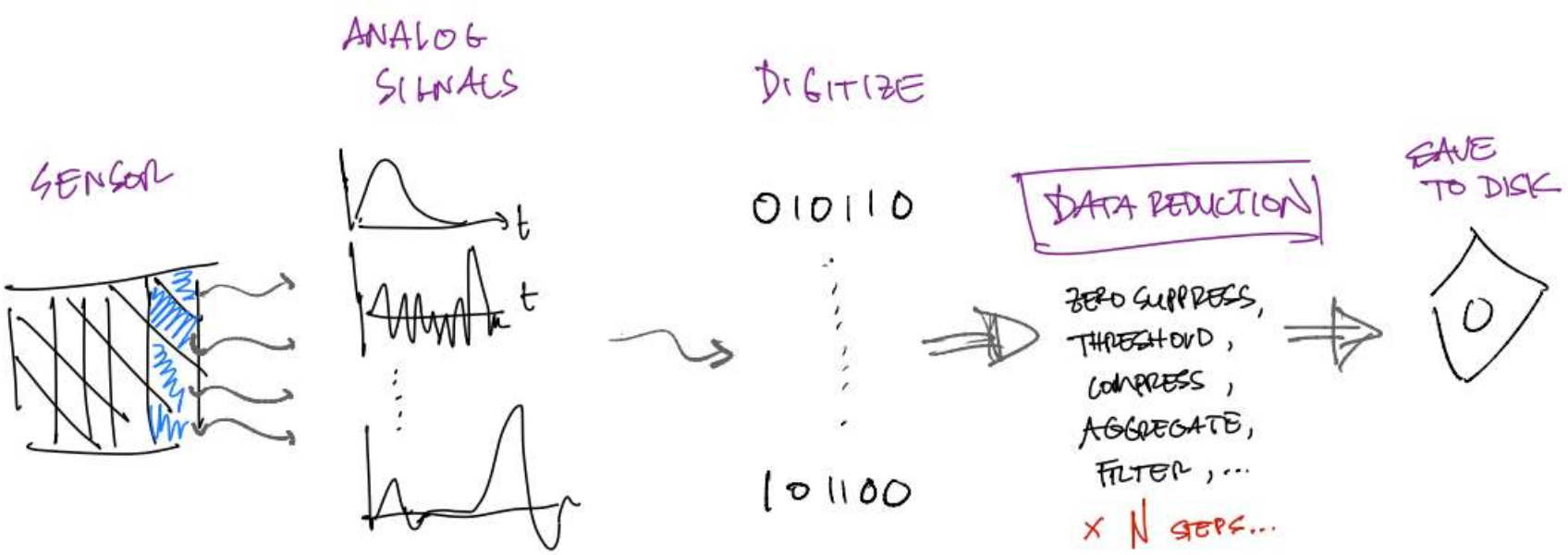
“Fast” ML for Science

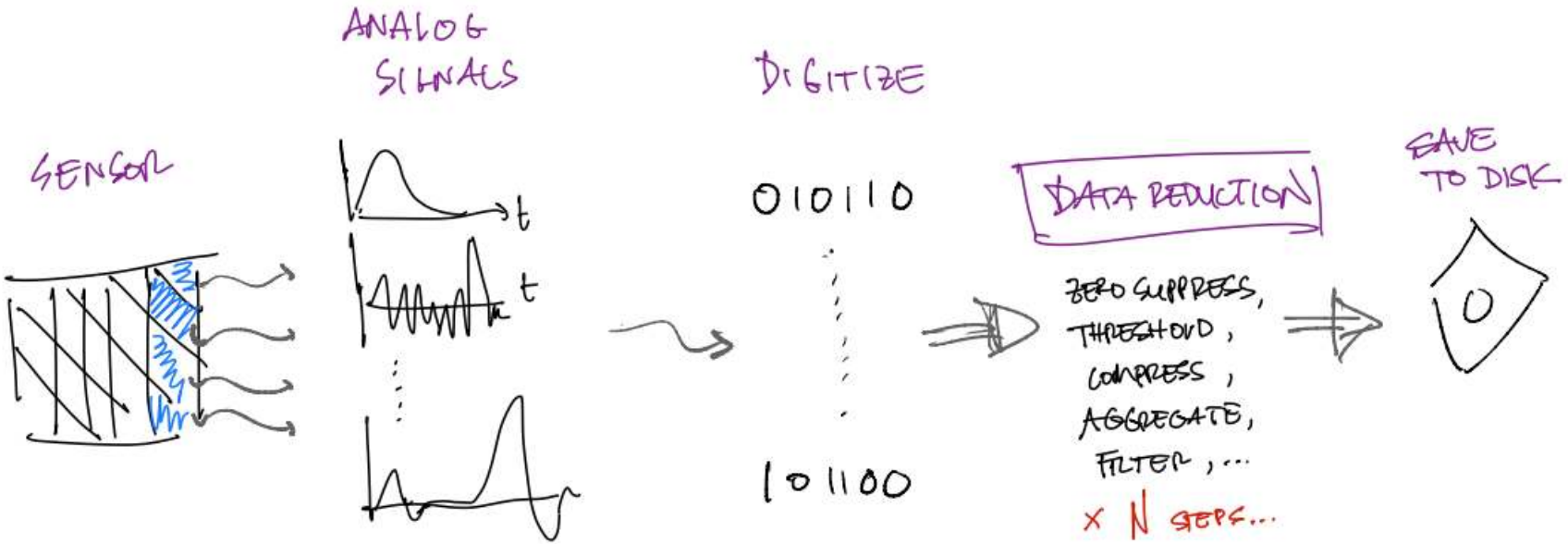
Nhan Tran, Fermilab & Duc Hoang, MIT
ICISE HW Camp
7 March 2024

With material from Aobo Li, Beomki Yeo, Javier Duarte

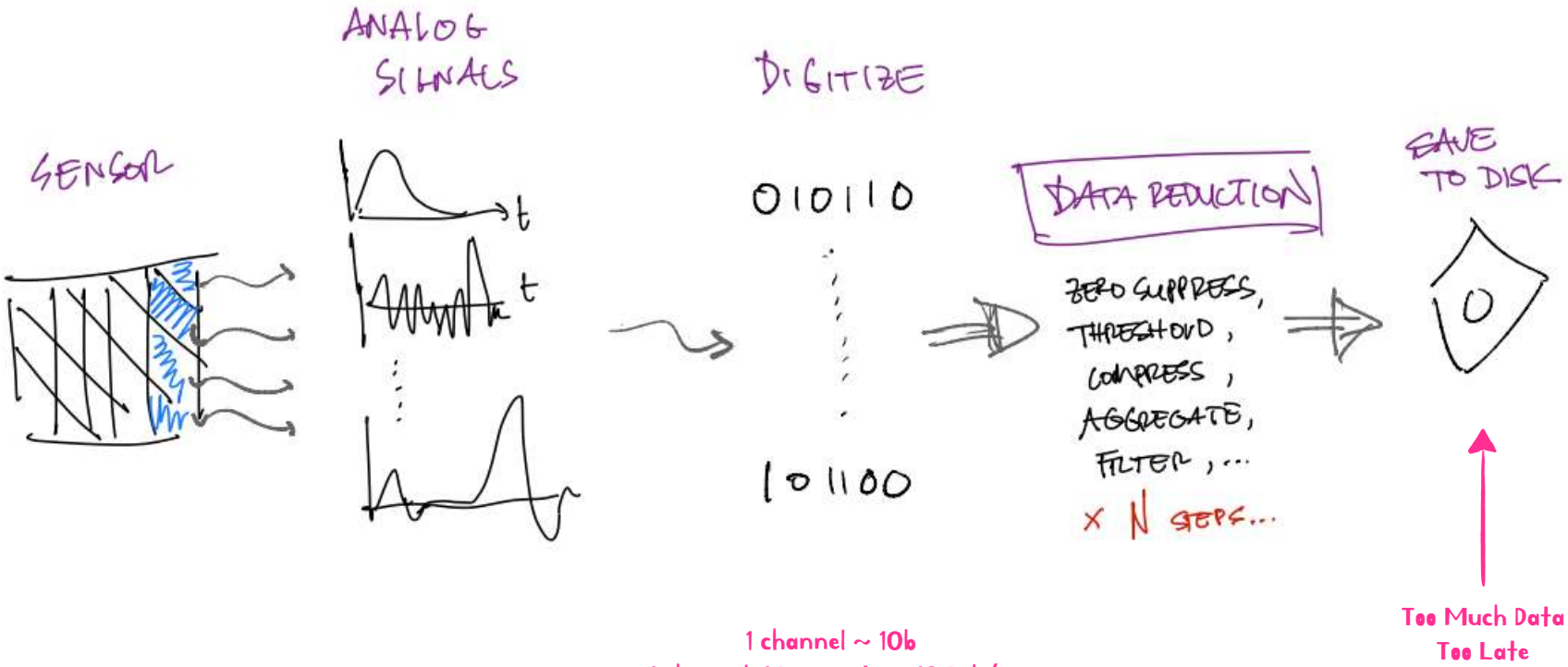
Table of contents

Fast ML for Science


SENSOR
ANALOG SIGNALS
t
t
DIGITIZE
010110
101100
DATA REDUCTION
ZERO SUPPRESS,
THRESHOLD,
COMPRESS,
AGGREGATE,
FILTER, ...
× N STEPS...
SAVE TO DISK

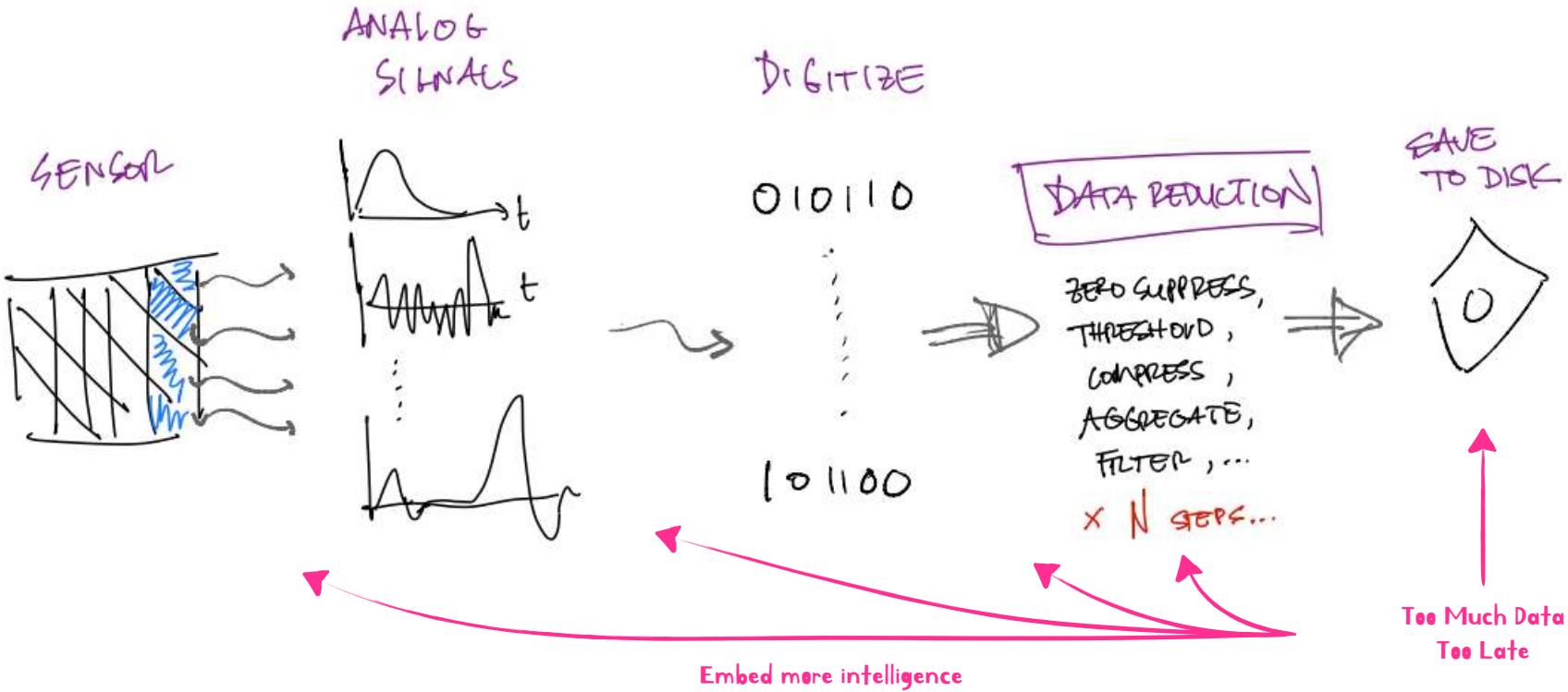


1 channel ~ 10b
1 channel, 1 MHz rate ~ 10 Mb/s
100k channels, 1 MHz rate ~ 1 Tb/s



1 channel ~ 10b
1 channel, 1 MHz rate ~ 10 Mb/s
100k channels, 1 MHz rate ~ 1 Tb/s

Too Much Data
Too Late


SENSOR
ANALOG SIGNALS
t
t
DIGITIZE
010110
101100
DATA REDUCTION
ZERO SUPPRESS,
THRESHOLD,
COMPRESS,
AGGREGATE,
FILTER, ...
X N STEPS...
SAVE TO DISK
Too Much Data
Too Late
Embed more intelligence

Fast ML for science and the extreme edge

"Scientific discoveries come from groundbreaking ideas and the capability to validate those ideas by testing nature at new scales - finer and more precise temporal and spatial resolution. This is leading to an explosion of data that must be interpreted, and ML is proving a powerful approach. The more efficiently we can test our hypotheses, the faster we can achieve discovery. To fully unleash the power of ML and accelerate discoveries, it is necessary to embed it into our scientific process, into our instruments and detectors."

Applications and Techniques for Fast Machine Learning in Science
https://doi.org/10.3389/fdata.2022.787421

The Need For Speed

The Need For Speed

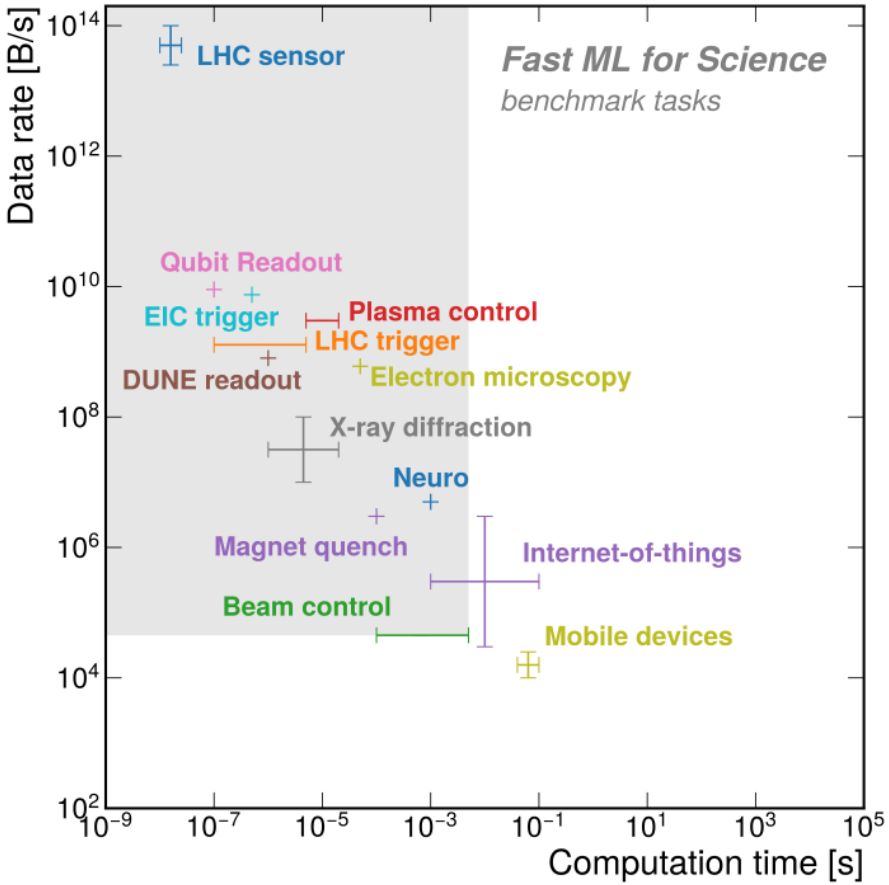
Benchmarks bring innovation

The Fast ML for Science community aims to bring **seemingly different domains** together to develop **techniques, tools, and platforms** for challenges that **far outpace industry.**

