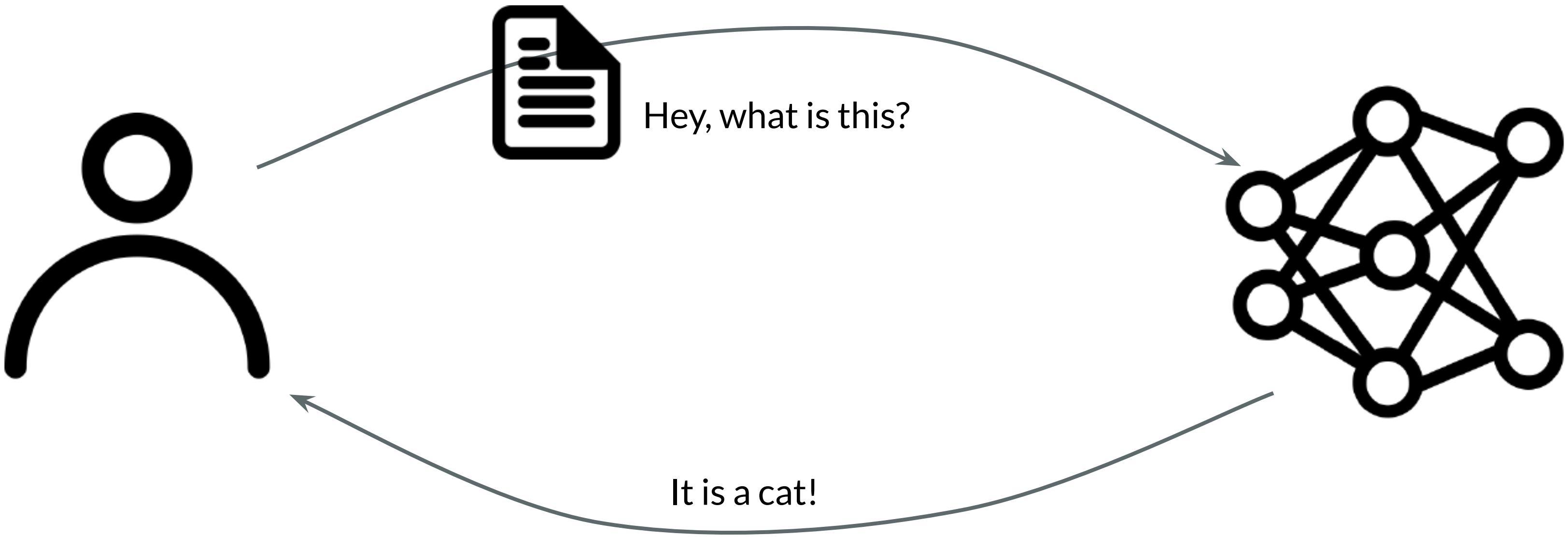
Deep learning systems deployment. How to do inference as fast as hell.

Alexey Kosmachev

So we have trained ML model. What's next?

Simple setup: user requests prediction for some data



What can go wrong with model.predict(...) ?

Standard method **predict** in your favourite ML library/framework could not be the best choice for **production**

- Heavy loaded by python bindings, training modules, etc
- Naive computation on CPU
- Bad performance on parallel queries

How to improve performance?

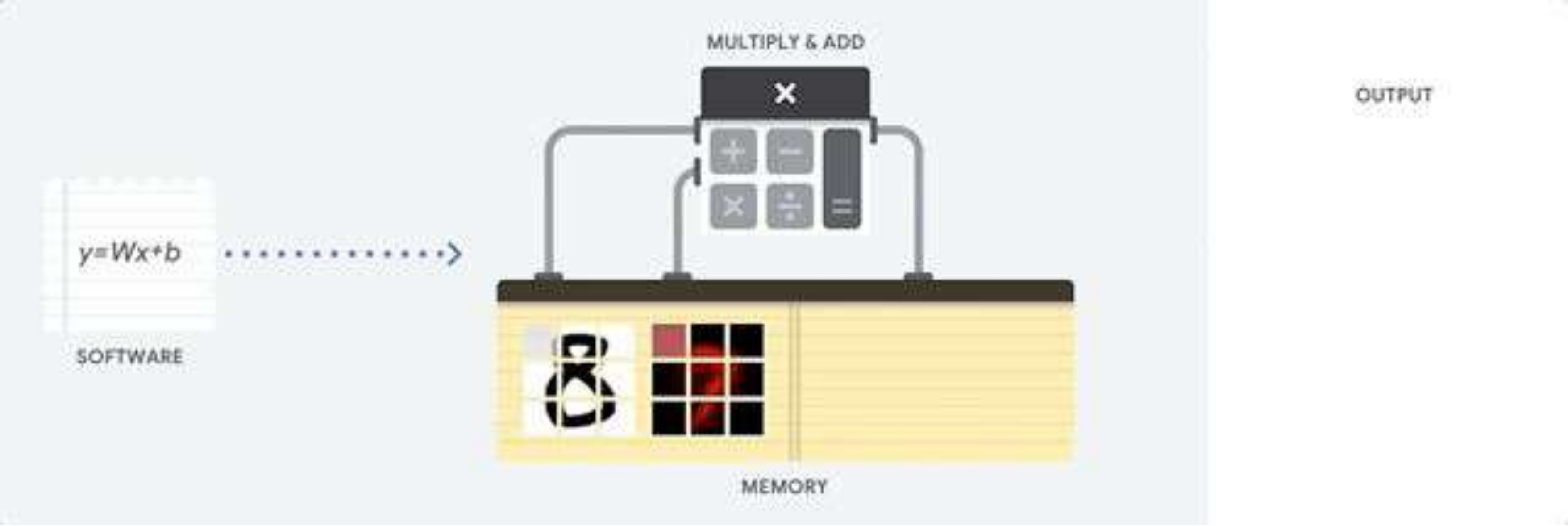
Level 1: Make the most from your **Hardware**

All sorts of processor units

There is **no one main processor** to rule them all, there are a lot of different chips with **plenty of architectures**

