ML Accelerators

CS4787/5777 Lecture 22 — Fall 2025

Recap: ML on Hardware

- Over the past few lectures, we've been talking about ML Systems techniques that affect the hardware
  - Parallelism
  - Distributed learning (basically super-parallelism)
  - Low-precision arithmetic
  - How machine learning frameworks interact with hardware
- However, all of this has been very agonistic to the specific hardware we're running on.
- In this lecture, we'll look at different hardware targets: **why GPUs became dominant for machine learning training**, and what the competitors are to GPUs

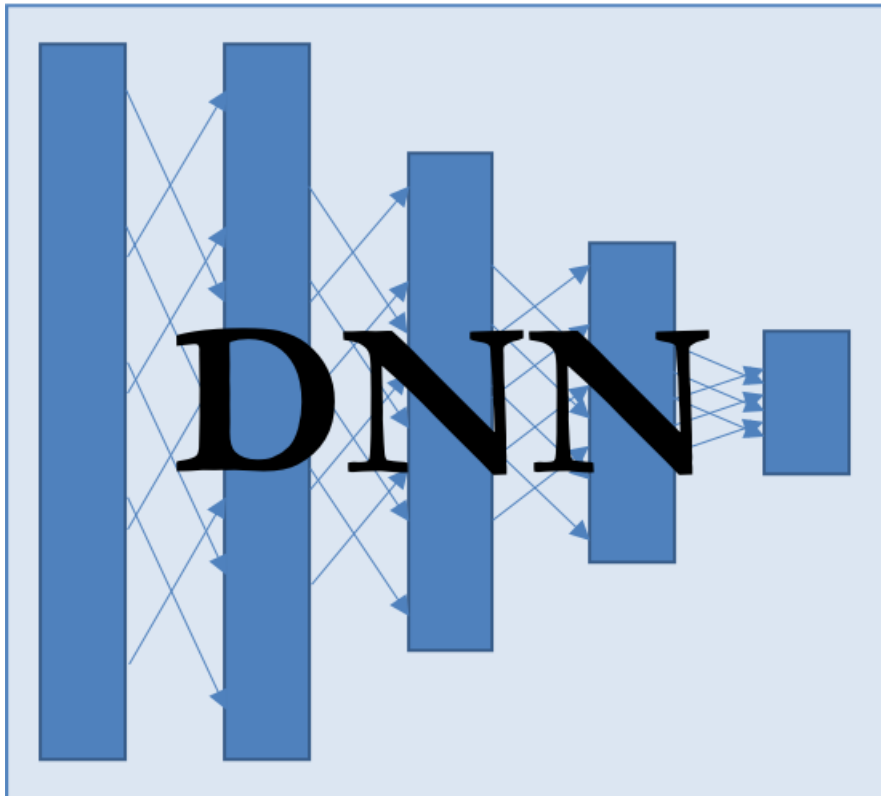
Modern ML Hardware

- CPUs
- GPUs
- FPGAs
- Specialized accelerators

Right now, **GPUs are dominant** for ML training…as we've seen

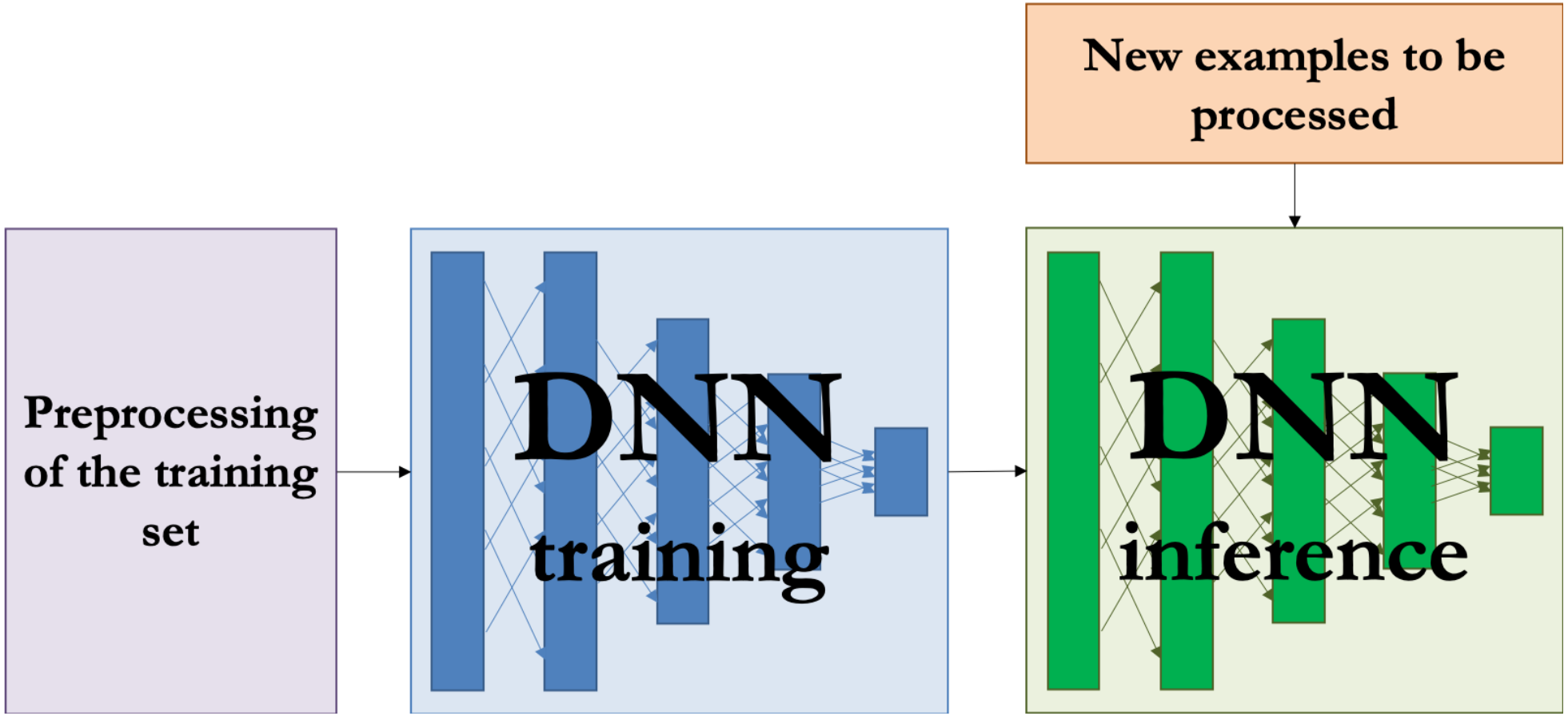
What does the modern ML pipeline look like?

- It's not just training a neural network.



What does the modern ML pipeline look like?

- The pipeline has many different components



Where can hardware help?

Everywhere!

There's interest in hardware everywhere in the pipeline

- both adapting existing hardware architectures, and
- developing new ones

What improvements can we get?

- Lower latency inference
  - To be able to make real-time predictions
- Higher throughput training
  - To train on larger datasets to produce more accurate models
- Lower power cost
  - Especially important as data and model sizes scale

How can hardware help?

Speed up the basic building blocks of machine learning

- Major building block: matrix-matrix multiply
- Other major building blocks: convolution, attention

Add data/memory paths specialized to machine learning computations and workloads

- Example: having a local cache to store network weights
- Create application-specific functional units
- Not for general ML, but for a specific domain

Why are GPUs so popular for machine learning?

Why are GPUs so popular for deep neural networks?

To answer this...we need to recall what we learned about GPU architectures.

Recall: GPU Parallelism

- The GPU supported parallel programs that were more parallel than those of the CPU.
- Unlike multicore CPUs, which supported computing different functions at the same time, the GPU computes the **same function on multiple elements of data**.
  - Example application: want to render a large number of triangles, each of which is lit by the same light sources.
  - Example application: want to transform all the objects in the scene based on the motion of the player. That is, the GPU could run the same function on a bunch of triangles in parallel, but couldn't easily compute a different function for each triangle.