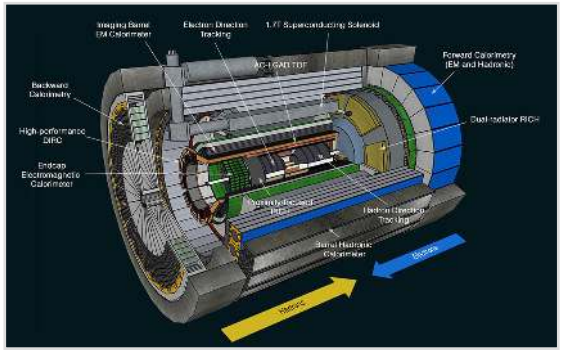
The Need For Speed

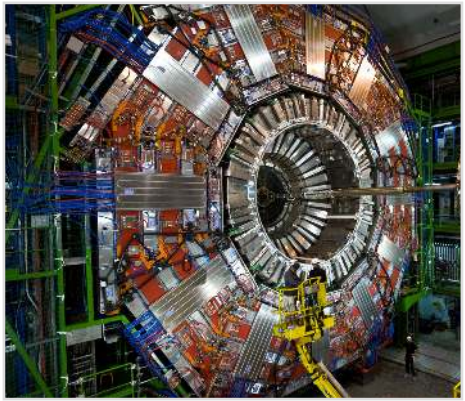
Benchmarks bring innovation

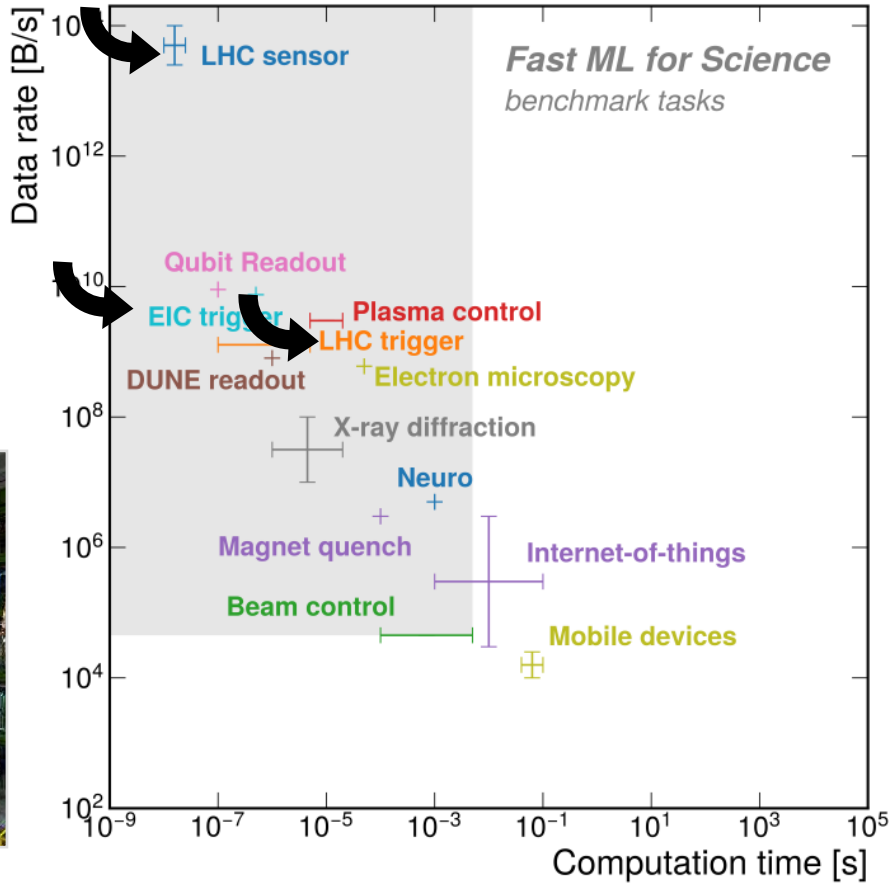
The Fast ML for Science community aims to bring **seemingly different domains** together to develop **techniques, tools, and platforms** for challenges that **far outpace industry.**