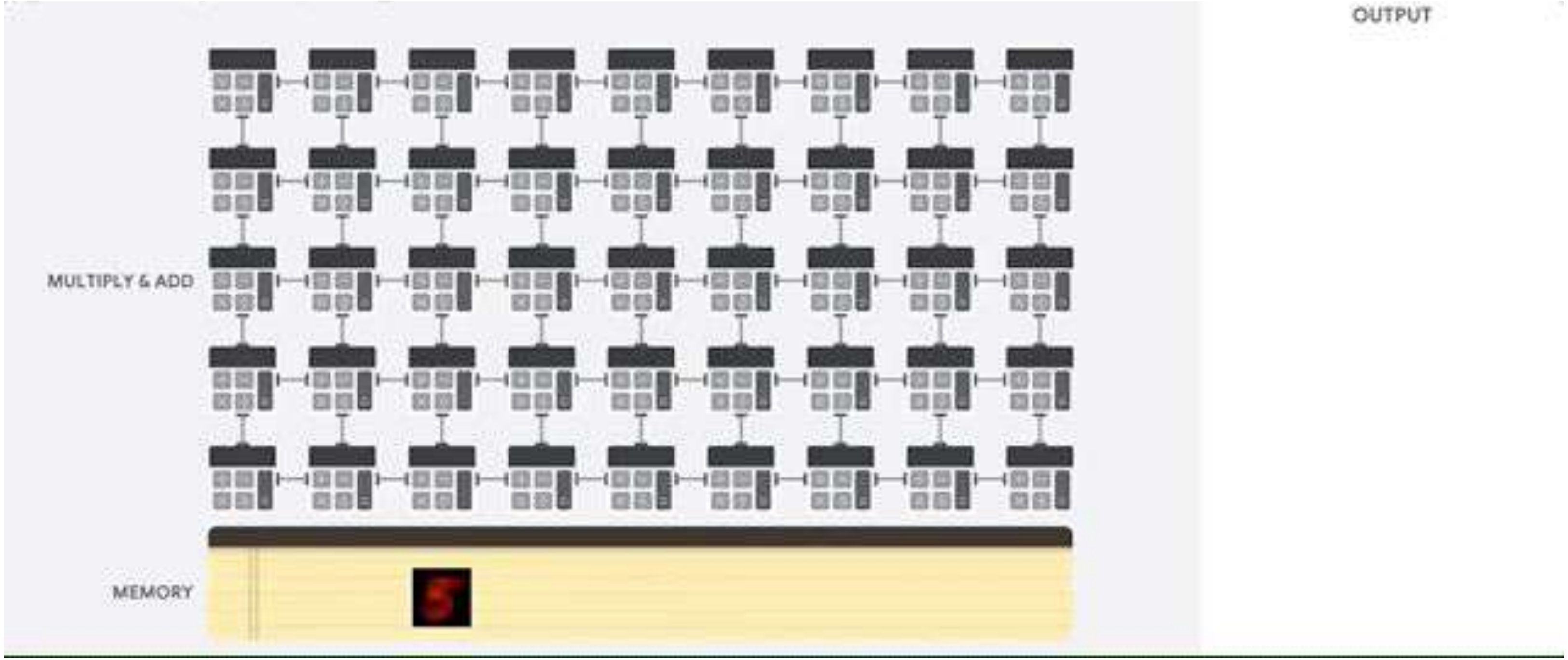
- CPU - x86, amd64, ARM, etc
- GPU - Tesla, Pascal, TeraScale, RDNA, etc
- + TPU, FPGA, VPU, NPU, etc

Units fancy schemas - CPU

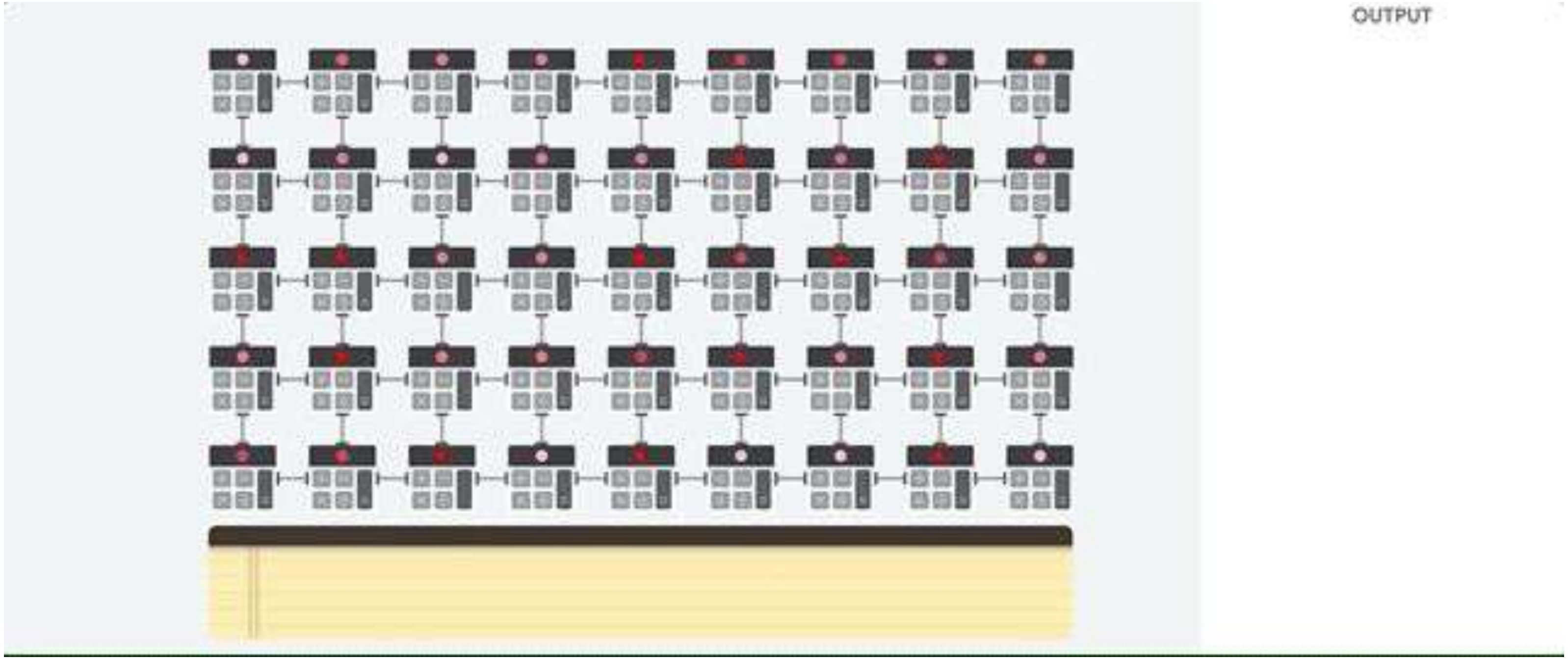


Units fancy schemas - GPU

Units fancy schemas - TPU (load)

