to find a program for a collection of machine learning models, or if you combine the amount of different models we want to optimize, there could be thousands of different variations out there. And they are always correlated. So instead of trying to do the search from scratch every time, there are opportunities to share the common knowledge among those searches.

So, I think it's an interesting problem set up that we can use technologies like hyperparameter search and structure search, to this domain, but also trying to develop specific algorithms that still apply to the particular domain.

Craig: within TVM the first step is to define the search space. Is that right?

Tianqi: The search space is not defined by the user, but the system will define the search space for a given operator.

For different programs, there are some common traits about search space. It's about sharing the machine learning model that predicts the program. So, the idea is that, you know, for a given program in that search space, we want to be able to predict how good that program is.

Ameet: The search space here is a set of all valid programs that you could, in theory, run on a specific piece of hardware. "Valid" here means that they all give you the same answer. So, they all form the same three by three convolution.

To follow up on the point about the connections to hyperparameter search: I think the biggest thing that I see is that they're not the same problem, but there are some interesting overlaps. And I think borrowing ideas from one to the other, going either way, it could be really interesting.

Tianqi: Yeah.

Ameet: There is some recent work, by folks at Google: called Auto ML Zero, which does take this cogeneration approach. The one thing that's different about hyperparameter search, and again, to remind everyone, hyperparameter search, the goal here is, the search space here now is again a set of, say, neural network models and also knobs related to training these models. And the goal is to find the best settings for these different knobs that you can tune. Such that, here you're optimizing for accuracy, not for speed. So that's one big difference. You're optimizing for speed with a fixed kind of answer. Hyperparameter search is optimizing for accuracy, but under this rough hyperparameter optimization or neural architecture search or Auto ML goal, because AutoML Zero does take this program generation approach, and it's very general, they show some sort of promising results, but it's really, really a large space and it's so expensive. They don't tell you how expensive it is. And I don't think that they leverage some of the structure that you probably are leveraging So yeah, I think, it'd be really interesting to look at the connection between those two.

Craig: So, TVM. It does a search within this defined search space, and it comes up with a series of programs that are applicable. And then it tests each program to see which are the best, according to speed and some other metrics. And then it gives the user a choice of which of those to use. And it gives you, for each, a deployable module that you can plug into your software. Is that a fair description?

Tianqi: Yeah, that's correct. the TVM will pick the right operators based on the programs for each of the stages and they will patch it together as a software that you can put into your software stack.

Craig: And do you start by defining your hardware so that it is testing against that particular hardware? Or could it be testing against a range of chips and then giving you different combinations? It'll run at this speed on this chip, but that chip costs this much money, or consumes that much energy. It'll run maybe a little slower on this chip, but this chip is cheaper and consumes less energy. Or that sort of

Tianqi: That's an excellent point, actually, you know, while this is not a direct feature, offered by TVM, it's something that we can easily build on top. Because, what TVM provides you is for a given hardware target and a given model, it will try to get you the best performing deployable module of that model on a particular hardware. So, what we can do is we can just build a software solution on top to try out each hardware backends on TVM for the same model, and then give you their suggestions of potential results. And this is actually something that OctoML, the company, is actually building a service for us so that the user doesn’t have to worry about these kinds of headaches in terms of trying things out and compiling across different hardware.

Craig: Yeah. And you're referring to OctoML, which is your startup. Can you tell us a little bit about that?

Tianqi: Yeah. So, OctoML is a startup company that we built around TVM. The goal is to actually support the open source community, but also build a product on top of it. So, in particular, one of our products we are currently building is called Octomizer, which is basically a service that exposes TVM as a service, so that user can directly

upload their models to the service, and it will return to the user a module that user can then go in plug in to their software environment. So, they don't have to worry about managing the compilation process, as well as compiling against different hardware backends and so on.

Craig: that's the model that Determined AI is following where the basic solution is open source. But there are people who aren't going to want to have to dive in and learn it. Or, there are people who don't want to spend the time to build bells and whistles on top of it. And they then will turn to the commercial enterprise which, of course having created the open source software, is the world expert on it. Is that right?