Recall: GPU Memory

- The GPU needed high bandwidth access to large texture memory to draw textures on models in real time.
  - This memory has **high latency**
- Unlike multicore CPUs, which have deep cache hierarchies, the GPU used parallelism to **hide latency**
  - Example application: when a shader reaches a lookup into texture memory, the GPU “core” will request the data; while the request is resolving, it runs other threads.
  - But actually switching to an independent program would be too expensive, so instead it does same program/different data

Recall: TensorCores

- NVIDIA GPUs have special hardware accelerator compute units for matrix multiply

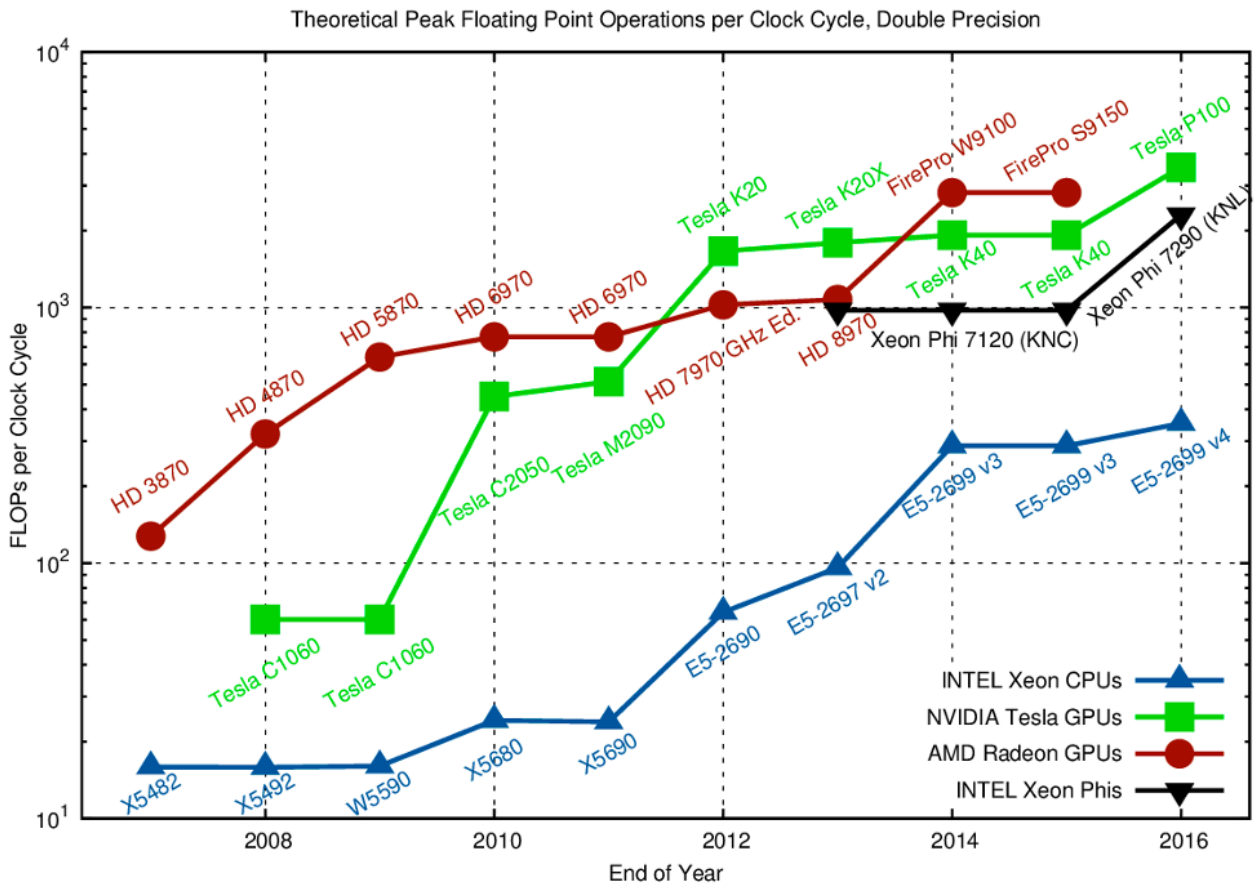
- Represents most of the FLOPs of the chip!
  - Note that these numbers report structured sparsity—real dense FLOPs are about half this
  - A relevant statistic here: about **500 fp8 FLOPs per byte** loaded from memory

Technical Specifications	
	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s

*With sparsity

FLOPS before TensorCores: GPU vs CPU

- FLOPS: **f**loating **p**oint **o**perations **p**er **s**econd



GPU FLOPS consistently exceed CPU FLOPS

From Karl Rupp's blog https://www.karlrupp.net/2016/08/flops-per-cycle-for-cpus-gpus-and-xeon-phis/

This was the best diagram I could find that shows trends over time.

Summary: GPU vs CPU

- CPU is a general purpose processor
  - Modern CPUs spend most of their area on deep caches
  - This makes the CPU a great choice for applications with random or non-uniform memory accesses

- GPU is optimized for
  - more compute intensive workloads
  - streaming memory models

Machine learning workloads are compute intensive and often support streaming data flows.