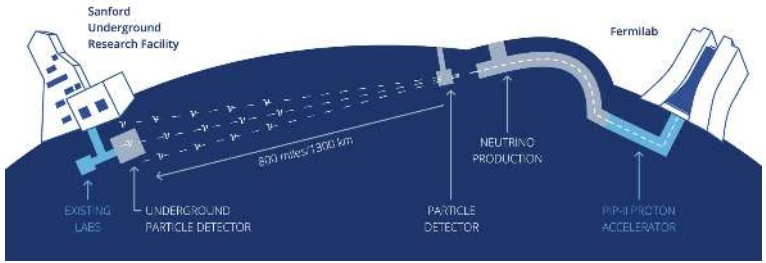
The Need For Speed

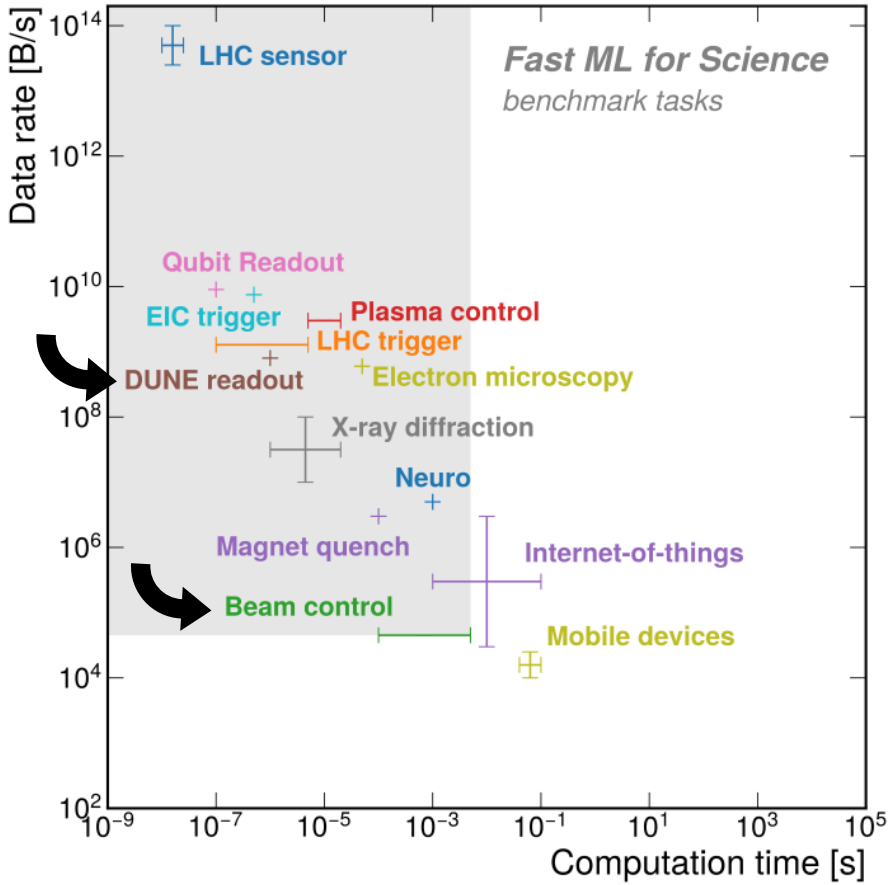




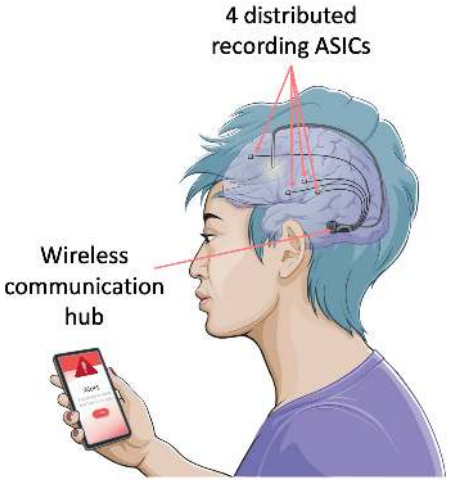


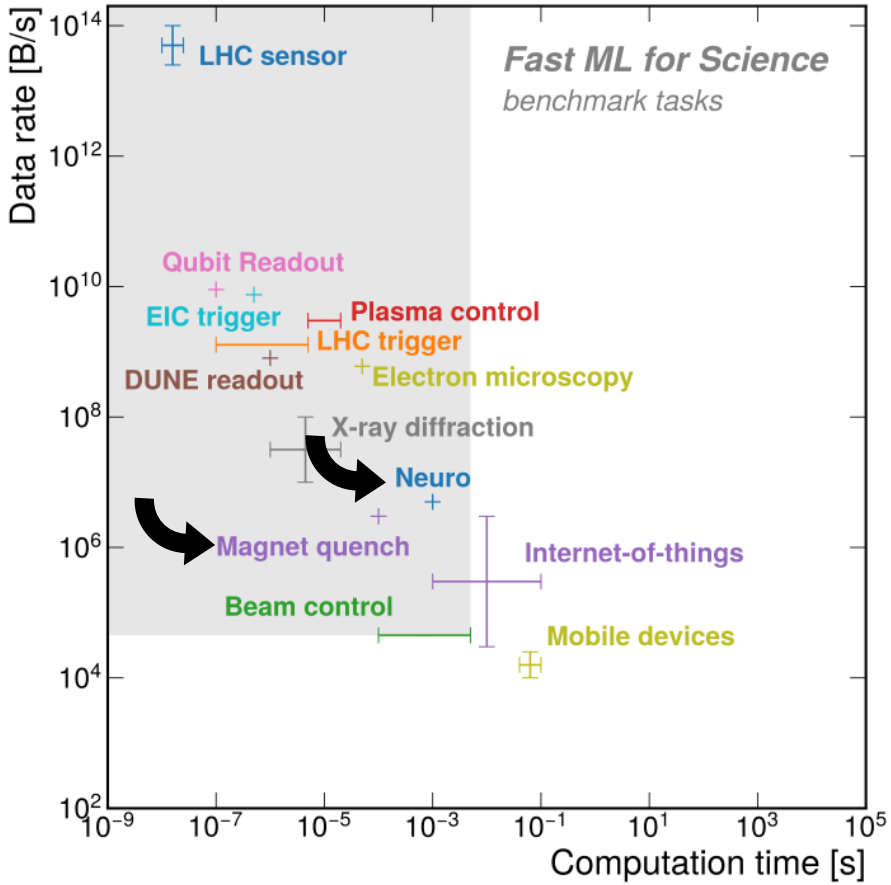
The Need For Speed



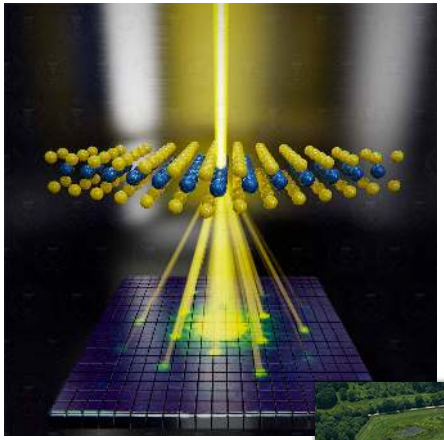


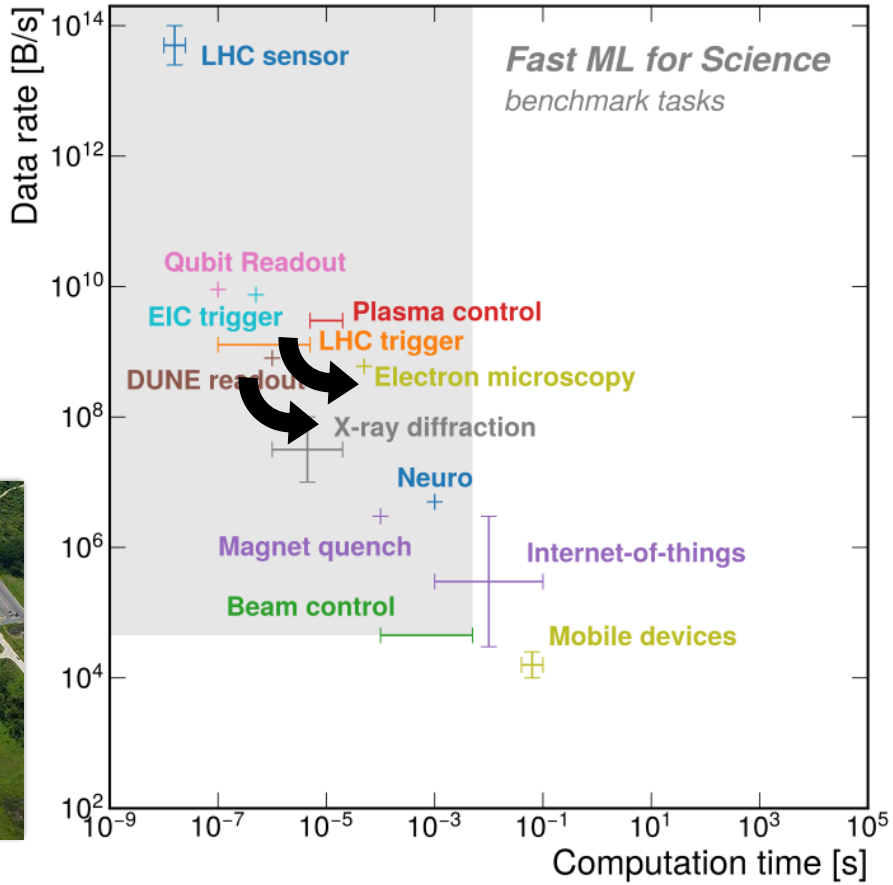
The Need For Speed



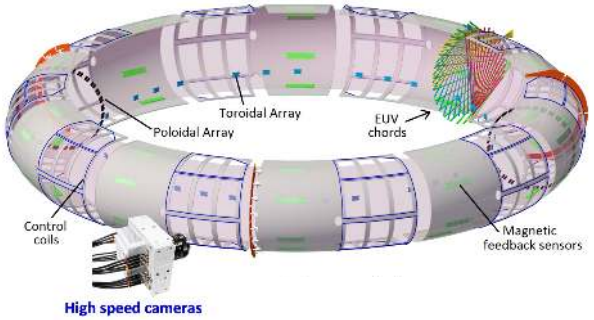


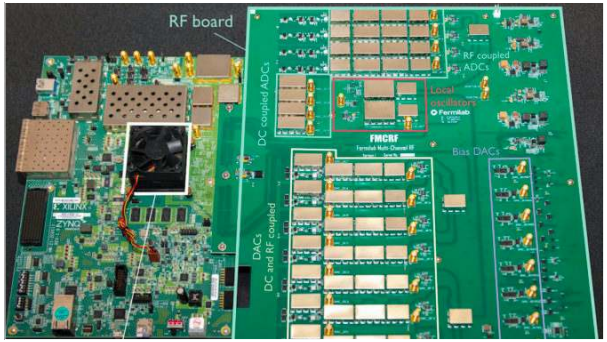
The Need For Speed

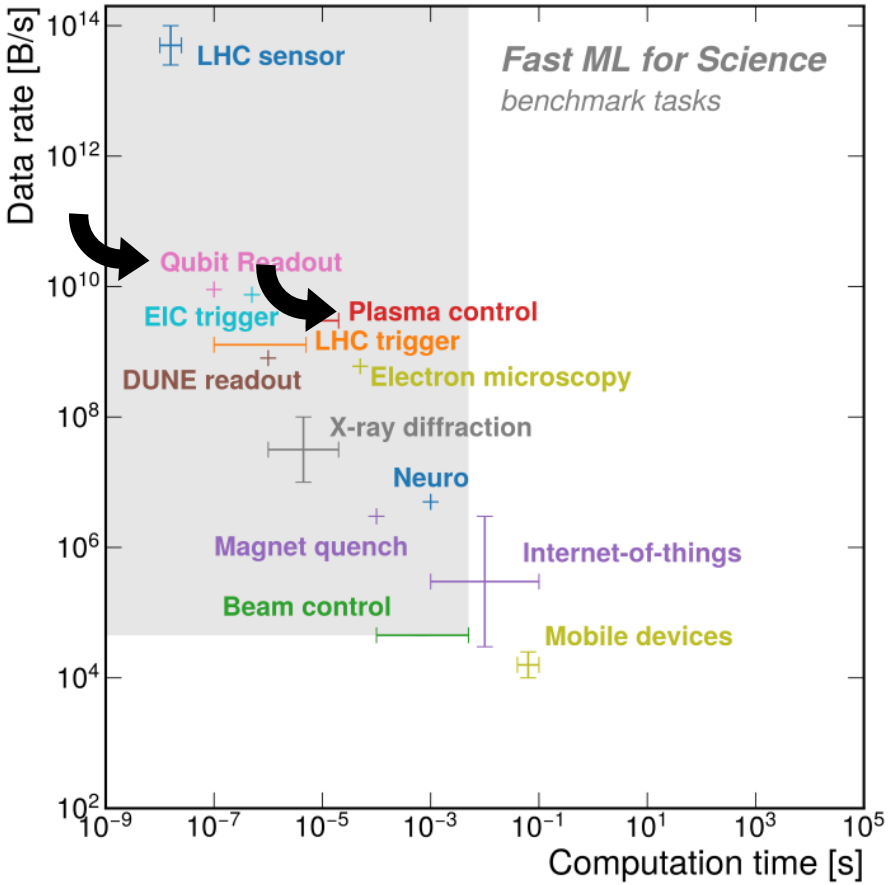




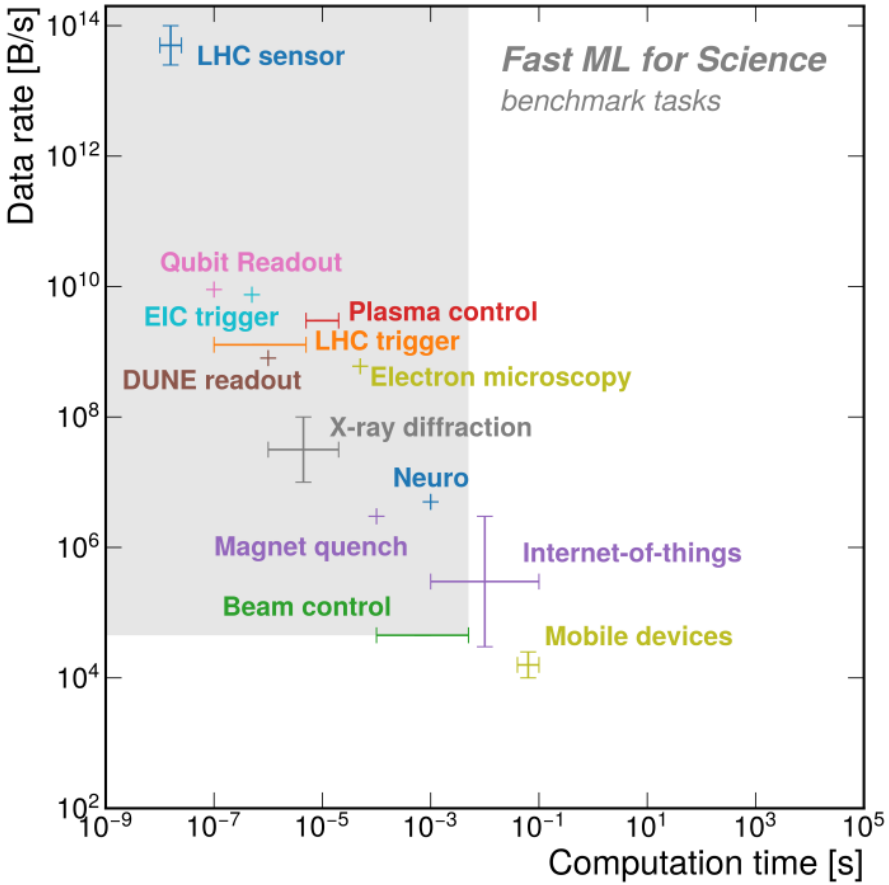
The Need For Speed



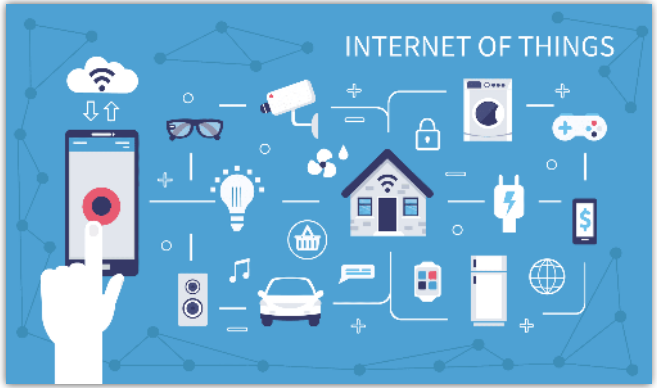




The Need For Speed



The Need For Speed



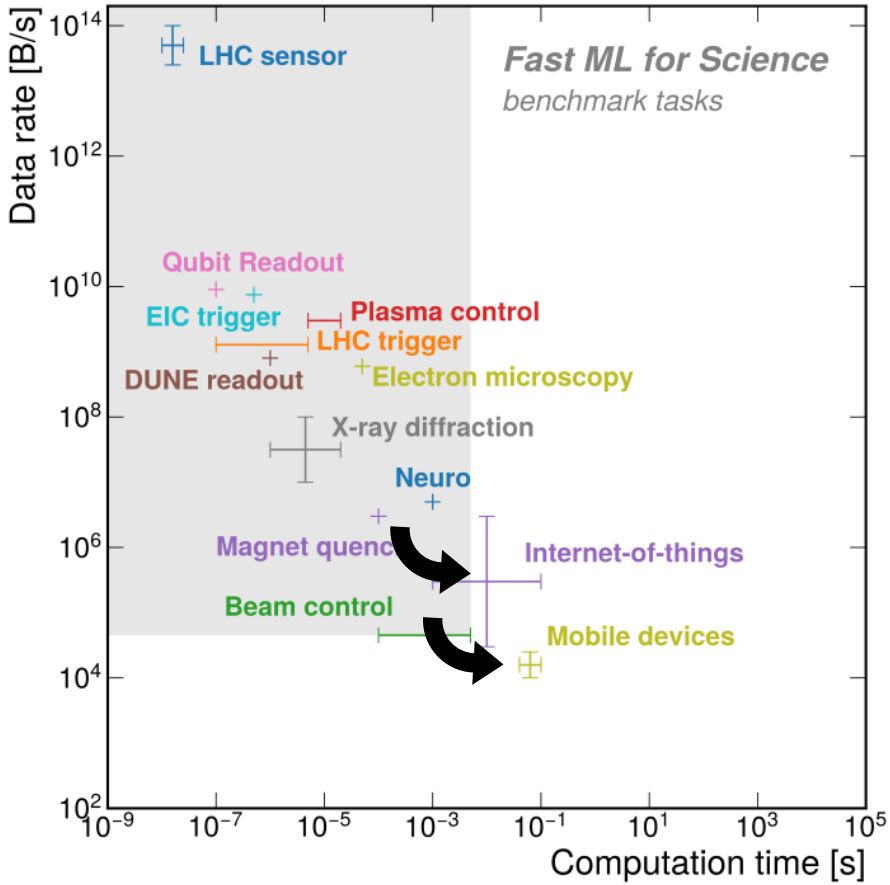
MLCommons launches machine learning benchmark for devices like smartwatches and voice assistants

by Ben Wodecki 6/16/2021

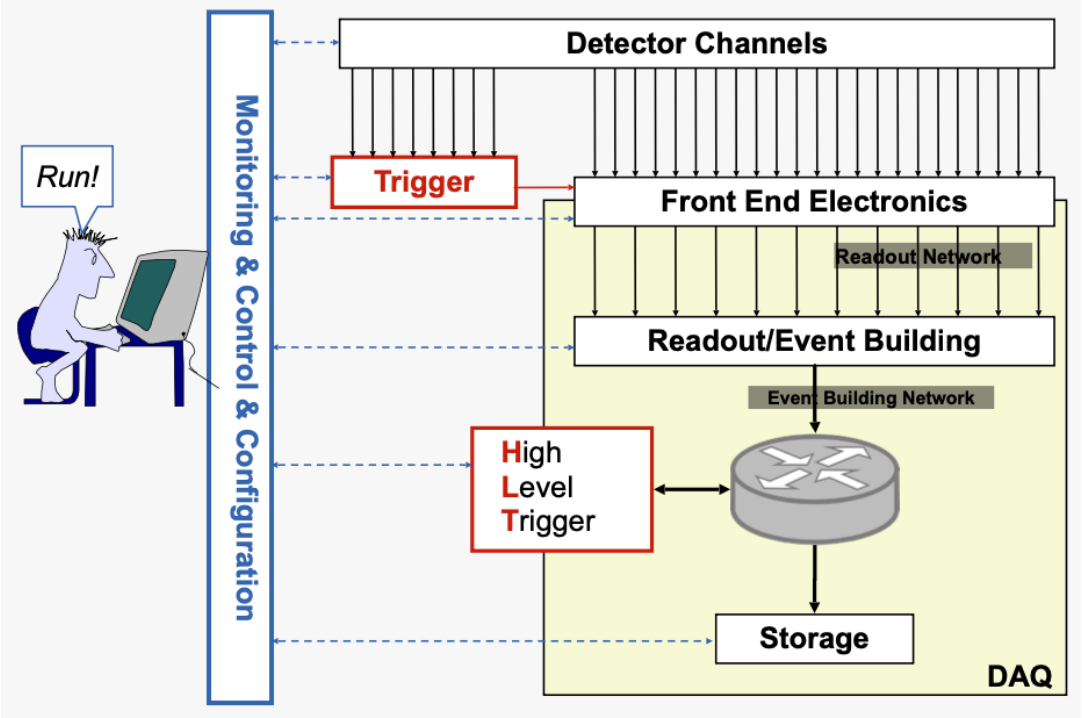
With experts from Qualcomm, Fermilab, and Google aiding in its development

MLCommons, the open engineering consortium behind the MLPerf benchmark test, has launched a new measurement suite aimed at 'tiny' devices like smartwatches and voice assistants.

MLPerf Tiny Inference is designed to compare performance of embedded devices and models with a footprint of 100kB or less, by measuring

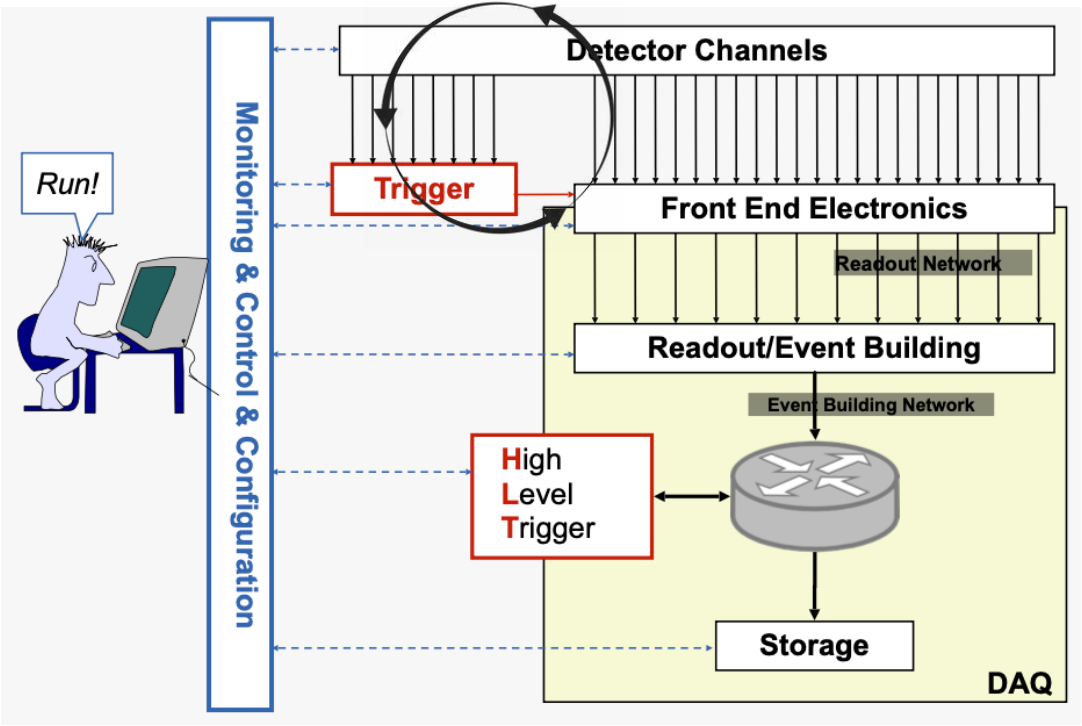


Fast and slow control



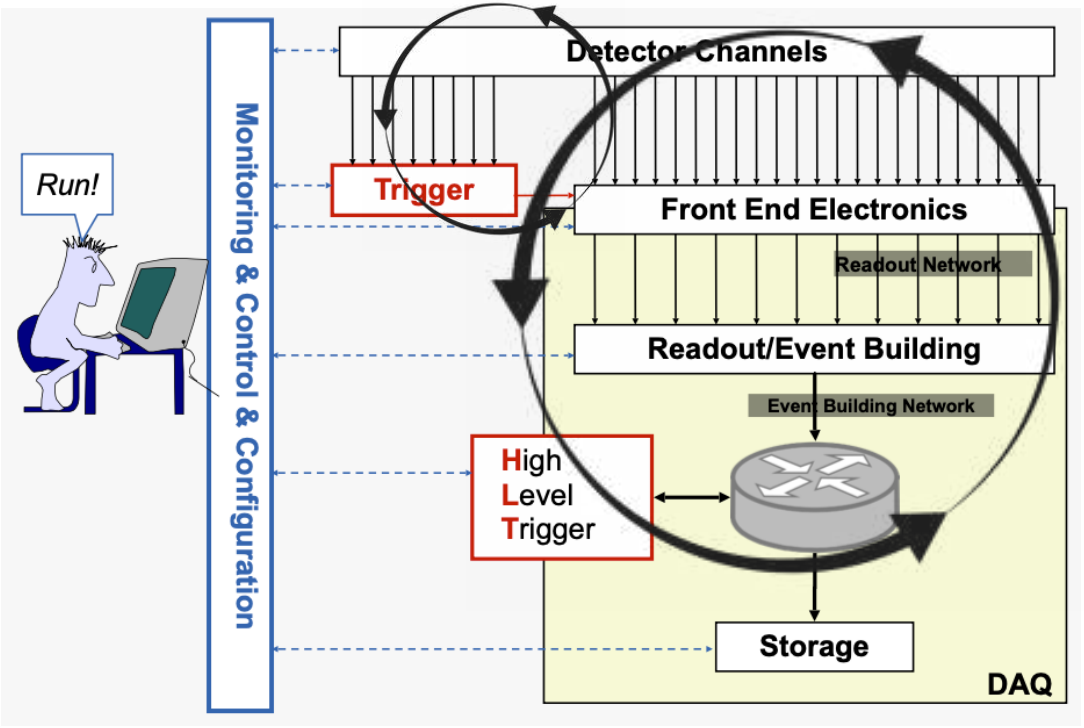
Fermilab

Fast and slow control



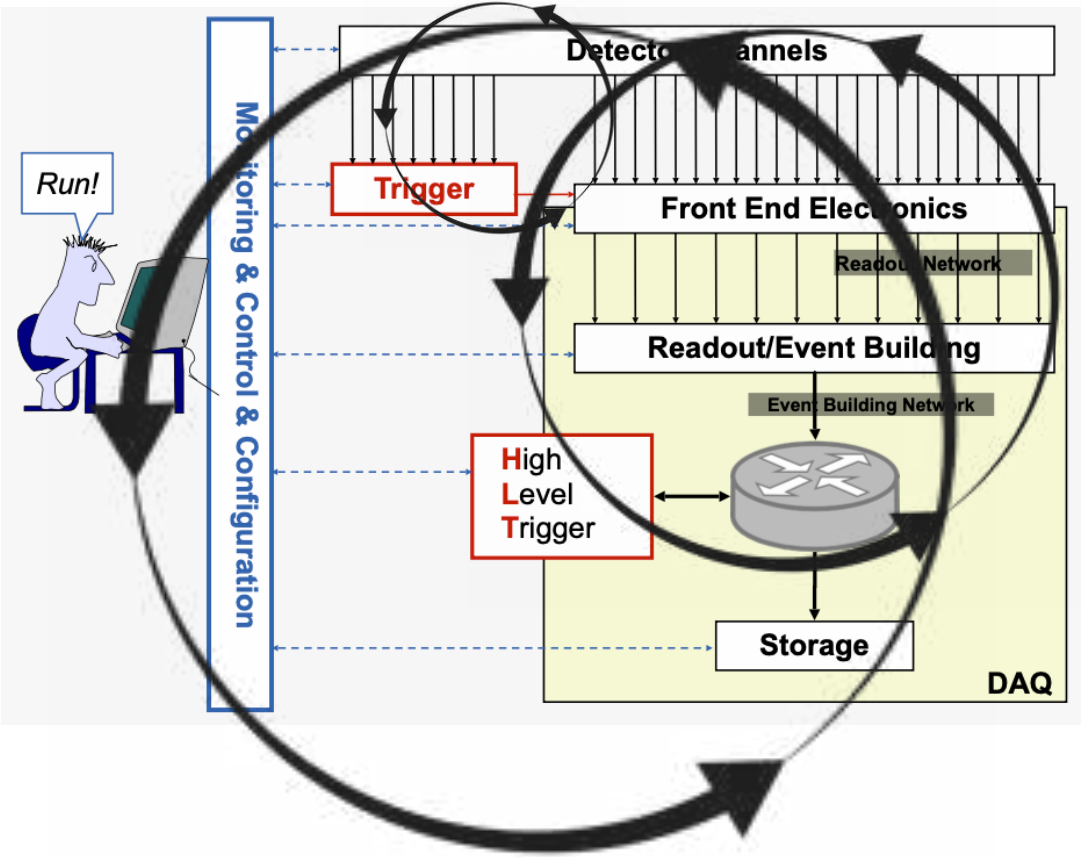
Fermilab

Fast and slow control



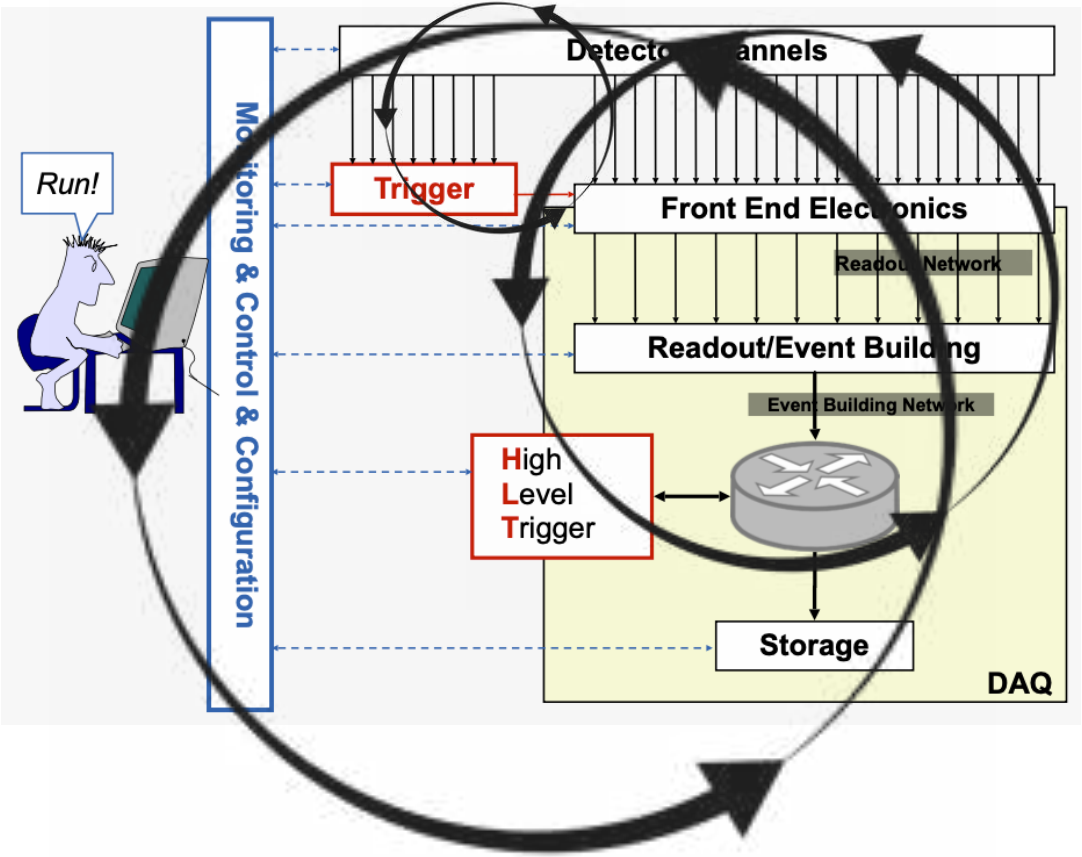
Fermilab

Fast and slow control



Fermilab

Fast and slow control



- Fast control
  - Immediate response to dynamics of the experiment and data readout
  - Event timing, triggering, etc.
- Slow control
  - Detector stability over minutes, days, weeks, months,...
  - Monitoring and controlling operational parameters: electronics gains, pedestals, calibrations, etc.