Ameet: Yeah, that's right. I think that the way that the two sets of projects, TVM and what we're building at Determined, the way they were started were different. TVM was started as an academic project and the startup came afterwards. Whereas for us, it was kind of the opposite. But I think in both cases, both to get a community behind a project, to get contributions, to get feedback, and also to help with adoption, the machine learning community kind of demands open source. And I think it's a good thing. Right? So, to help with adoption, people want to be able to play around with things. Even if they eventually want to pay for something, they want to get their hands dirty first. And it's a great way of, whether you're an academic project or a company, it's a great way for the folks developing the software to get feedback, too. So, it makes the product a lot better, a lot quicker, and it kind of helps to get the word out.

Craig: And Ameet, Determined is a platform that sits kind of in the middle of this ML pipeline. It's a platform on which you build machine learning models. Are you then using TVM?

Ameet: not yet, but I think it is a very complimentary sort of thing. One thing that we are banking on is this idea that there already is and there will continue to be a proliferation of different sorts of hardware, both for training at deployment. For somebody developing models, it'd be nice to not have to be locked into one piece of hardware, or to have to worry about which one to use when, and be able to benefit from different sorts of them.

And, that's a bet we're making at Determined. That's certainly a big reason that TVM exists. So, from that point of view, we're hardware agnostic and we think being hardware agnostic is more and more important moving forward, because there's going to be more and more hardware to be agnostic to. And I think TVM and OctoML are also banking on the assumption, which I agree with, that this proliferation is going to continue to get bigger and bigger.

Craig: And TQ, we've had a conversation with David Patterson about the evolution of hardware for machine learning. And, he was saying that there's now specialization, not only in chips for machine learning, but chips for specific algorithms within machine learning. Is TVM able to adapt to every new piece of hardware that is developed going forward?

Tianqi: Yeah, that is exactly one of the motivations that built TVM, there is a lot more specialization happening. And as Dave Patterson mentioned, it's the golden age of computer architecture, where there are, hundreds of companies working on different variations of specialized chips.

The problem of targeting those chips each of them to have their own way of operating, right? So, they have their own instruction sets. And, in the case of TVM, this does create a very huge challenge. But our current proposed solution is not only trying to create a description of the high-level model, but also trying to create a description of what certain hardware is able to do a special kind of operation, like a matrix publication four by four. Another hardware might have a different operation, but we try to create this two-level description. And the goal of the system is trying to incorporate these two informations, so that we can take the high-level representation to target low-level chips.

Of course, this is still a very open area, in the sense that the solution that TVM targets specialized hardware nowadays still requires a bit of manual effort in terms of how do we define the hardware stack, as well as how do we define the search space, especially targeting the hardware. by having a description of the high-level needs and high-level capability, we want to be able to automatically use machine learning to create that.

Ameet: I'd love to hear more about that. So, let's say I am a hardware vendor. I come up with a new specialized piece of hardware. And I say, okay, TQ knows how to help me unleash the power of my hardware.

Tianqi: So, if you look at the specialized hardware, actually, there are still some common patterns among them. For example, each of the hardware will have some set of memory hierarchy, which means that it will read the data from global DRAM into some kind of registers. And sometimes the register is partitioned in different ways and sometimes they are like L1 level cache, so there will be a different description of memory hierarchy. And then, each of the hardware will also have their special instruction sets. So, for example, in a typical CPU, or like a GPU, their normal instruction sets are skeleton instruction, which means that you can take a floating point, take another floating point and multiply together, and then store that into certain target devices.

In order to create accelerations, though, a lot of the newer version of accelerators have more coarse grain operations. So, an instruction is not about multiplying one floating point with another floating point. An instruction can be something like take a matrix that's stored in your local register file and multiply that matrix with another matrix.

Ameet: That high level operation that was implemented in hardware rather than...

Tianqi: In the hardware. So, what we do is actually we try to also describe the schematics in TVMs' internal representations. For example, if it's a 4x4 matrix multiplication, we would describe the TVM saying that this hardware has this primitive called 4x4 matrix multiplication.