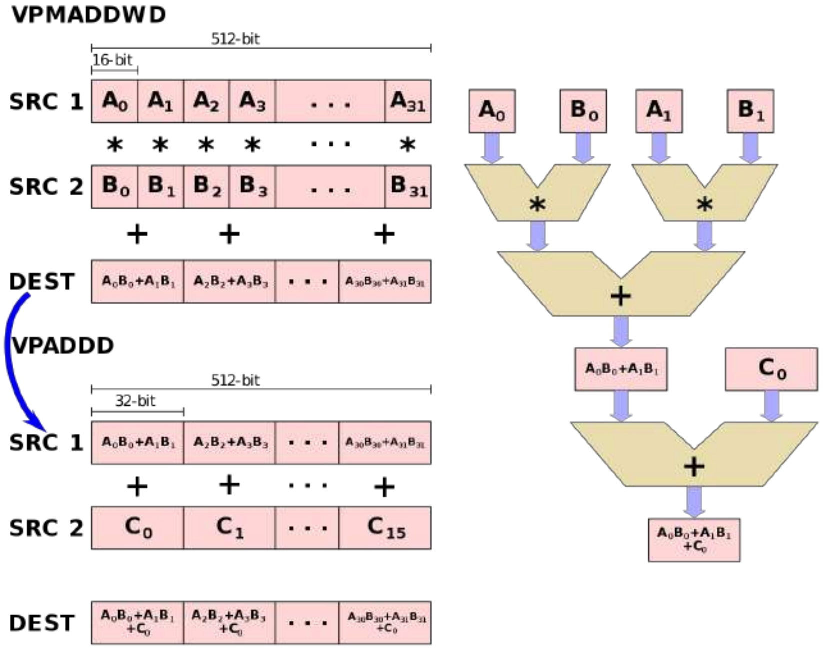


Units fancy schemas - TPU (compute)



see: https://cloud.google.com/tpu/docs/intro-to-tpu

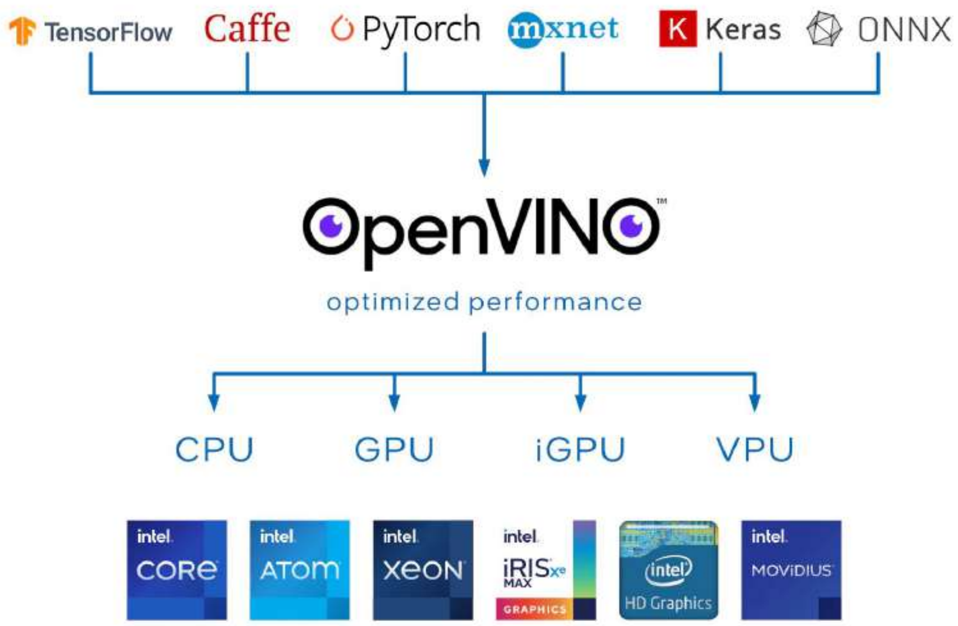
AVX-512 Vector Neural Network Instructions x86



see https://en.wikichip.org/wiki/x86/avx512_vnni

Meet the OpenVINO

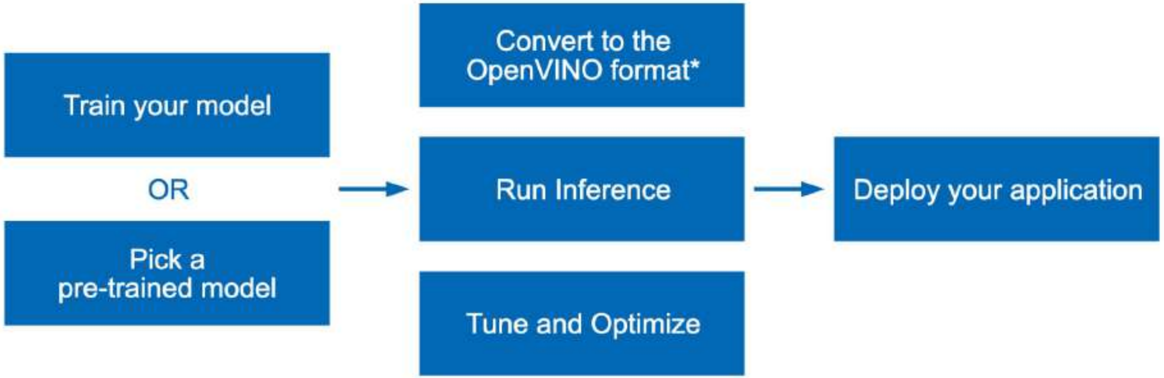
OpenVINO - Open Visual Inference and Neural network Optimization



Meet the OpenVINO

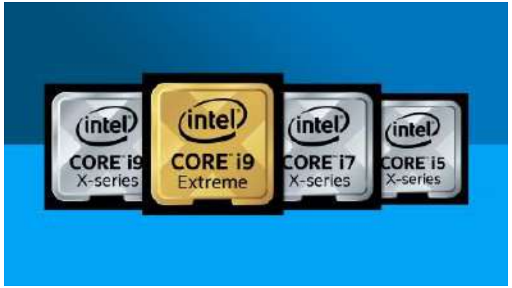
Two major components:

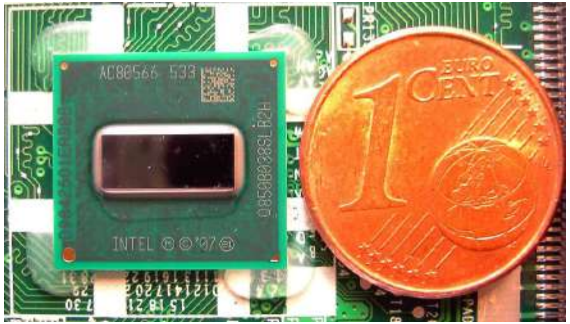
- Neural Network Compression Framework (NNCF)
- Inference Engine



Inference Engine

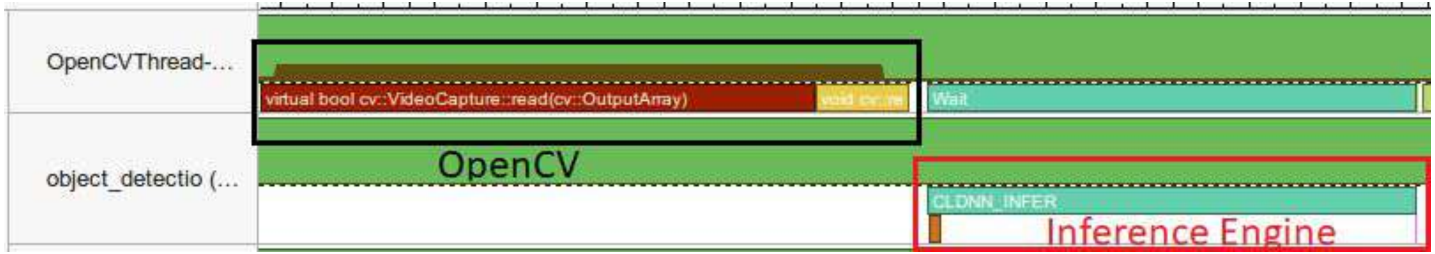
- Leveraging processor units architecture capabilities



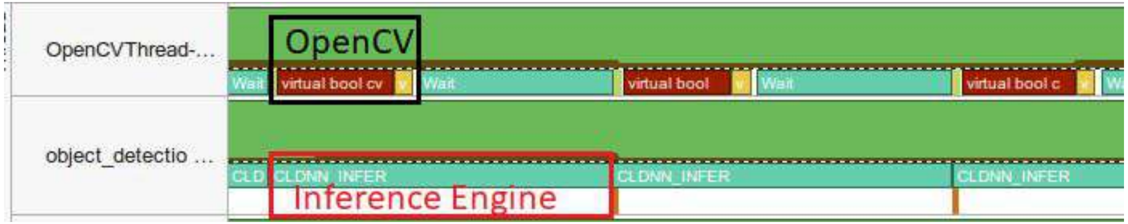


Inference Engine

- Leveraging processor units architecture capabilities
- Inference Engine Async



Sync Mode



Async Mode

Inference Engine

- Leveraging processor units architecture capabilities
- Inference Engine Async
- Throughput/Latency Mode for CPU

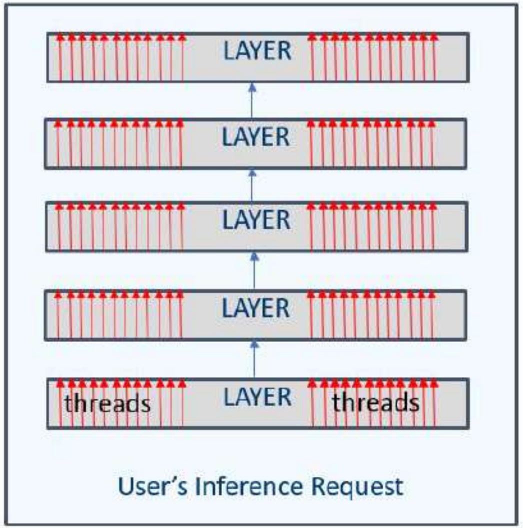
Inference Engine

Conventional Approach

Every CNN op is internally parallelized over **full** number of CPU cores => bad for non-scalable ops

A lot of sync between many threads =>overhead

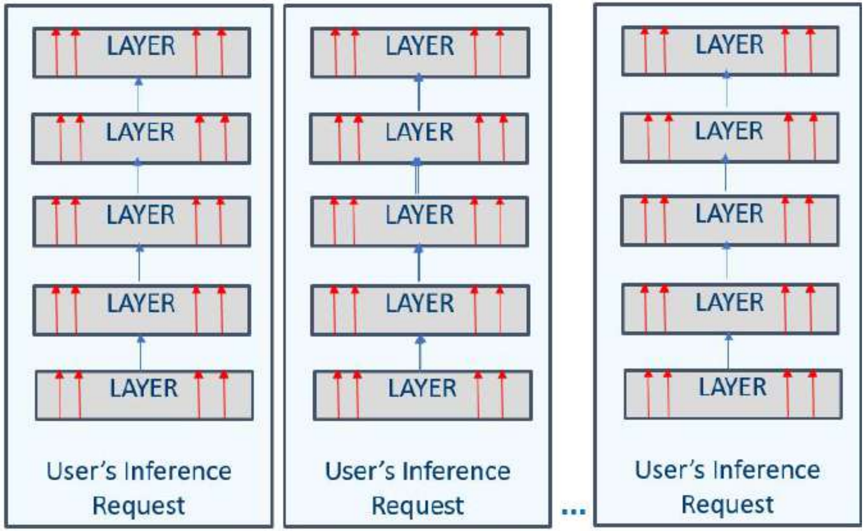
Only option to improve efficiency is batching



Streams

CPU cores are evenly distributed between **execution streams** (each 1-4 threads)