GPUs in machine learning

- Because of their high data parallelism, GPUs provide **an ideal substrate for large-scale numerical computation**.
  - In particular, GPUs can perform matrix multiplies very fast.
  - Just like BLAS on the CPU, there's an optimized library from NVIDIA "cuBLAS" that does matrix multiples efficiently on their GPUs.
  - There's even a specialized library of primitives designed for deep learning: cuDNN.

- Machine learning frameworks, such as TensorFlow and PyTorch, are designed to support computation on GPUs.
  - And training a deep net on a GPU can decrease training time by an order of magnitude.

So should we always use GPUs?

Will we always use GPUs?

Maybe not!

Challengers to the GPU

- **Hybrid GPU/CPU devices with AI acceleration**
  - Like Apple's M-series
- **Low-power devices (often hybrid)**
  - Like mobile-device-targeted chips
- **Configurable hardware like FPGAs**
- **AI Accelerators (NPUs)**
  - **Mostly designed to accelerate matrix-matrix multiply**
  - **Three broad categories:**
    - **training-focused**
    - **inference-focused**
    - **general-purpose**
- **Accelerators special-cased to specific applications**

Machine Learning Accelerators

Examples and tradeoffs

Effect of Hardware on Statistical Performance

- Sometimes, when we change device, we expect the same results: same learned model, same accuracy. **But this is not always the case. Why?**

- One reason: FP arithmetic is not associative, so reordering to get better performance on different hardware can change the results slightly!
  - This can even be the case for multiple runs on the same hardware.
- Another reason: randomness.
  - Different hardware platforms could use different pseudo-random generators or use random numbers in a different order.
- Another reason: hardware can do approximation to trade-off precision for hardware efficiency.
  - E.g. if one device runs on 16-bit floats, it can give different results than another device that runs on bfloats

Effect of Hardware on Statistical Performance

- Sometimes, when we change device, we expect the same results: same learned model, same accuracy. **But this is not always the case. Why?**

- **Generally, we expect specialized ML hardware to produce learned models that are similar in quality, but not necessarily exactly the same, as baseline chips (CPU/GPU).**

- Same thing for inference outputs!

Programming ML Hardware

- One major issue when developing new hardware for ML, and the question we should always be asking: **how is the device programmed?**
- To be useful for DNN applications, we should be able to map a high-level program written in PyTorch to run on the hardware.
- For most accelerators on the market today, this "just works" — all you gotta do is move to the device.
  - E.g. TPU (right)

```
import torch
import torch_xla
```

```
dev = torch_xla.device()
```

```
x = torch.randn(1024,1024,device=dev,dtype=torch.float16)
y = torch.randn(1024,1024,device=dev,dtype=torch.float16)
```

```
x @ y.t()
```

```
tensor([[-33.2500, -47.7812,  -5.5039,  ..., -15.1016,  52.3
        [-11.1875,  -0.2277,  27.4844,  ...,  32.3750,   5.2
        [  9.1953,  18.9688,  24.0000,  ...,  21.1562, -10.4
        ...,
        [-25.3750,   5.2500,   4.1016,  ...,  29.7500,  -7.4
        [-20.4375, -38.5312, -31.6719,  ...,   5.0742,  19.2
        [  1.1514,  23.1094, -41.6250,  ...,  27.3594,  11.7
       device='xla:0', dtype=torch.float16)
```

GPU as ML accelerator

- The only real distinction between GPUs and ML accelerators is that GPUs weren't originally designed for AI/ML.
  - But there's nothing in the architecture that separates GPUs from all purpose-built ML accelerators.
- ...although individual accelerators do usually have features that GPUs lack.
- As ML tasks capture more of the market for GPUs, GPU designers have been adjusting their architectures to fit ML applications.
  - For example, by supporting low-precision arithmetic.
- As GPU architectures become more specialized to AI tasks, it becomes more accurate to think of them as ML accelerators.

FPGA as ML accelerator

- All computer processors are basically integrated circuit: electronic circuits etched on a single piece of silicon.

- Usually this circuit is fixed when the chip is designed.

- A **field-programmable gate array** or FPGA is a type of chip that allows the end-user to reconfigure the circuit it computes in the field (hence the name).