Ameet: Abstractly, is the process you just described basically defining the search space in terms of whatever primitives, whether they're lower level or these advanced higher level...

Tianqi: So, I think it's kind of decoupled. First of all, you need to define the hardware capability, right? You describe the capability of matrix multiplication instruction and the capability of, or what we call schematics of the hardware instruction. Once you have that, then you can try to derive a search space so that they can map the high-level description to the low-level hardware instruction. Right now, that search with definition step is a bit manual. In the sense that for, say, Nvidia Tensor Core, we need a developer to go and write out that search space. What the mapping might look like.

Ameet: The search space derived from those specs is somewhat manual.

Tianqi: Yeah. And a research goal I will have to continue working on is actually try to make that more automatic. The idea is that you take a stack from, say, the hardware instruction, and you also take a stack from what the user might want to do, and automatically derive a search space that allows us to map from the high-level demands to a less special instruction than hardware might provide.

Ameet: I see. Okay. So, step one is create the hardware and fix the specifications. That's what the hardware person does. Step two, which is currently somewhat manual, but you will figure out how to make it automated one day, is coming up with a search space of operations that you can perform using that spec. You run TVM to search over that search space. This next question I'm asking is, is TVM run once or is it run once per particular model -- is it optimized for the hardware or optimized for the combination of hardware and particular model that you want to be used in the hardware?

Tianqi: Yeah, it's a combination of the model and the hardware.

Ameet: So, this whole process of coming up with a new search space, you do each time you have a new model?

Tianqi: Not really. Because we can define a common search space template for a certain type of operator, like a convolution.

Ameet: But you might discover new implementations for each conceptual operation you want to perform for every model that you're using.

Tianqi: Yeah. That's correct.

Craig: And then TVM takes that program or what it's found in that search space and writes a module that can talk to your specific model. Is that right?

Tianqi: And writes a module that can run the specific model on that specific hardware.

Craig: Have you guys worked with Cerebras on the wafer scale engine?

Tianqi: We have not worked in particular with Cerebras in this case, but their problem is actually quite interesting. In the sense that they need to build a quite specialized solution for their wafer scale computing.

I think TVM will be able to be used as part of a solution in the wafer scale computing as well. In the sense that, even in wafer scale computing, there are two tasks that you normally need to do. And that's high-level planning which is about how you place your operations on the wafer itself. And in that case, it's quite close to the high-level graph optimization capabilities that TVM might provide. Another thing that you need to do is you always want to be able to generate low level code that runs on each of the, basically, a cell in a wafer. And the program generational search techniques that we talk about here are quite relevant to that problem.

Ameet: So, if we're running TVM for every single model, for inference, I can imagine I've already trained this model, I know I want to run it on a chip. I'm planning on doing inference a lot. It's certainly worth spending time optimizing the execution of this model on this chip because I want to do it faster, cheaper, more energy efficient. How expensive is that step? And is the overhead of just doing the TVM optimization -- it seems clear that it makes sense in deployment. For training, depending on how big the effort is...

Tianqi: A lot of that also boils down to how much that you have to do from scratch. One of the interesting perspectives of machine learning models is that the more data you feed to those models, the smarter they become. In other words, you know, machine learning models are very good at memorizing interpolated things. So, imagine that we build a system that's already seeing a lot of machine learning workloads, and collect data about those workloads. Then that model is already memorizing the operations that you have performed before. Right? So, the idea is that, by being able to learn from the huge amount of historical data, the amount of effort we need to spend or optimize for future...

Ameet: The cost of TVM optimization gets lower and lower over time because the model itself is learning from previous models that's optimized?

Tianqi: Yes. So, that is also still an open area in the sense that, right now we do not guarantee that we will have that determined case of lowering, but we have already seen examples of model transfer where we can make use of historical data to be able to reduce the optimization of the future workloads.

Craig: So where does TVM go from here? What are some of the features that you're looking at building out? Not necessarily through OctoML, but in the open source.