Less threads per stream => less sync, better locality, finer granularity



Requests are executed in parallel, each with small #threads

LAYER-WISE THE STREAMS IMPLY MUCH LESS SYNCH

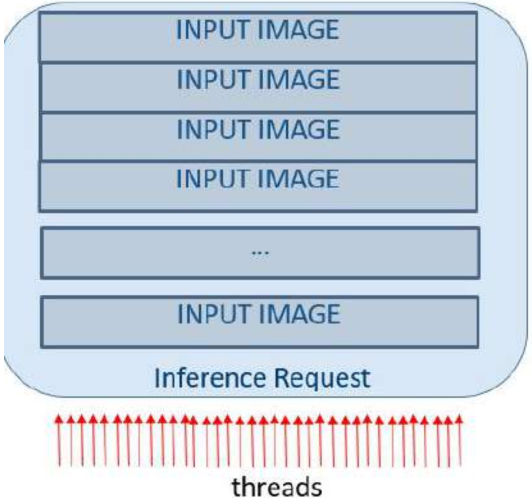
Inference Engine

Large Batch Approach

All threads are doing all inputs at once

Assumes all layers are parallelized well

"Fat" requests are executed one by one

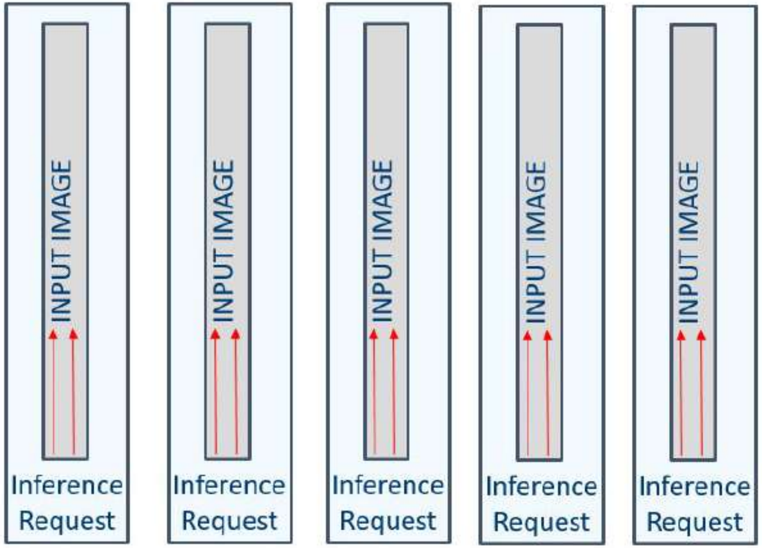


Streams

CPU cores are evenly distributed between **(execution) streams**

"Parallelize the outermost loop" rule of thumb

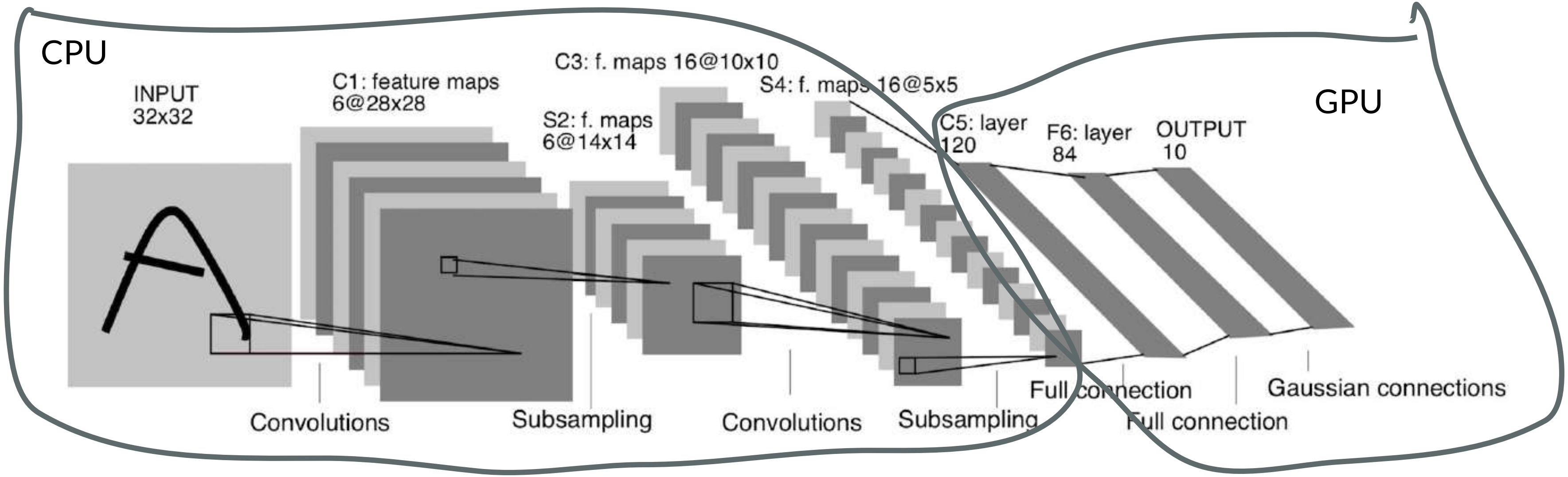
Individual requests are executed in parallel



INPUTS-WISE THE STREAMS ARE THE "TRANSPOSED" BATCH

Inference Engine

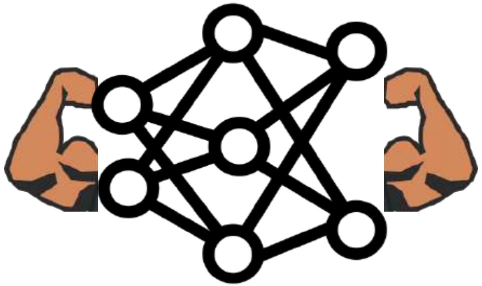
- Leveraging processor units architecture capabilities
- Inference Engine Async
- Throughput/Latency Mode for CPU
- Heterogeneous mode - single inference on different devices



How to improve performance?

Level 1: Make the most from your **Hardware**

Level 2: Make the most from your **Neural Network**



Repack NN

TorchScript - compiled language, optimized for run torch models

- No python dependency
  - Can be embedded into other native apps (e.g. in C++)
  - Do not lock GIL in python
- Faster execution due to statically typed, jit-compiled runtime

Repack NN

ONNX - Open Neural Network Exchange, unified format with common set of operators for NN

- Engine agnostic format enables you to convert any NN format to any other
- Can run on almost any inference engine for NN