ML - the basics

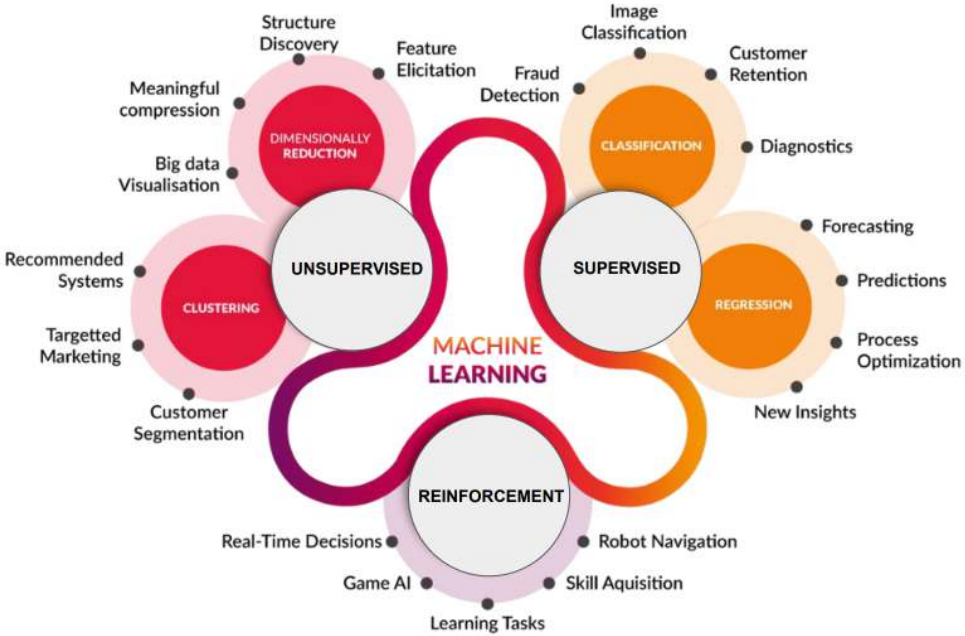
Why AI?

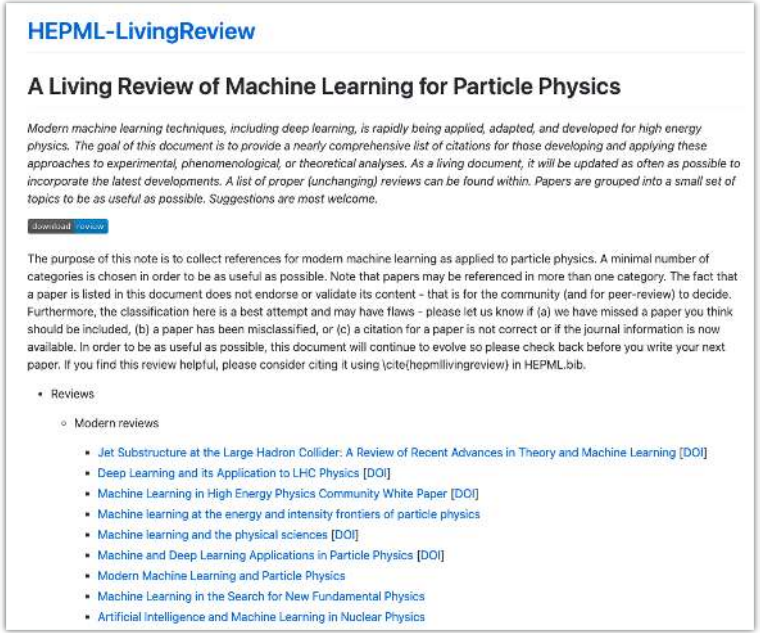
Universal function approximation - fit with customizable objective:

f(inputs; lots of parameters) = output

- Expressive: able to find patterns and correlations in high-dimensional data not explicitly accounted for
- Powerful: can unlock large gains in performance
- Adaptive, flexible, autonomous: able to adapt to new data, conditions automatically; handles all different types of data representations

All of AI in one slide





HEPML-LivingReview

A Living Review of Machine Learning for Particle Physics

Modern machine learning techniques, including deep learning, is rapidly being applied, adapted, and developed for high energy physics. The goal of this document is to provide a nearly comprehensive list of citations for those developing and applying these approaches to experimental, phenomenological, or theoretical analyses. As a living document, it will be updated as often as possible to incorporate the latest developments. A list of proper (unchanging) reviews can be found within. Papers are grouped into a small set of topics to be as useful as possible. Suggestions are most welcome.

Download Review

The purpose of this note is to collect references for modern machine learning as applied to particle physics. A minimal number of categories is chosen in order to be as useful as possible. Note that papers may be referenced in more than one category. The fact that a paper is listed in this document does not endorse or validate its content - that is for the community (and for peer-review) to decide. Furthermore, the classification here is a best attempt and may have flaws - please let us know if (a) we have missed a paper you think should be included, (b) a paper has been misclassified, or (c) a citation for a paper is not correct or if the journal information is now available. In order to be as useful as possible, this document will continue to evolve so please check back before you write your next paper. If you find this review helpful, please consider citing it using \cite{hepmllivingreview} in HEPML.bib.

- Reviews
  - Modern reviews
    - Jet Substructure at the Large Hadron Collider: A Review of Recent Advances in Theory and Machine Learning [DOI]
    - Deep Learning and its Application to LHC Physics [DOI]
    - Machine Learning in High Energy Physics Community White Paper [DOI]
    - Machine learning at the energy and intensity frontiers of particle physics
    - Machine learning and the physical sciences [DOI]
    - Machine and Deep Learning Applications in Particle Physics [DOI]
    - Modern Machine Learning and Particle Physics
    - Machine Learning in the Search for New Fundamental Physics
    - Artificial Intelligence and Machine Learning in Nuclear Physics

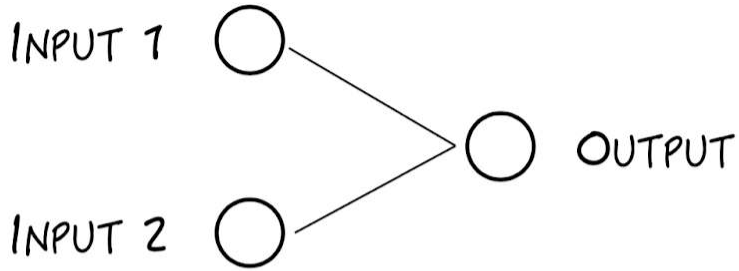
https://iml-wg.github.io/HEPML-LivingReview

Fermilab

Basic elements of machine learning

- Learning mathematical models from data that:
  - characterize the patterns, regularities, and relationships amongst variables in the system
- Three key components:
  - Model: chosen mathematical model
    - Depends on the task, data modality
  - Learning: estimate statistical model from data
  - Prediction and Inference: using statistical model to make predictions on new data points and infer properties of system(s)

Machine learning computation

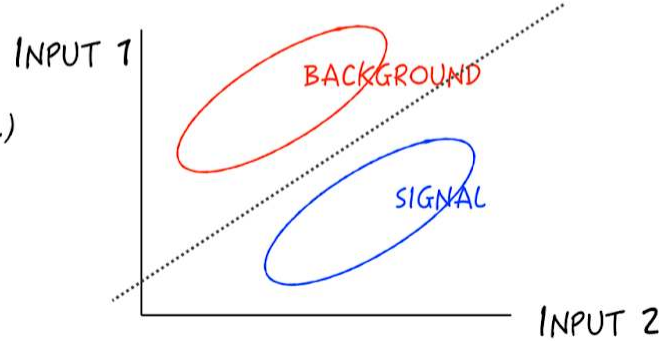


$$\vec{O}_j = \vec{I}_i \times \overleftrightarrow{W}_{ij} + \vec{b}_j$$

Simple 2 input example

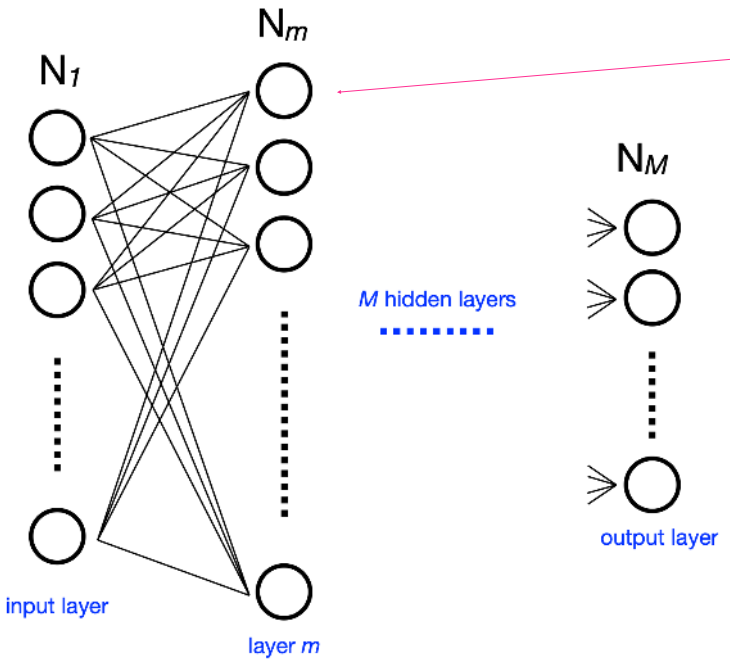
(Fisher linear discriminant, linear support vector machine,…)

$$O_1 = I_1 \times W_{11} + I_2 \times W_{21} + b_1$$



Fermilab

Machine learning computation



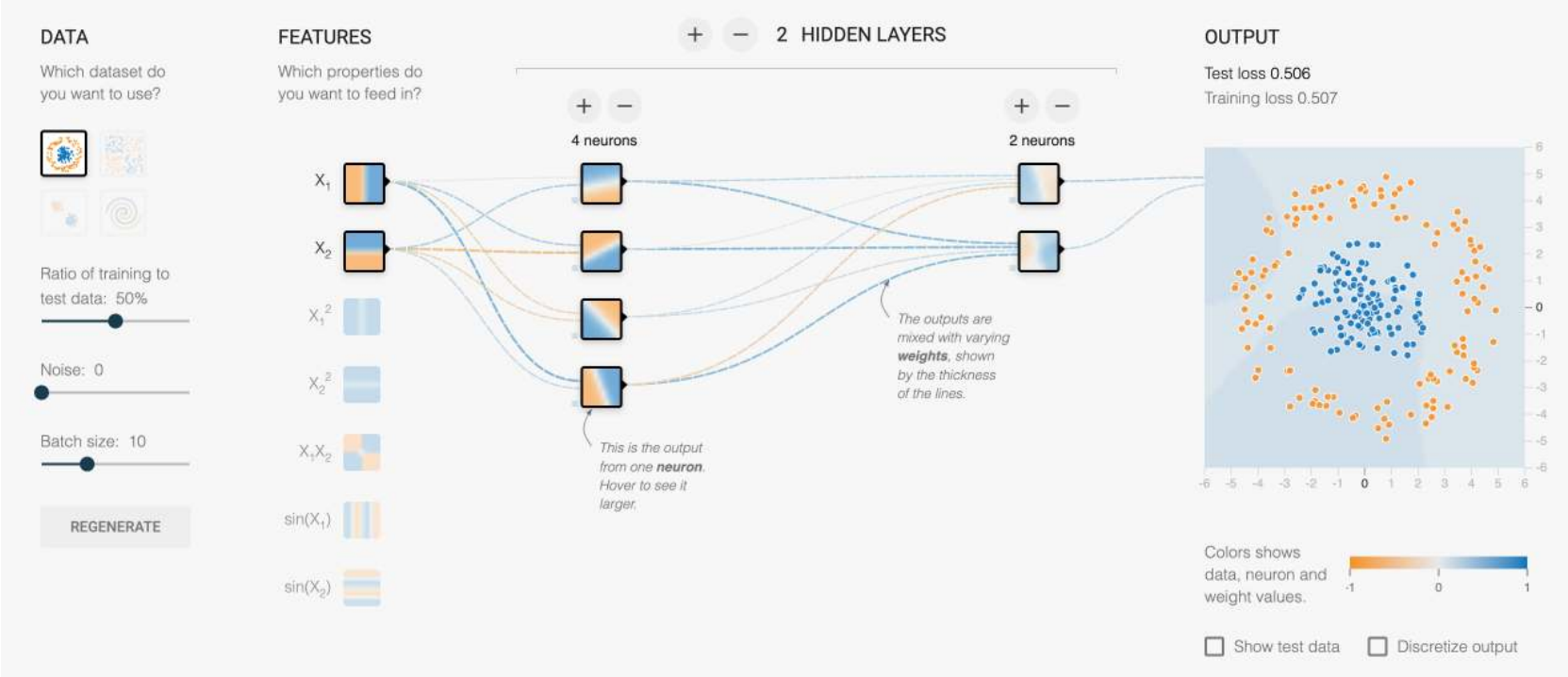
$$\boldsymbol{x}_m = g_m\left(\boldsymbol{W}_{m,m-1}\boldsymbol{x}_{m-1} + \boldsymbol{b}_m\right)$$