FPGA as ML accelerator (continued)

FPGAs consist of an array of programmable circuits that can each individually do a small amount of computation, as well as a programmable interconnect that connects these circuits together. The large number of programmable gates in the FPGA makes it a **naturally highly parallel device**.

- You can program it by specifying the circuit you want it to compute in terms of logic gates: basic AND, OR, NOT, etc.
  - Although in practice higher-level languages like Verilog are used.
- This doesn't actually involve physical changes to the circuit that's actually etched on the physical silicon of the FPGA: that's fixed. Rather, the FPGA constructs a logical circuit that is reconfigurable.
  - FPGAs were used historically for circuit simulation.

Why would we want to use an FPGA instead of a GPU/GPU?

- An important property of FPGAs that distinguishes them from CPUs/GPUs: you can choose to have data flow through the chip however you want!
  - Unlike CPUs which are tied to their cache-heirarchy-based data model, or GPUs which perform best under a streaming model.
- FPGAs often use less power to accomplish the same work compared with other architectures.
  - But they are also typically slower.

Why would we want to use an FPGA instead of an ASIC?

- Pro for FPGA: Much cheaper to program and FPGA than the design an ASIC.
- Pro for FPGA: A single FPGA costs much less than the first ASIC you synthesize.
- Pro for FPGA: FPGA designs can be adjusted on the fly.
- Pro for ASIC: The marginal cost of producing an additional ASIC is lower if you really want to synthesize millions or billions of them.
- Pro for ASIC: ASICs can typically achieve higher speed and lower power.

Users of FPGAs for Machine Learning

Early use: Microsoft's Project Capatult/Project Brainwave.

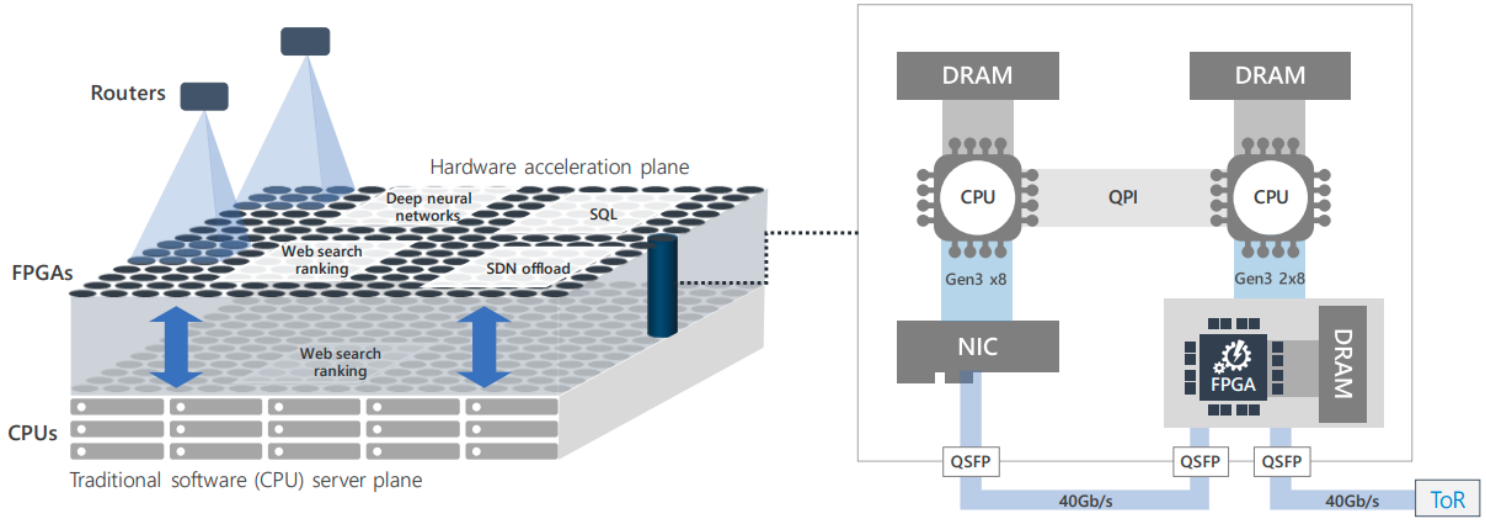


Figure 1. The first generation of Catapult-enhanced servers in production (right) consists of dual Xeon CPUs with a PCIe-attached FPGA. Each FPGA sits in-line between the 40Gbps server NIC and the TOR, enabling in-situ processing of network packets and point-to-point connectivity with up to hundreds of thousands of other FPGAs at datacenter scale. Multiple FPGAs can be allocated as a single shared hardware microservice with no software in the loop (left), enabling scalable workloads and better load balancing between CPUs and FPGAs.

from the paper *"Serving DNNs in RealTime at Datacenter Scale with Project Brainwave"*

An example of a designed-for-ML accelerator

The Tensor Processing Unit (TPU)

Google's Tensor Processing Unit (TPU) in 2015 was one of the first specialized architectures for machine learning and AI applications.