Tianqi: there are quite a lot of exciting things that we are looking at. First of all, we are looking into covering more machine learning model variations. So as a first step, most systems start as a static graph model where we know the contradictions are known ahead of time. I have a lot of new models, like natural language processing, which start to contain more dynamics. So basically, you need to wait until you run to know what the next step will look like. So, there is some effort in supporting that. There are also efforts supporting what we call TinyML. The idea is that instead of putting the machine learning on server classes, we also want to put them in those

very tiny devices, it's not even like a mobile phone, it could be something like a sensor that is very cheap. And being able to deploy machine learning models on these devices. Smart microphone could be an example. If we want to manufacture a very cheap smart microphone, we could use TVM to offer some of the, you might say, wake-word model onto those devices. As a matter of fact, in last year's TVM conference, our friends from Amazon told us that Alexa was already using TVM to run the wake-word detection.

There are also efforts on automating more specialized accelerator support. Ameet and I already talked about that problem of how do you describe the hardware and how do we automate that process? So, there is still a lot of interesting research on their direction. And there is also effort on trying to make it more accessible. Just like machine learning software like PyTorch or TensorFlow, one of the reasons a lot of users like to use them is because people can just write a few lines of Python code to express their need, and build a model out of it.

We also have a vision to try to make the machine learning optimization and deployment accessible, so that users can also write a few lines of Python code to express their intent and then be able to automatically optimize those models and also customize the behavior of the compiler based on that.

Ameet: Who do you view the users of TVM or OctoML to be? So, one set of users is anyone developing new hardware, but I get the sense that you have a broader set of users.

Tianqi: There are a few categories, right? Definitely, there are people who are hardware vendors who want to build better software support for their hardware, is one category. For example, in the TVM open source community, we got contributions from, Qualcomm, Nvidia, Intel, basically almost all the hardware vendors that you can think of. There are also end users who are interested in deploying their machine learning applications on different settings, like, deploying their machine learning models on tiny devices, or on server class devices.

Ameet: So, it could be, even if I'm using a more standard chip, I'm using a GPU. If my model is exotic enough, it could be that TVM could help further optimize the primitives, even on that pretty well-established chip.

Tianqi: That is correct. And we have seen a lot of stories around that, actually.

Ameet: So, it's much more general. There are a lot of people creating new hardware, but there's many, many more people, I think, creating new models. A lot of this discussion has been on deep learning. and for good reason, right? A lot of the hardware proliferation is based on deep learning, the cost, the amount of data, and so on. For instance, the reason I thought about this was we were talking about LAPACK before and how MATLABs linear algebra operations, you know, in the same way that you can kind of, as a sanity check, see how well you do relative to NVIDIA's built in CUDA on standard models. And as a sanity check, the fact that you can do about as well is really impressive. Have you looked at either traditional ML or even not ML stuff like LAPACK or something like that?

Tianqi: Yeah, so, I would say most of the ML algorithms that use Tensor operation can apply, there is a lot of support around Tensor computations, there are also research on repurposing traditional models, like decision trees, to use tensor-based operation. So, there's a project from Microsoft called Hummingbird, that actually compiles the decision tree models into the operations that runs on TVM and PyTorch. And by using TVM they can beat customized written decision tree inferences, which is kind of amazing.

So, the same tactic does apply to broader settings than the deep learning setting, which is already quite important.

TVM provides optimizations for both edge devices, like RaspberryPI mobile devices, as well as those IOT style devices that are even more lightweight and require even more fine grain optimization. And optimization there includes things like using exotic computations like integer operations, or sometimes people will try to pack bit arithmetics into these computations because they cannot afford floating-point units on those devices. And because those computations are so exotic, there are no previously provided libraries for them. And, TVM is a very good use case to generate libraries for this kind of computation.

Ameet: So, we talked about, TVM is optimizing two pools of hardware and particular architectures. Does it care about the weights of those architectures? I know we talked about this before, but I think it's a good point to clarify.

Tianqi: Yeah. So, for some optimizations like just finding the best program for the operation, we do not need weights. And there are use cases where people rightfully want to protect their weights for privacy reasons. So, we can perform certain optimization on that. But there is also further optimization we can perform if we know the weight information, for example, we could perform quantization, or do some layout transformation on a weight, so that they can run more effectively.