Matrix-vector multiply

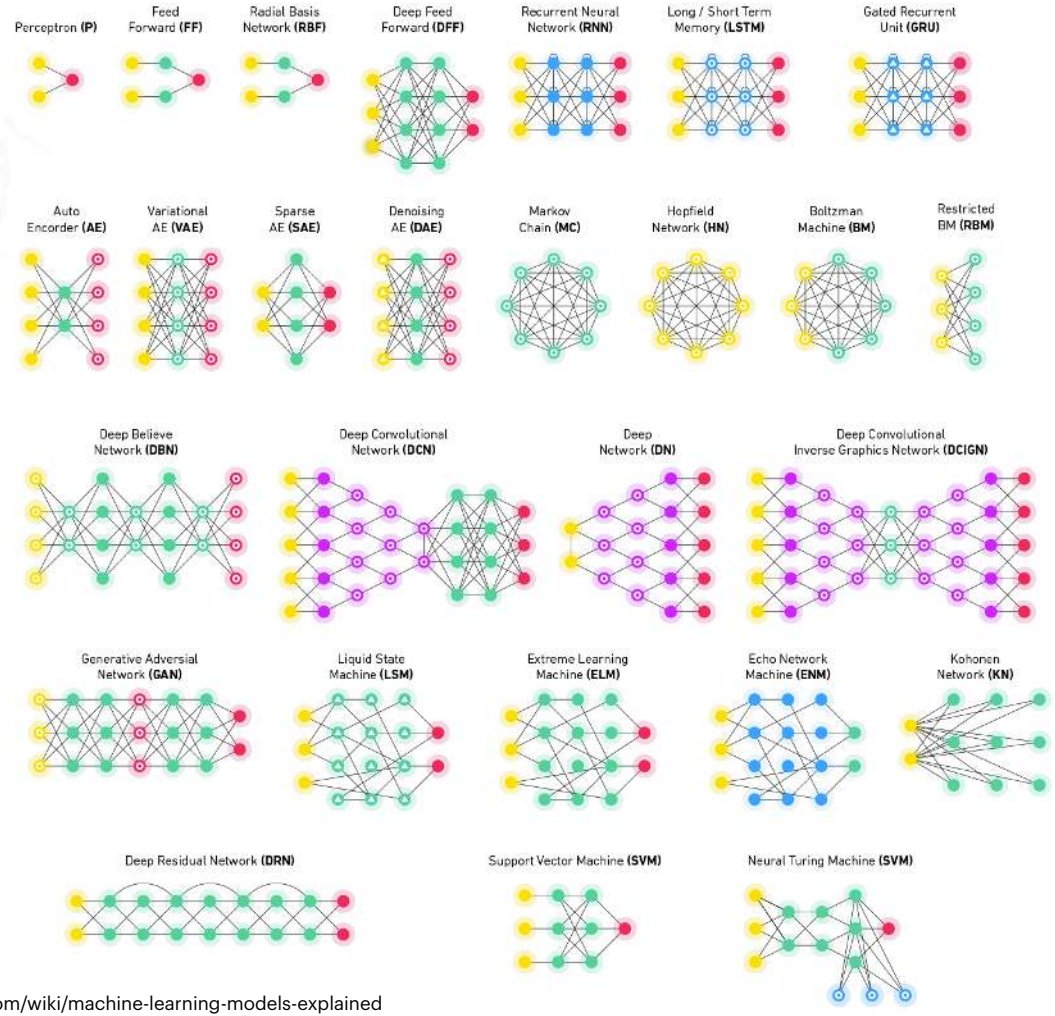
Non-linear activation function

Some intuition

https://playground.tensorflow.org/

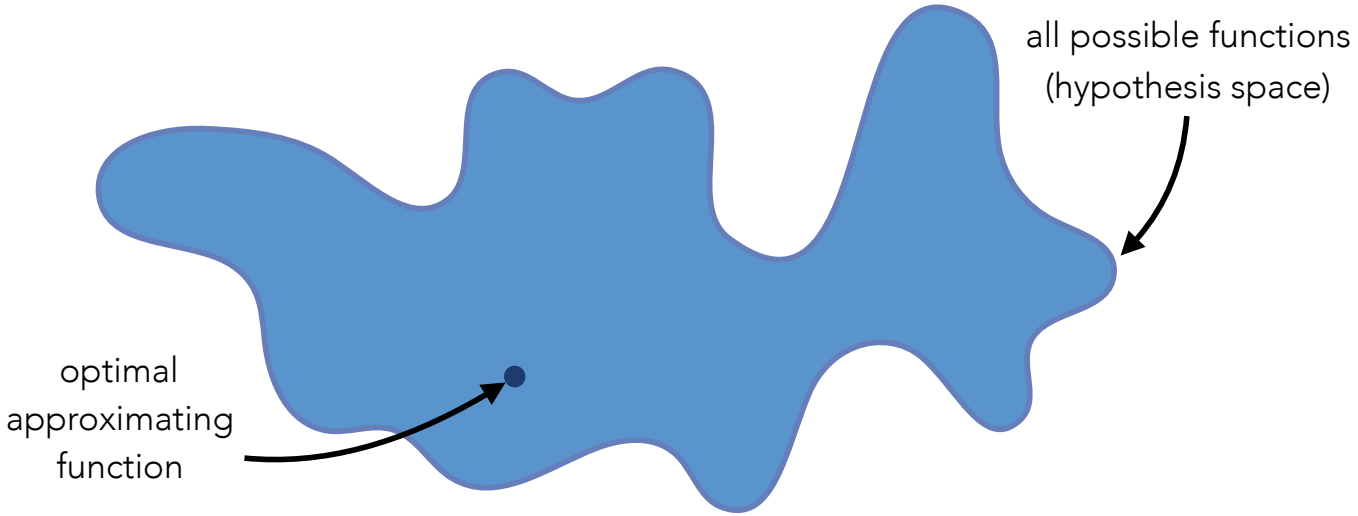


Fermilab



 https://machine-learning.paperspace.com/wiki/machine-learning-models-explained

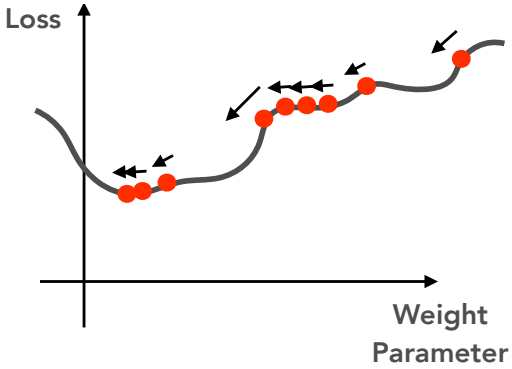
neural networks are universal function approximators,
but we still must find an optimal approximating function



we do so by <u>adjusting the weights</u>

Learning = optimization

learning as *optimization*



to learn the weights, we need the **derivative** of the loss w.r.t. the weight

i.e. "*how should the weight be updated to decrease the loss?*"

$$w' = w - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

with multiple weights, we need the **gradient** of the loss w.r.t. the weights

$$\mathbf{w}' = \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}$$

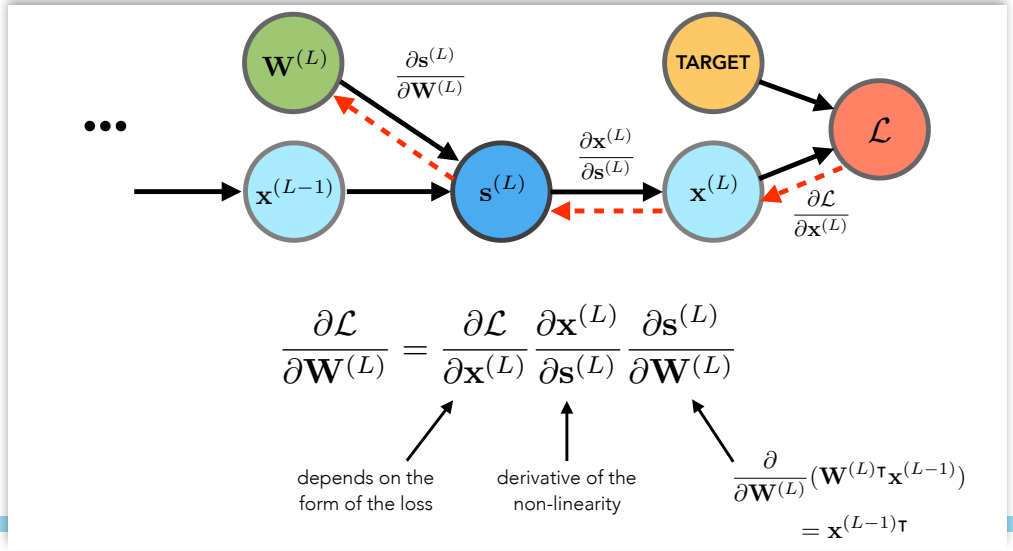
Backpropagation

a neural network defines a function of composed operations

$$f_L(\mathbf{w}_L, f_{L-1}(\mathbf{w}_{L-1}, \dots f_1(\mathbf{w}_1, \mathbf{x}) \dots))$$

and the loss $\mathcal{L}$ is a function of the network output

→ use chain rule to calculate gradients



Fermilab

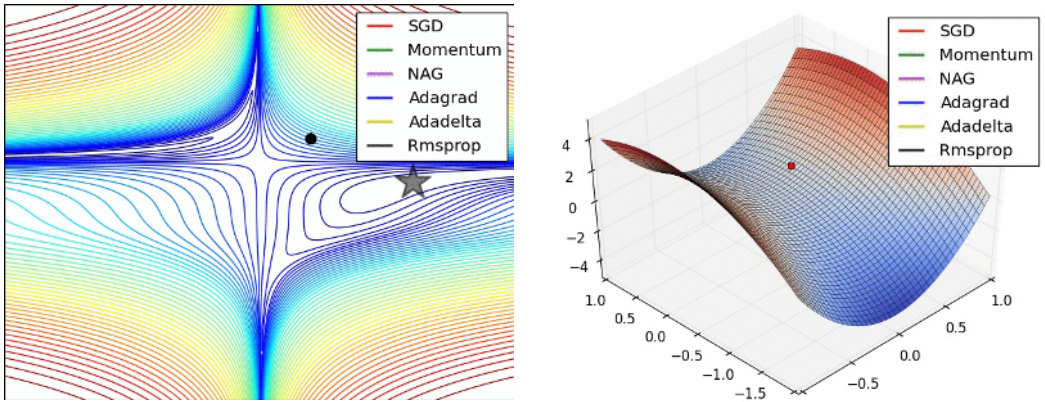
Stochastic gradient descent

See animated gifs: http://ruder.io/optimizing-gradient-descent/

stochastic gradient descent (SGD): $w = w - \alpha \tilde{\nabla}_w \mathcal{L}$

use *stochastic gradient* estimate to *descend* the surface of the loss function

recent variants use additional terms to maintain "memory" of previous gradient information and scale gradients per parameter



local minima and saddle points are largely not an issue

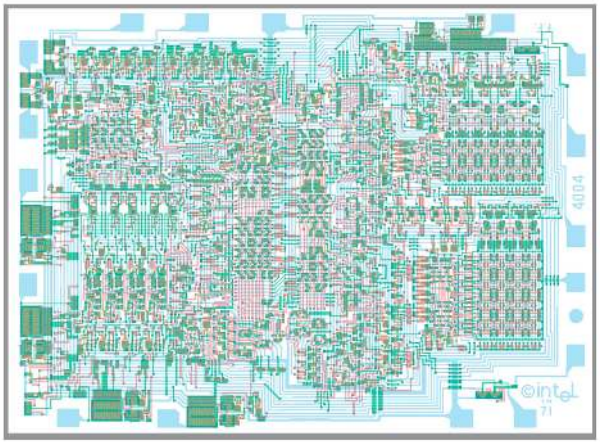
in many dimensions, can move in exponentially more directions

Compute technology & platforms

Basics of computing

Beomki Yeo
https://indico.cern.ch/event/1405035/contributions/5937283/

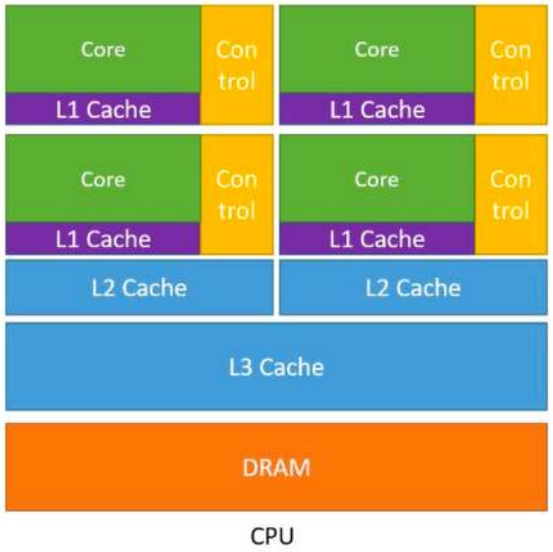
- Microprocessor: A single Integrated Circuit which can do data processing and logic control
- Integrated Circuit: A chunk of transistors
- Transistor: A minimal building block of electronics



Intel 4004 chipset design (2300 transistors)

CPU

- CPU (Central Processing Unit): Made of Cores, Caches and Control Units
  - Core: Algorithm Logical Units (ALUs) and registers
    - ALU: performs mathematical operations
    - Register: small storage which stores data being processed
  - Cache: On-chip memory
  - Control Unit: Distribute operations to other units



Fermilab

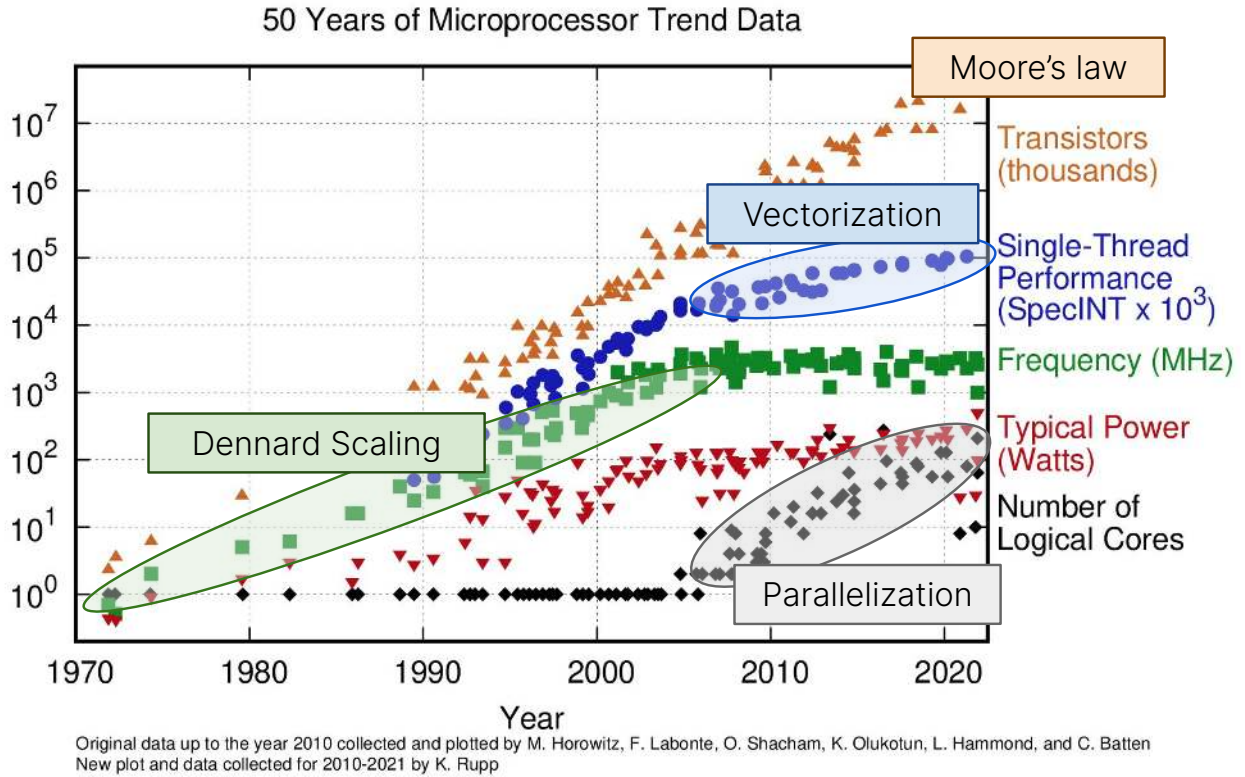
Moore's law, Dennard Scaling, Pollack's Rule

- Moore's Law: observation that the number of transistors in processors doubles every two years
- Dennard Scaling
  - Free scaling of the frequency (f) for the same power consumption (P)
  - $P=\alpha CV^2f$
  - Capacitance (C) and operating voltage (V) are linearly reduced with the size of transistor
- Pollack's rule
  - Observation of Performance ~ $\sqrt{N}$ (N = the number of transistors)
  - Moore's law allows more number of transistors (N) on the same chip size

Moore's law, Dennard Scaling, Pollack's Rule

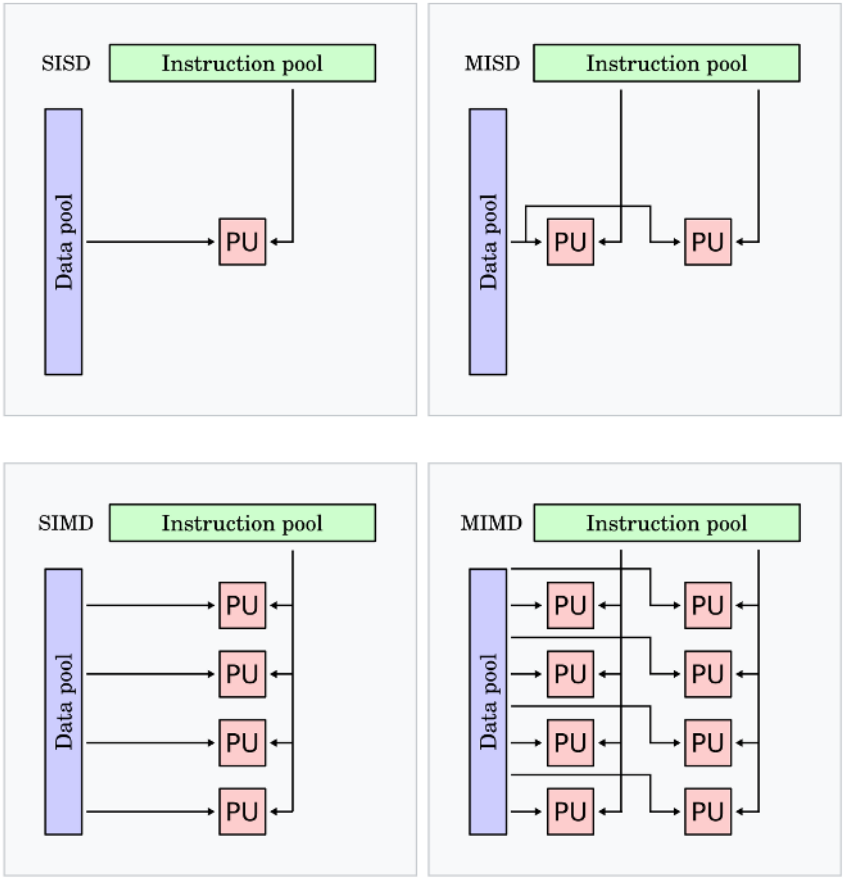
Below the transistor size of 65 nm (since yr. 2005), the current leakage ($I_{leakage}$) is not negligible anymore