- The original version focused on **fast inference via high-throughput 8-bit arithmetic.**
- Most of the chip is dedicated to **accelerating 8-bit integer dense-matrix-dense-matrix multiplies**
  - Note that even though the numbers it multiples are in 8-bit, it uses 32-bit accumulators to sum up the results.
  - This larger accumulator is common in ML architectures that use low precision.
- ...with a little bit of logic on the side to apply activation functions. **The second- and third-generation TPUs were designed to also support training and can calculate in float/bfloat16.**

Now ML accelerators are found in the cloud!

E.g. TPUs:

Cloud TPU version	Description	Availability
Ironwood	Our most powerful and efficient TPU yet, for the largest scale training and inference	Ironwood TPU will be general available in Q4, 2025
Trillium	Sixth-generation TPU. Improved energy efficiency and peak compute performance per chip for training and inference	Trillium is generally available in North America (US East region), Europe (West region), and Asia (Northeast region)
Cloud TPU v5p	Powerful TPU for building large, complex foundational models	Cloud TPU v5p is generally available in North America (US East region)
Cloud TPU v5e	Cost-effective and accessible TPU for medium-to-large-scale training and inference workloads	Cloud TPU v5e is generally available in North America (US Central/East/South/ West regions), Europe (West region), and Asia (Southeast region)

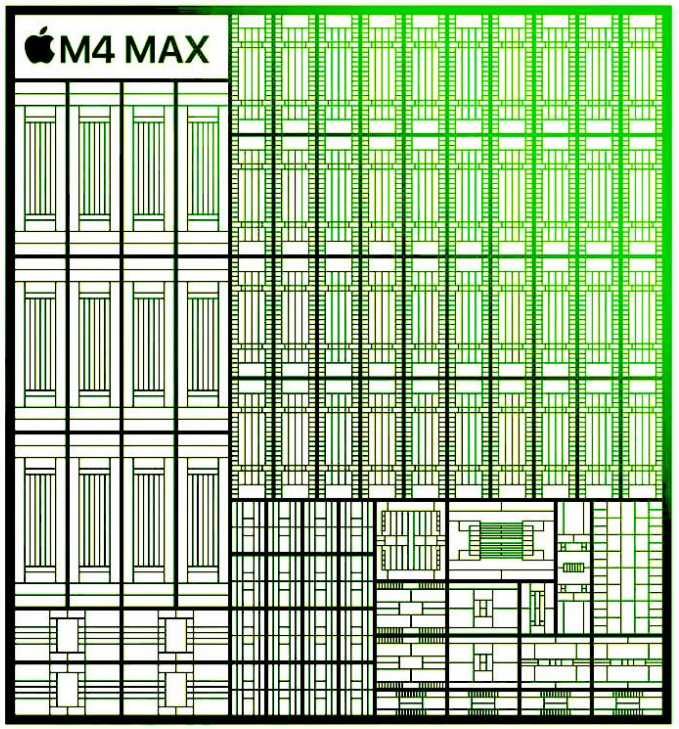
How do we program a TPU

- Essentially the same as for a GPU

- Google supports running ML kernels on TPUs.
  - A TPU is just another accelerator the ML framework can compute on and store data on
  - This makes it (relatively) easy to train and infer deep neural networks using tools you're already familiar with.

Hybrid Systems

- Recent systems-on-a-chip developed for personal devices (laptops, phones) now tend to include AI accelerators.
  - You can think of this as like a CPU+GPU combo
- Examples include the M-Series chips from Apple



How to choose your accelerator?

- Different accelerators provide different trade-offs

- Need to ask yourself some questions
  - Am I running training or inference?
  - What is my budget?
  - Where am I running? Am I limited to one cloud provider? Am I running on an edge device?
  - What did other people with similar tasks use?
  - Do I care more about throughput or latency? Or energy?

Questions?