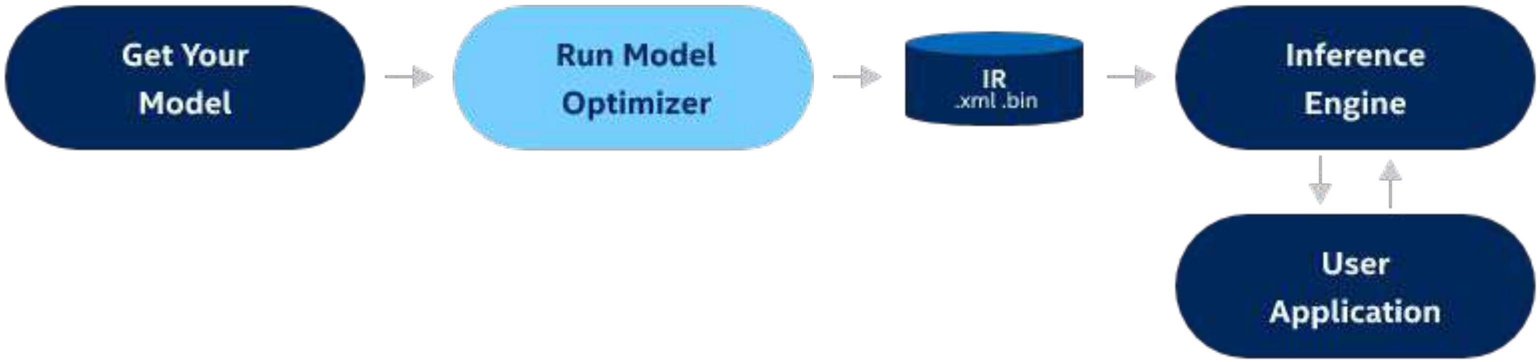
ONNX

Neural Network Compression Framework

- Quantization
- Pruning

Neural Network Compression Framework

- Quantization
- Pruning
- Format - Intermediate Representation (IR)



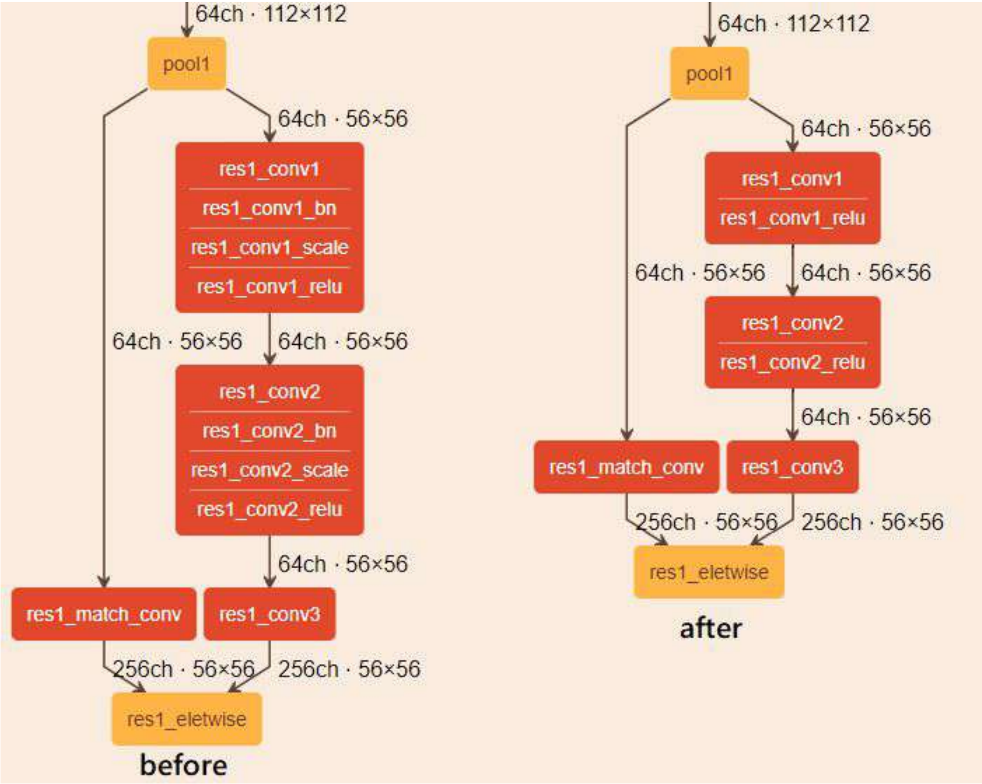
Neural Network Compression Framework

Intermediate Representation (IR)

- Encode NN for different precisions (FP32, FP16, INT8, etc)
- Plenty of optimization techniques
  - Linear Operations Fusing
  - Specialized optimizations (ResNet optimization, Grouped Convolution Fusing for TF, etc)

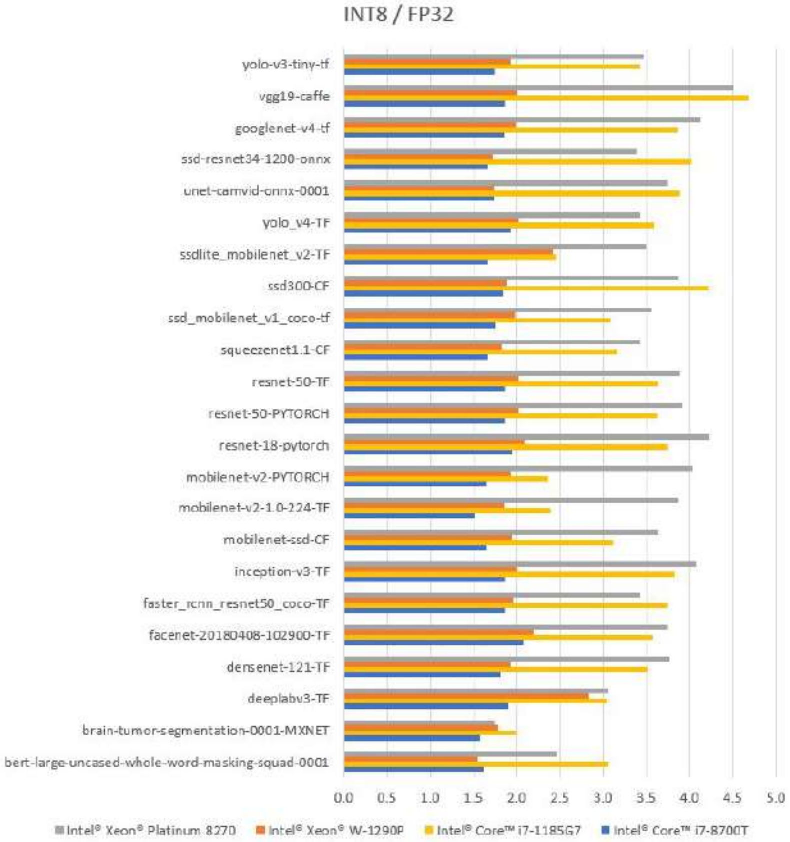
Neural Network Compression Framework

Batch Normalization and **Scale Shift** are just **Mul** → **Add** sequence which can be **fused** into one layer



Neural Network Compression Framework

Benchmarking **INT8 vs FP32** on different Intel Chips

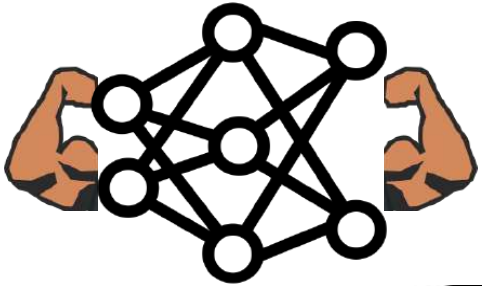


How to improve performance?