$P=\alpha CV^2f + VI_{leakage}$



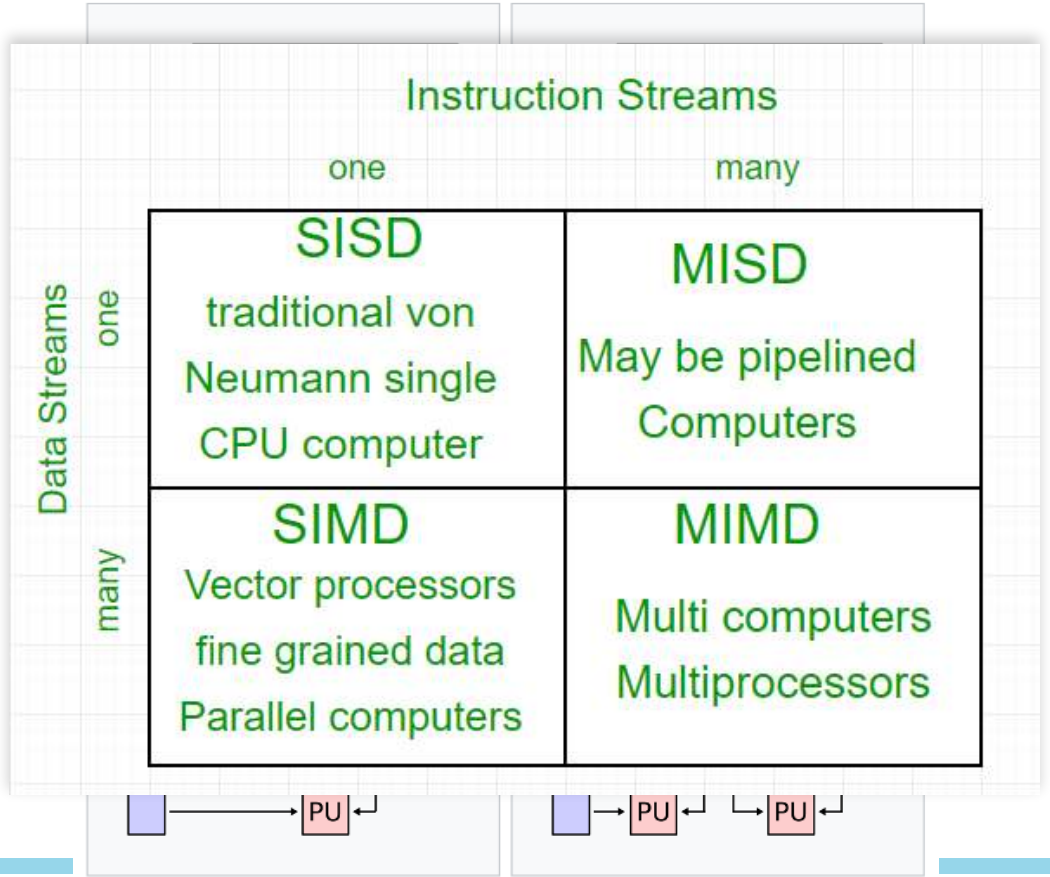
Fermilab

Flynn's Taxonomy



Fermilab

Flynn's Taxonomy

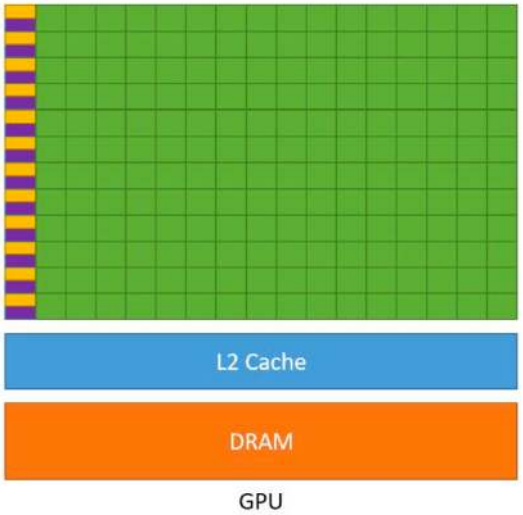


Fermilab

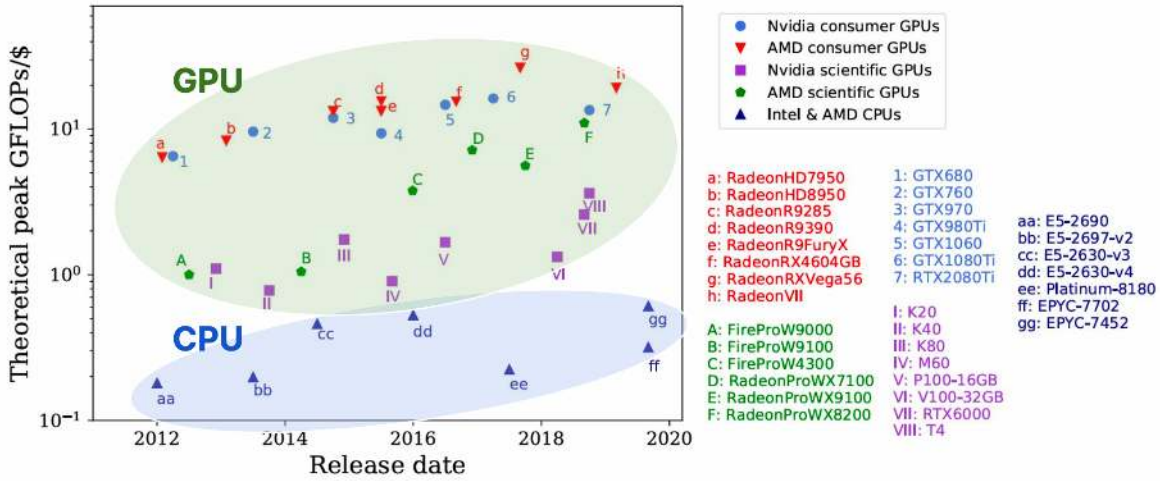
https://arxiv.org/pdf/2003.11491

GPUs

- Graphical processing unit
- Many number of cores (~1000)
  - Much simpler than CPU
  - Small caches



- Originally intended for graphics on PC screen
- Major vendors: Nvidia, AMD, Intel



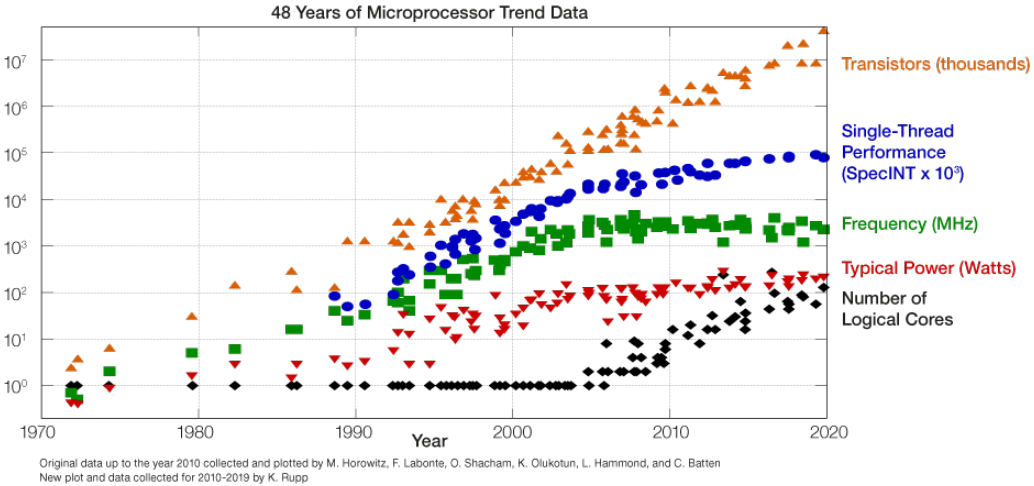
Fermilab

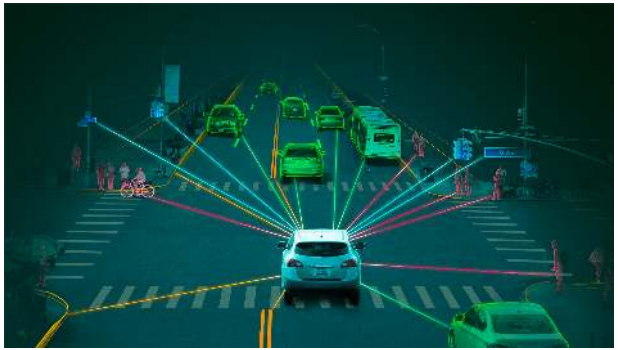
Rise of ML

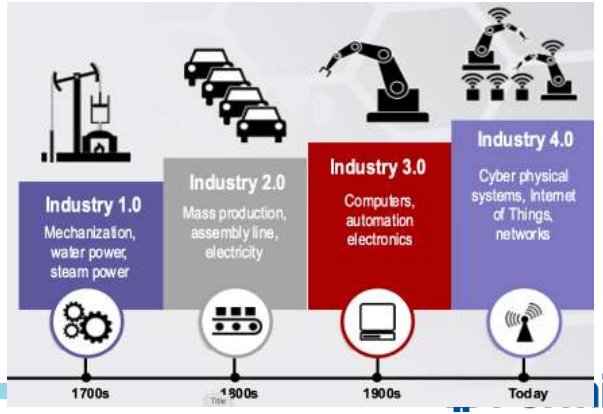
- Necessity/Data
- Hardware
- ML Research
- Tools

Rise of ML

- Necessity/Data
- Hardware
- ML Research
- Tools

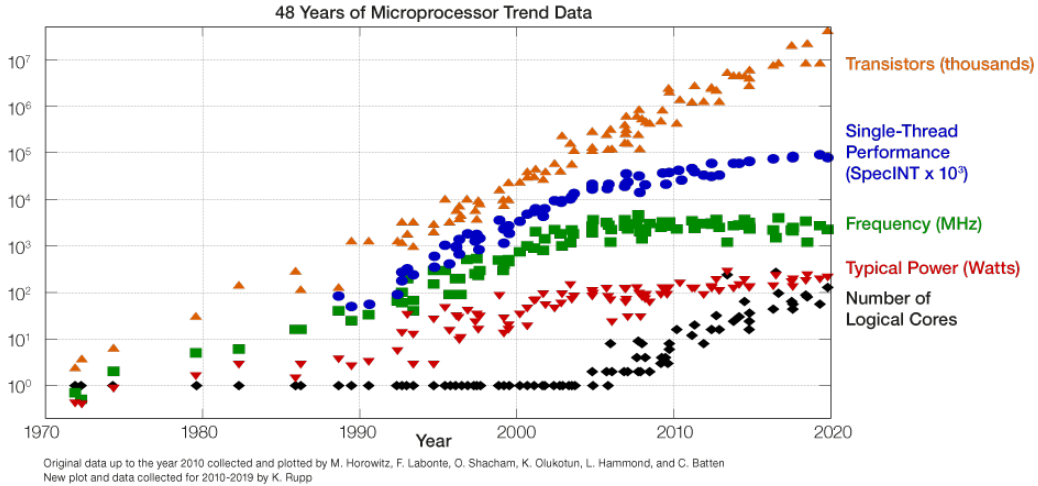


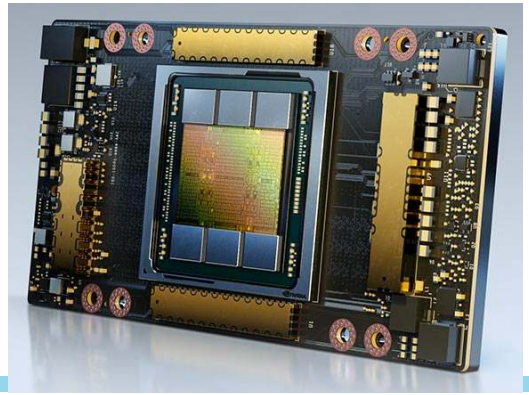




Rise of ML

- Necessity/Data
- Hardware
- ML Research
- Tools





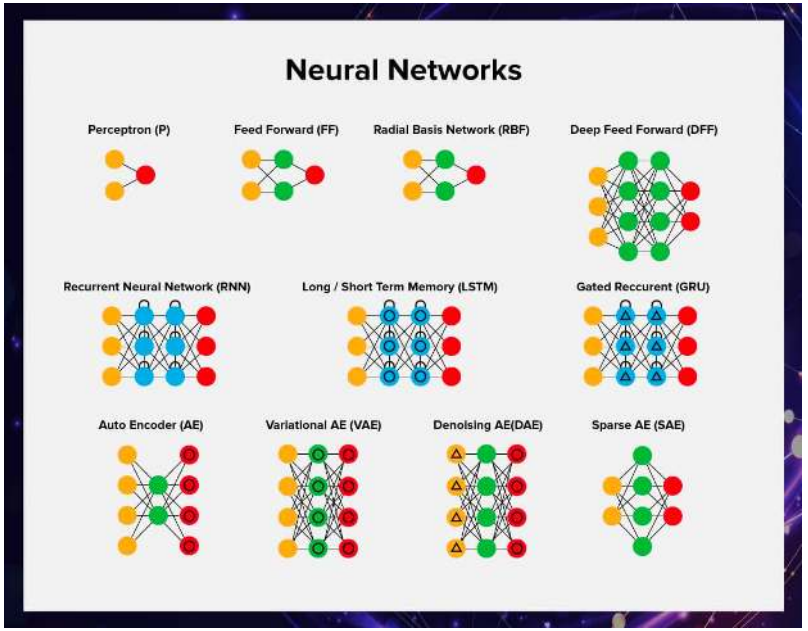
Technology size	Year	Technology size	Year
10 um	1971	130 nm	2001
6 um	1974	90 nm	2004
3 um	1977	65 nm	2006
1.5 um	1982	45 nm	2008
1 um	1985	32 nm	2010
800 nm	1989	22 nm	2012
600 nm	1994	14 nm	2014
350 nm	1995	10 nm	2017
250 nm	1997	7 nm	2018
180 nm	1999	5 nm	2020

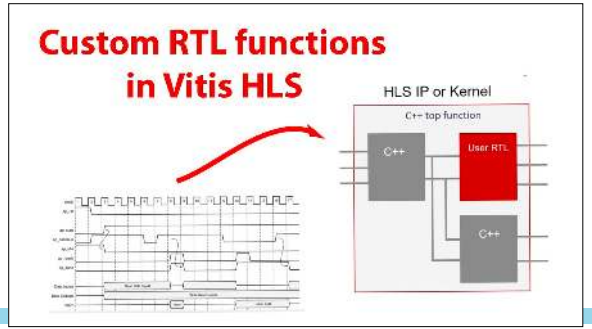
Rise of ML

- Necessity/Data
- Hardware
- ML Research
- Tools

TensorFlow

PyTorch





NVIDIA CUDA

Fermilab

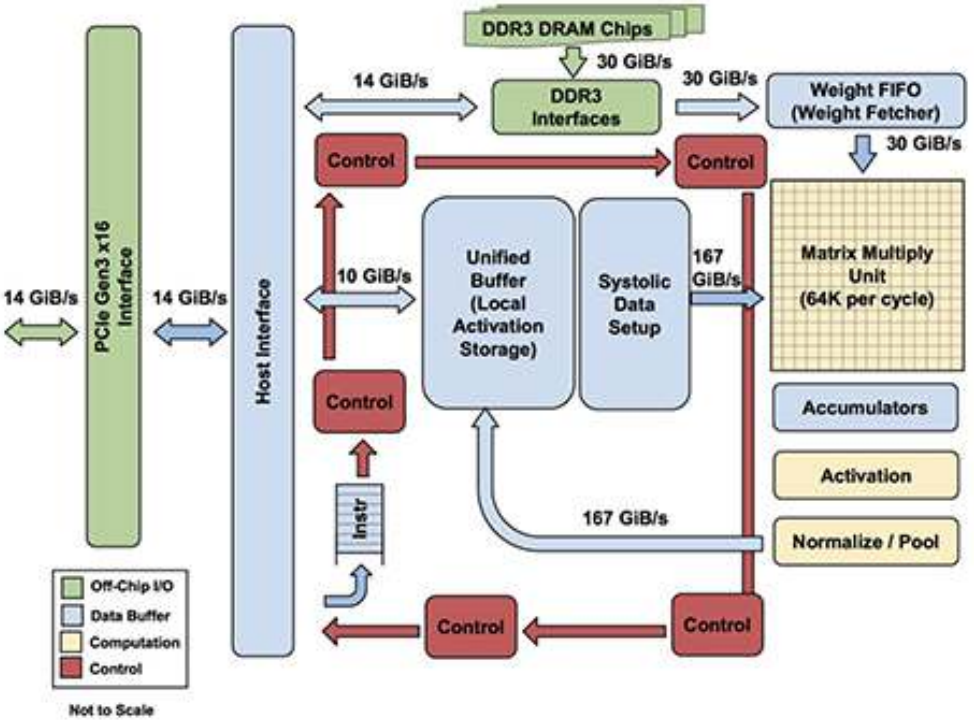

COMMUNICATIONS
CACM.ACM.ORG
OF THE ACM
02/2019 VOL.62 NO.02
A New Golden Age for Computer Architecture
Agriculture Technology
Monitoring Noise Pollution
The Computational Sprinting Game
Blockchain from a Distributed Computing Perspective

Fermilab

Types of compute

Types of compute

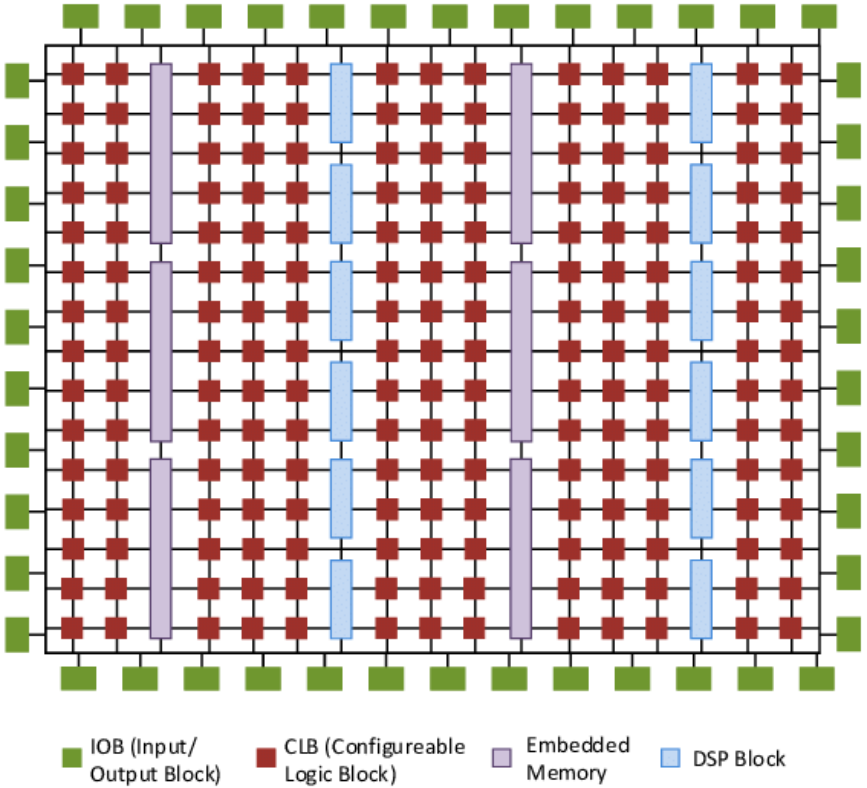
- ASIC
  - Google TPU block diagram
  - Very efficient compute but long development times and challenge to make general purpose



Fermilab

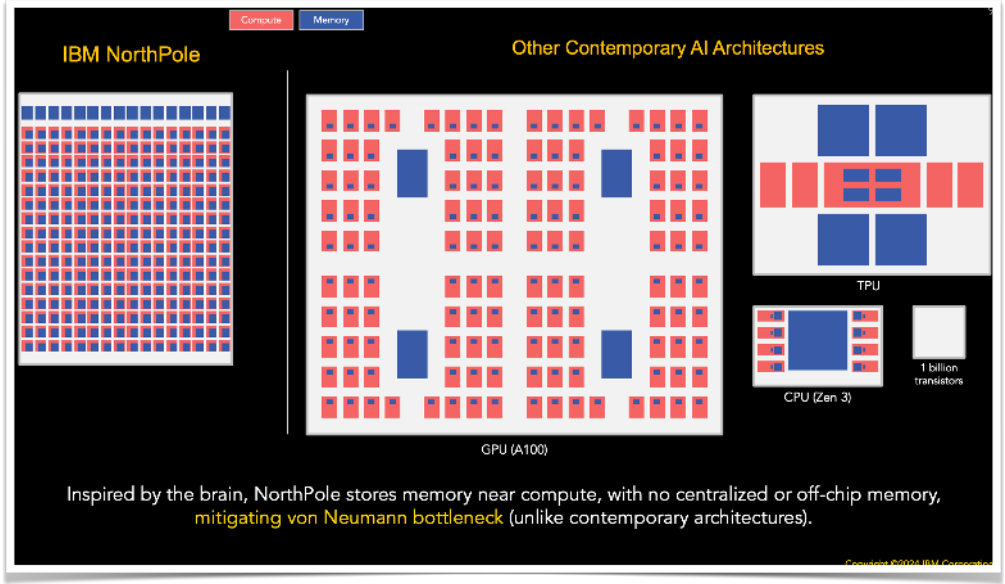
Types of compute

- FPGA
  - More flexible to changing workloads
  - Still not that easy to program



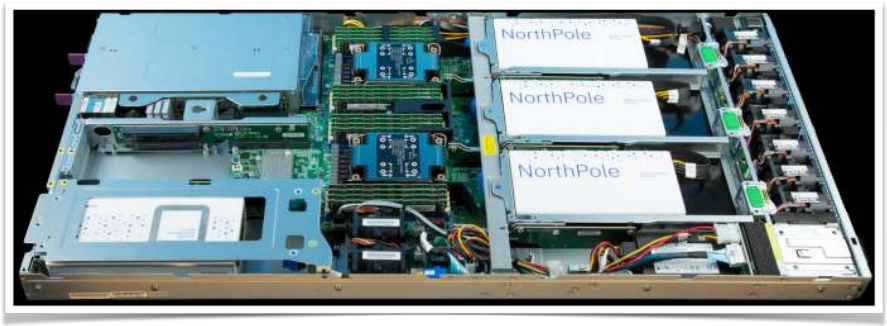
Types of compute

- NPUs
  - Fast moving space
  - Immature software ecosystem
  - Interoperability a challenge