Level 1: Make the most from your **Hardware**

Level 2: Make the most from your **Neural Network**

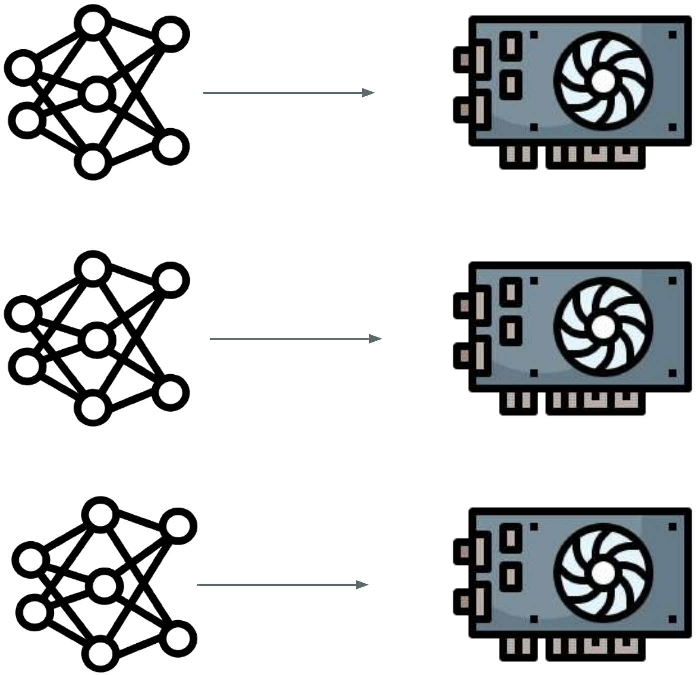
Level 3: Make the most from your **Cluster**



Is it enough to just buy a bunch of GPUs?

Naive and simple approach

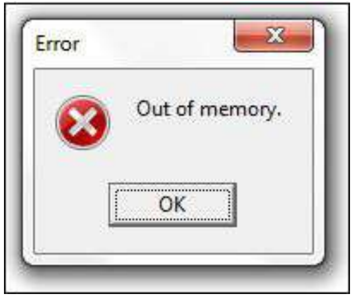
One model to **one** GPU



Is it enough to just buy a bunch of GPUs?

Problems with naive approach

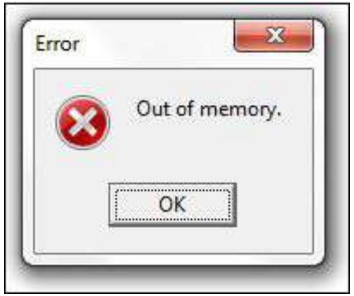
- Uncontrolled workload can lead to OOM

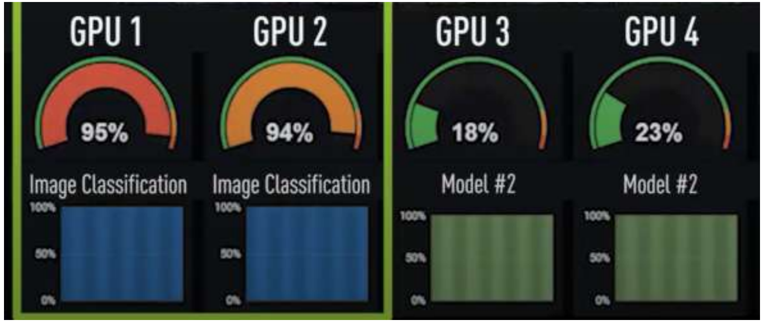


Is it enough to just buy a bunch of GPUs?

Problems with naive approach

- Uncontrolled workload can lead to OOM
- Overload of one GPU and idling of remained cluster

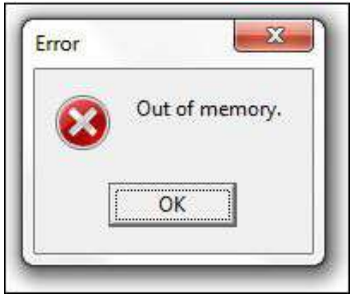


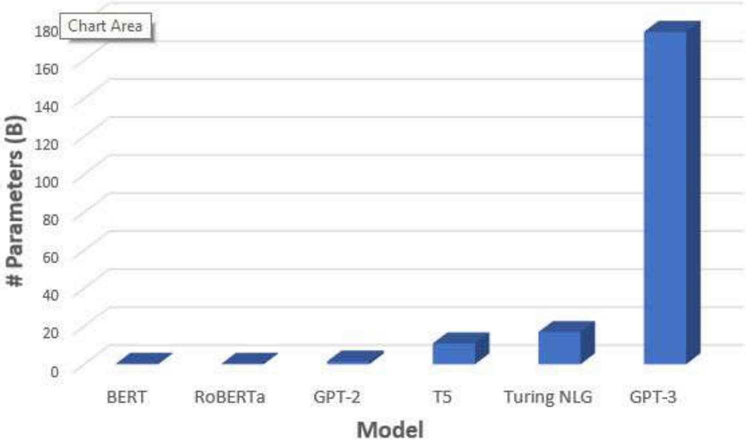


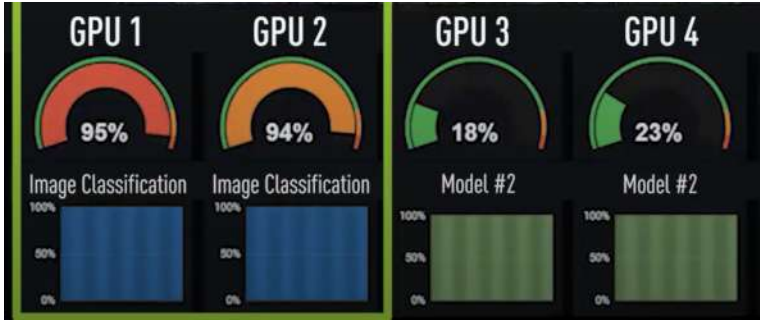
Is it enough to just buy a bunch of GPUs?

Problems with naive approach

- Uncontrolled workload can lead to OOM
- Overload of one GPU and idling of remained cluster
- Model can be bigger than one GPU

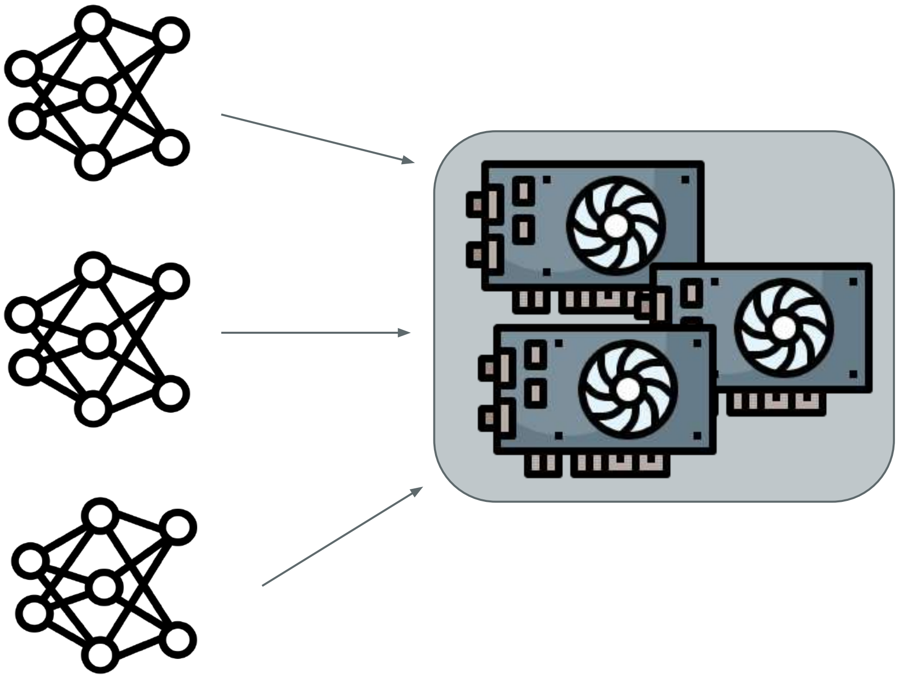




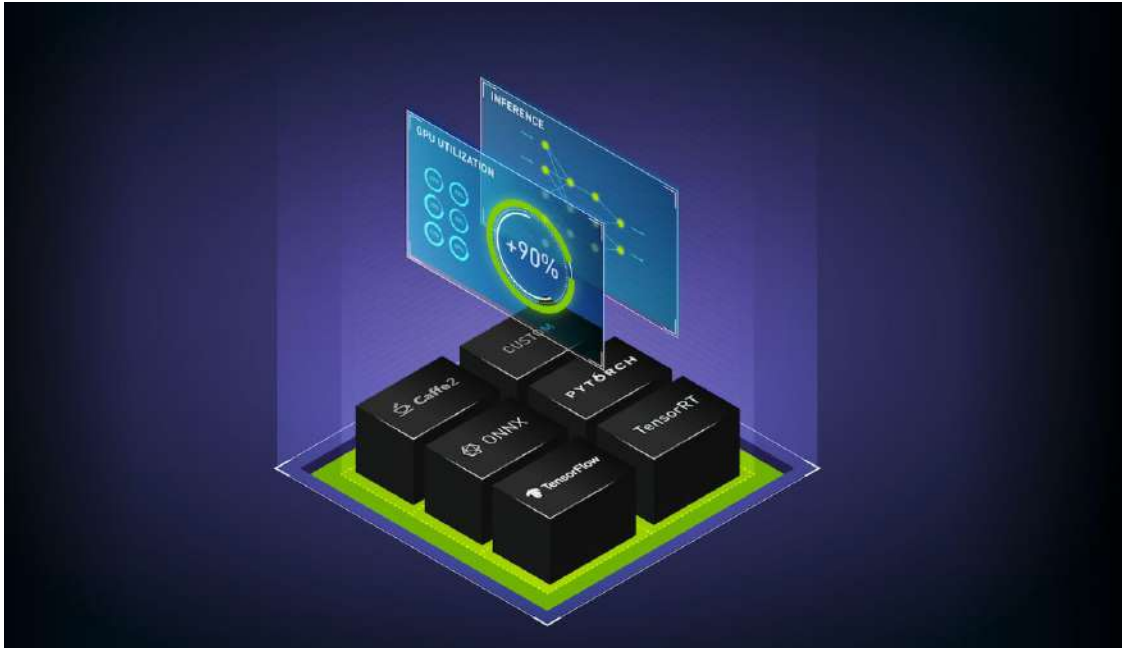


Is it enough to just buy a bunch of GPUs?

Solution - make one **mega GPU**
by **clustering** multiple simple GPUs



Meet the Nvidia Triton



Meet the Nvidia Triton

Features

- Spread models across all units (GPU & CPU)
- Combine individual inference requests together
- Use multi-node inference for large models (via NCCL)
- Autoscale cluster for workload

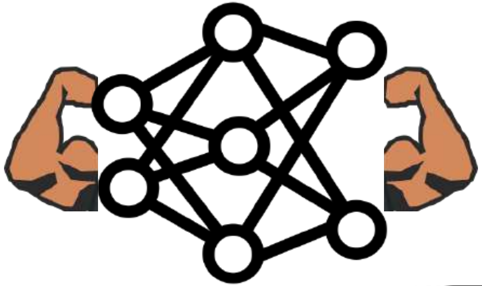
How to improve performance?

Level 1: Make the most from your **Hardware**

Level 2: Make the most from your **Neural Network**

Level 3: Make the most from your **Cluster**

Tip: move your NN to **client's device**



Native apps

- Core ML - library for iOS
  - Runs natively on mobile devices
  - Has libs for NLP, CV, Speech and Sound Analysis
  - Can be accelerated by BNNS and Metal framework
- ML Kit - library for Android
  - Runs natively on mobile devices
  - Has libs for common tasks - object detection, language processing - and for mobile specific - barcode scanning, digital inc recognition, selfie segmentation, etc
  - Can be accelerated by NPUs

Tensorflow JS

- Runs natively in web browser
- Has a lot of libraries by community
- Can be accelerated by WebGL

TensorFlow.js

Credits for icons

Neural Network by Ian Rahmadi Kurniawan from NounProject.com

User by Heztasia from NounProject.com

Document by Heztasia from NounProject.com