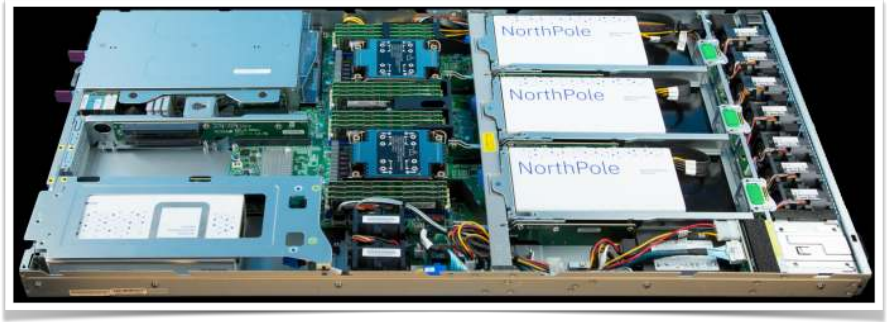


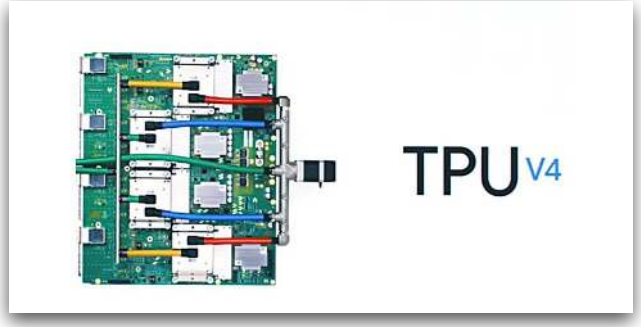
Coprocessors

Coprocessors

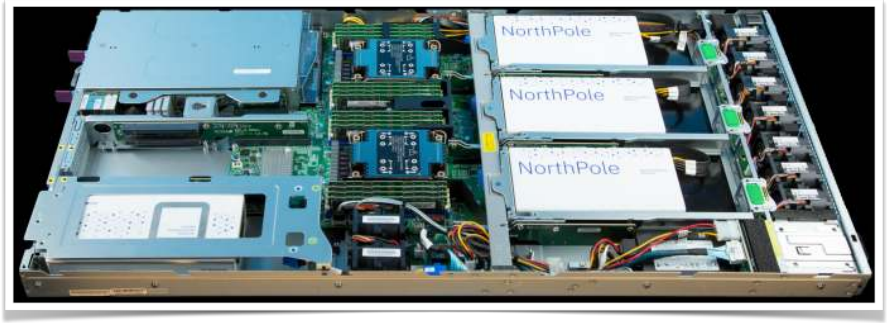


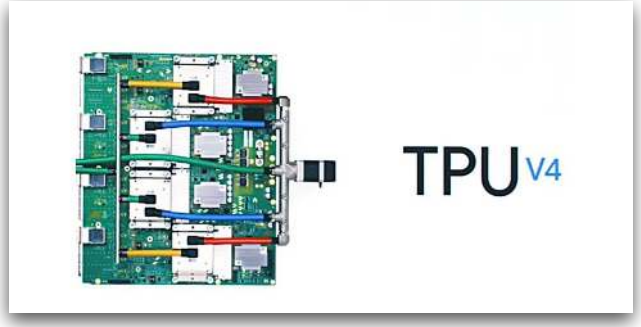
Coprocessors

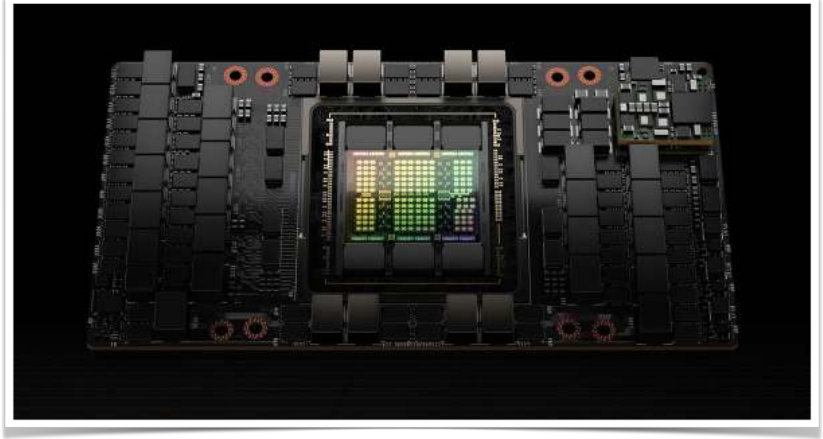




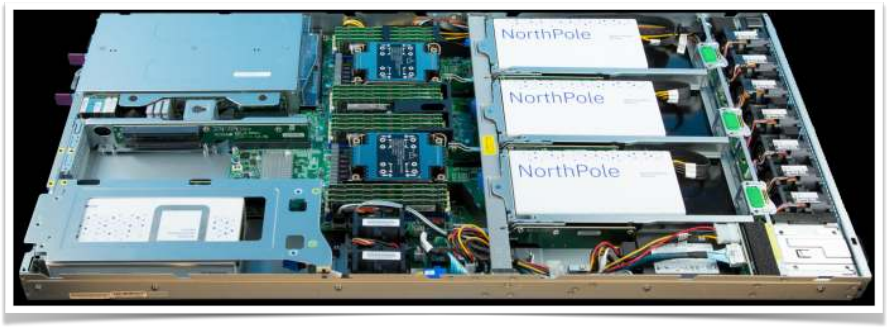
Coprocessors

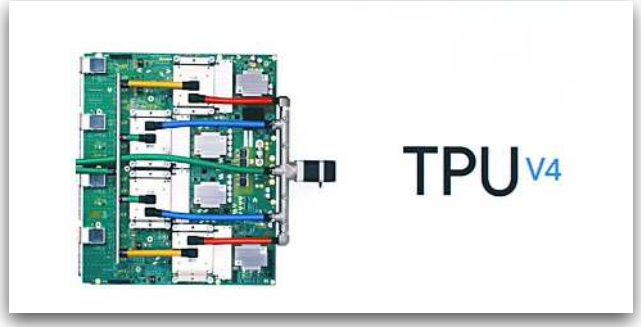


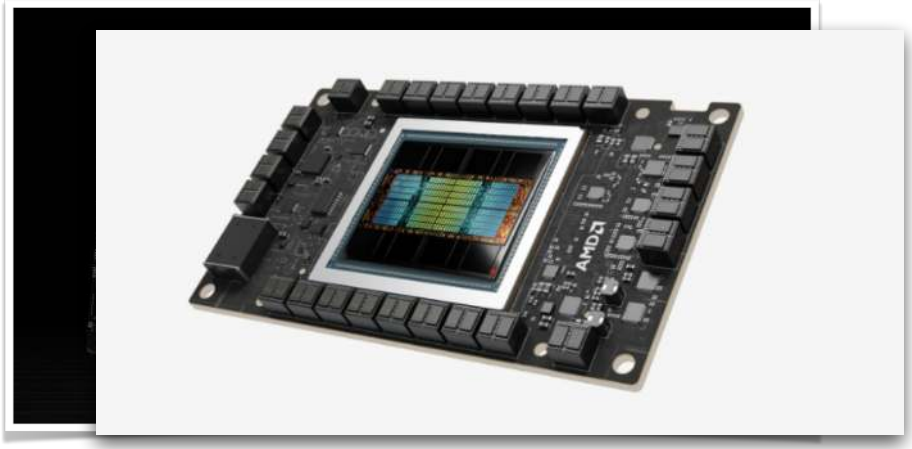




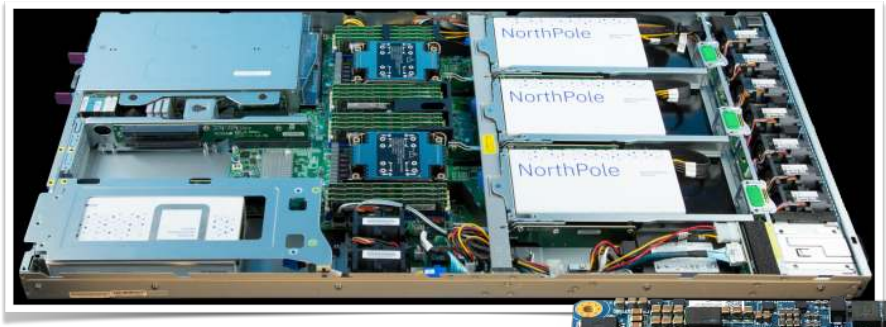
Coprocessors

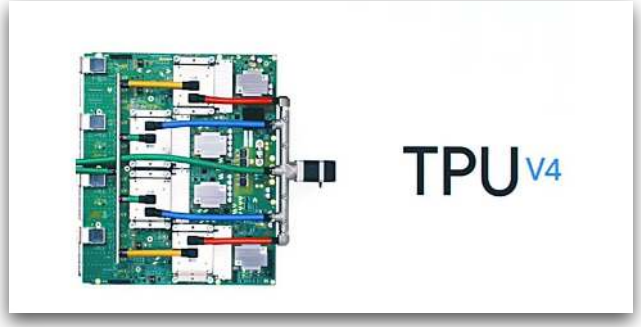


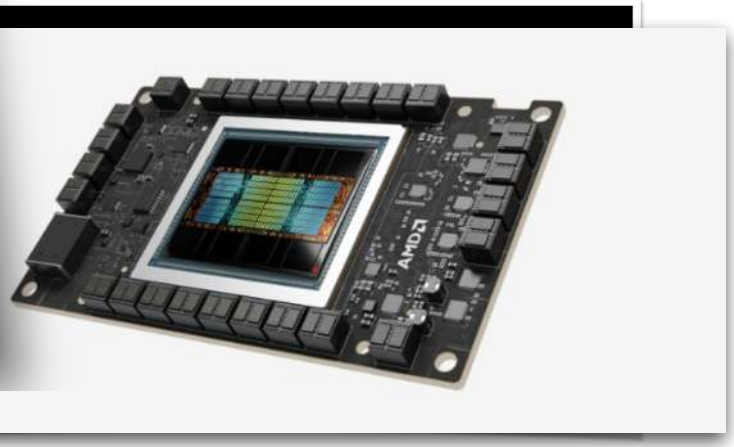




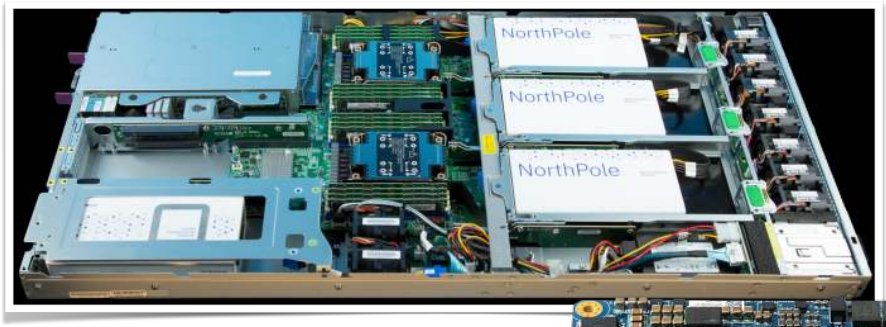
Coprocessors

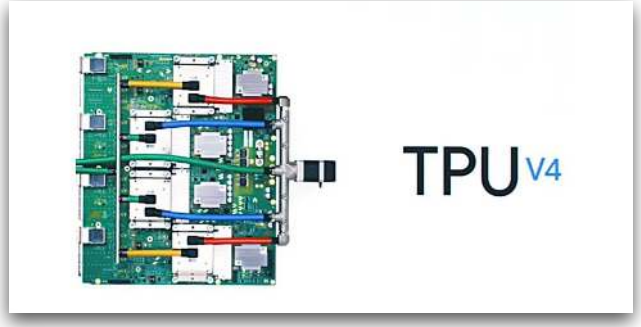


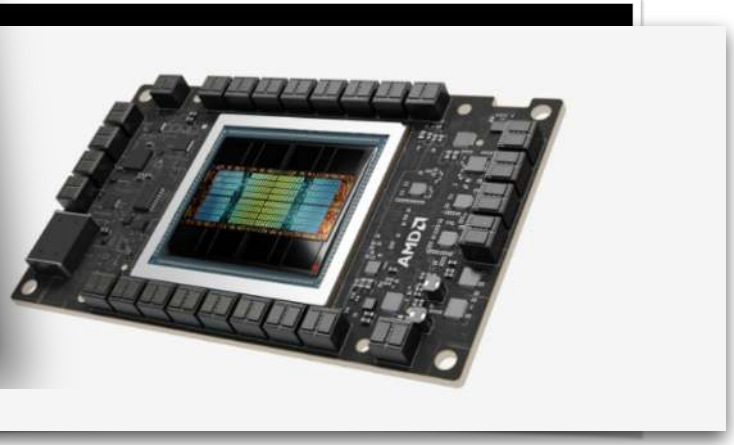




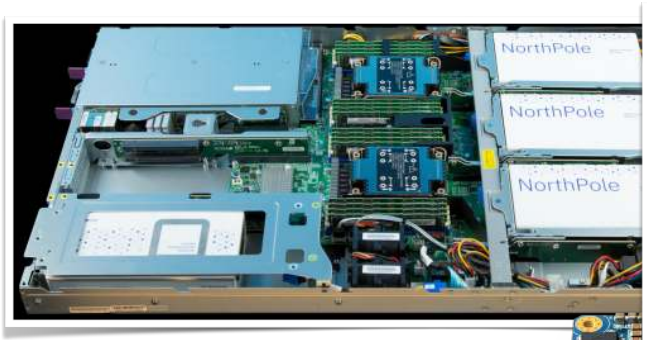
Coprocessors

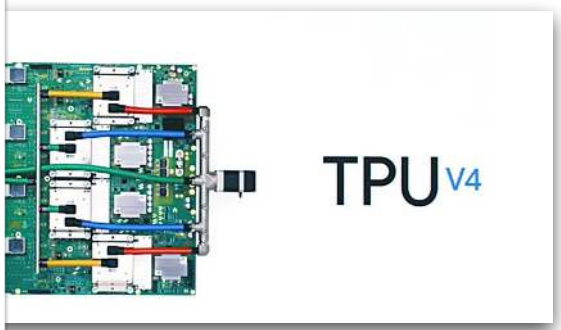


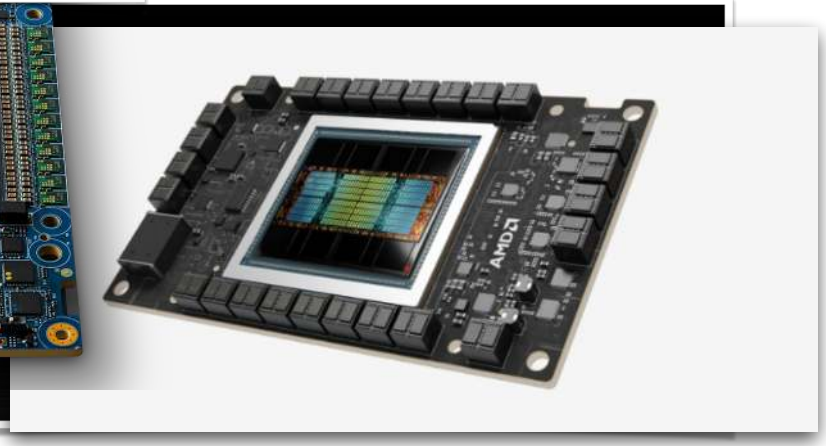




Coprocessors

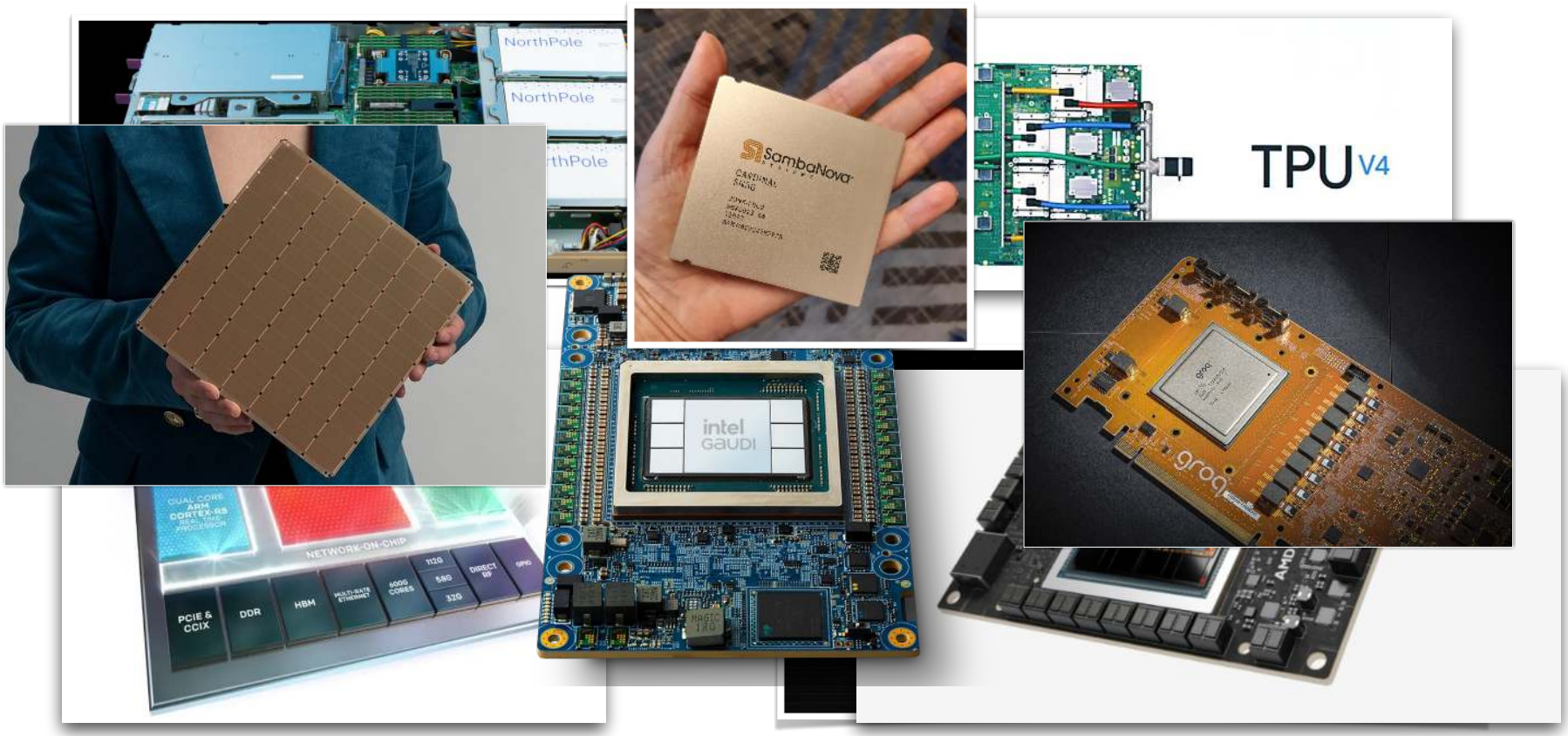




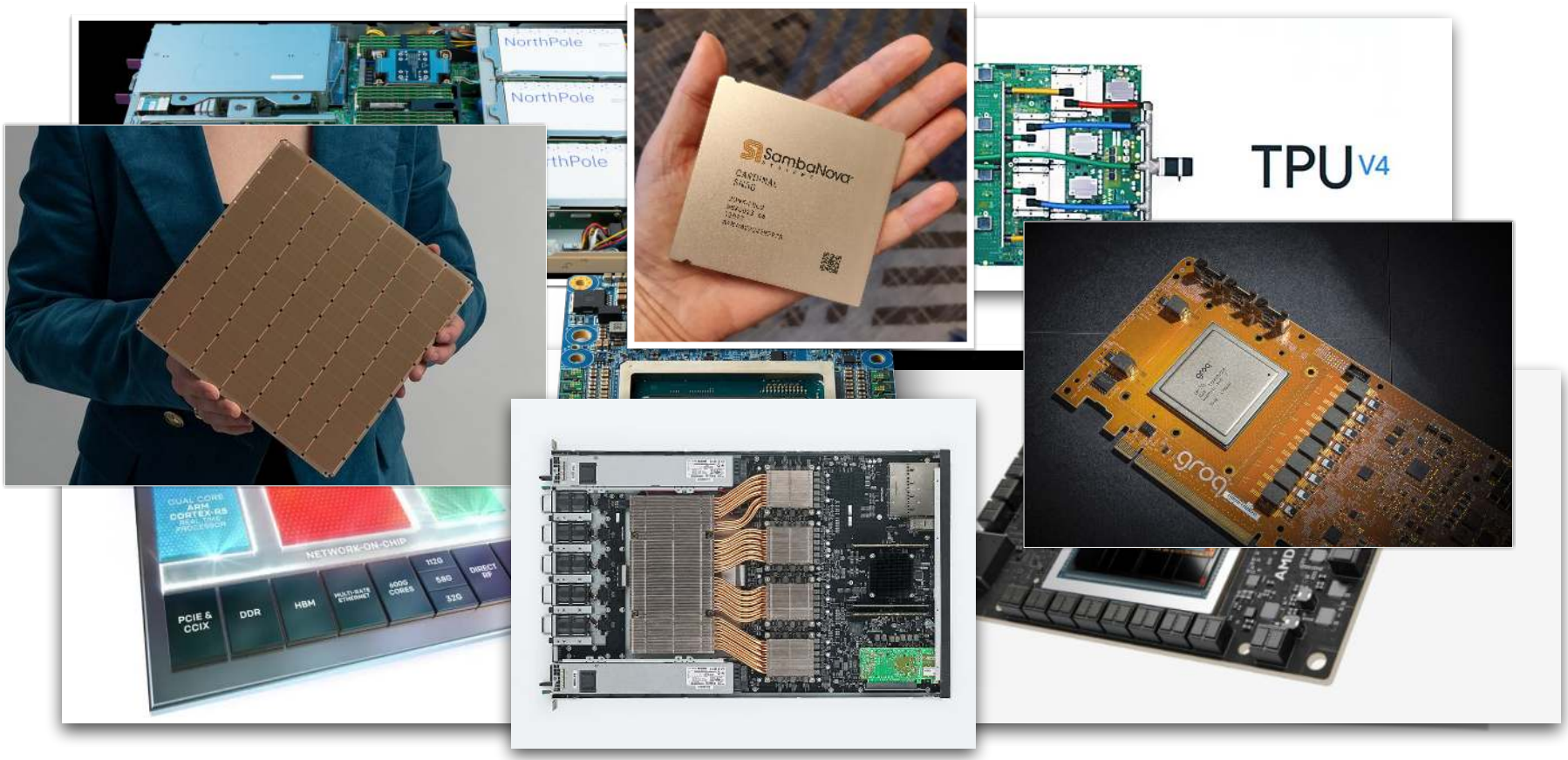


Coprocessors

Coprocessors



Coprocessors



Modalities of processing

	CPU	GPU	FPGA
Latency	O (10) µs	O (100) µs	Deterministic, O (100) ns
I/O with processor	Ethernet, USB, PCIe	PCIe, Nvlink	Connectivity to any data source via printed circuit board (PCB)
Engineering cost	Low entry level (programmable with c++, python, etc.)	Low entry level (programmable with CUDA, OpenCL, etc.)	Some high-level syntax available, traditionally VHDL, Verilog (specialized engineer)
Single precision floating point performance	O (10) TFLOPs	O (10) TFLOPs	Optimized for fixed point performance
Serial / parallel	Optimized for serial performance, increasingly using vector processing	Optimized for parallel performance	Optimized for parallel performance
Memory	O (100) GB RAM	O (10) GB	O (10) MB (on the FPGA itself, not the PCB)
Backward compatibility	Compatible, except for vector instruction sets	Compatible, except for specific features only available on modern GPUs	Not easily backward compatible

Accelerated compute

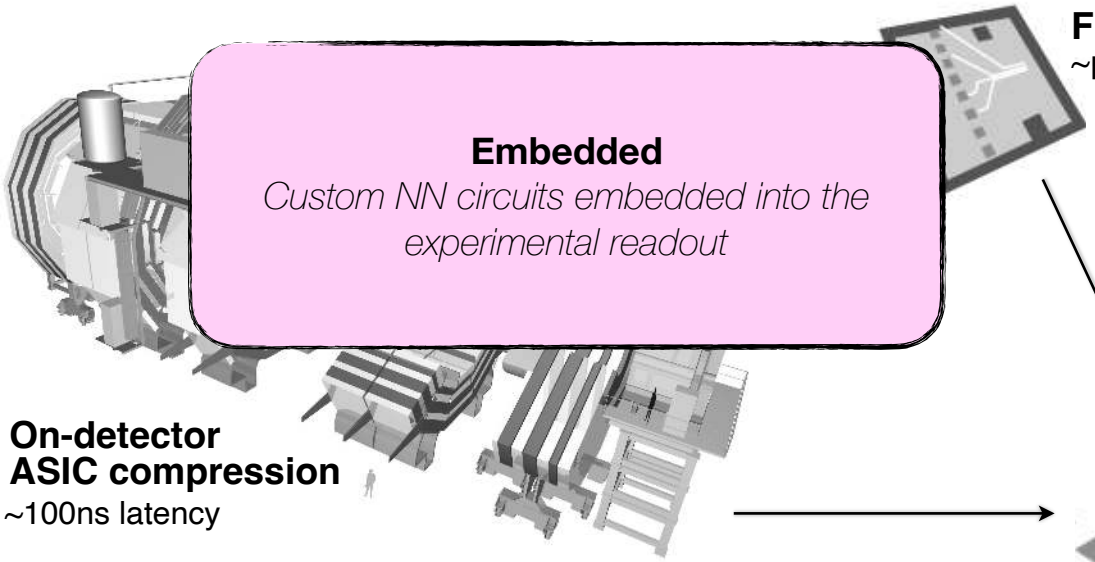
Embedded Systems

Embedded in our experiments; often (hard) real-time latency constraints, custom architectures

Coprocessors

Traditional datacenter-scale compute; throughput-driven; general purpose architectures

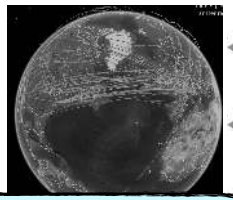
Fast ML regimes



Embedded
Custom NN circuits embedded into the experimental readout

FPGA filter stack
~μs latency

On-detector ASIC compression
~100ns latency



Coprocessors
Optimally leverage advances in industry for task-based hardware acceleration as-a-service

On-prem CPU/GPU filter farm
~100 ms latency

Efficient ML codesign
(For embedded systems)

Spatial programming and FPGAs

- Field Programmable Gate Arrays are reprogrammable integrated circuits
- Contain many different building blocks ('resources') which are connected together as you desire
- Originally popular for prototyping ASICs, but now also for high performance computing

Now Intel!

Now AMD!

Spatial programming and FPGAs

- Field Programmable Gate Arrays are reprogrammable integrated circuits
- **Logic cells / Look Up Tables** perform arbitrary functions on small bitwidth inputs (2-6)
  - These can be used for boolean operations, arithmetic, small memories
- **Flip-Flops** register data in time with the clock pulse
- **DSPs (Digital Signal Processor)** are specialized units for multiplication and arithmetic
  - Faster and more efficient than using LUTs for these types of operations
- **BRAMs** are small, fast memories - RAMs, ROMs, FIFOs (18Kb each in Xilinx)
  - Memories using BRAMs more efficient than using LUTs

.

Spatial programming and FPGAs

- Field Programmable Gate Arrays are reprogrammable integrated circuits

- **High speed transceivers** with Tb/s total bandwidth
  PCIe, (Multi) Gigabit Ethernet, Infiniband
- AND: Support **highly parallel** algorithm implementations
- **Low power per Op** (relative to CPU/GPU)

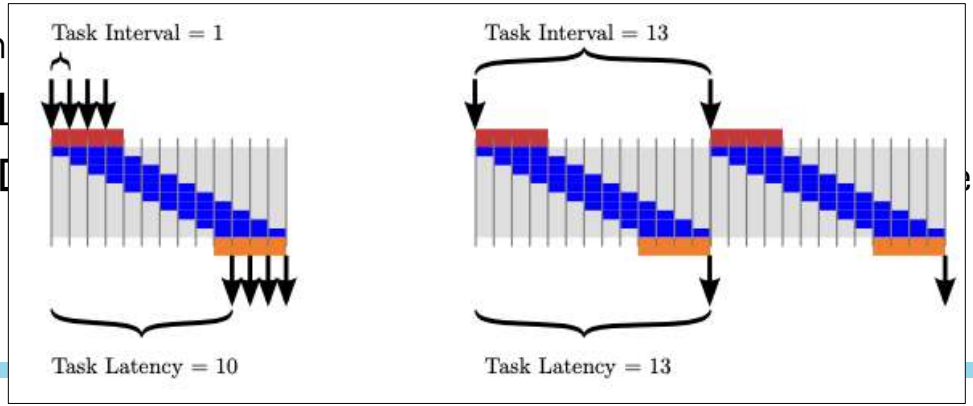
- Cons:
  - Limited resources on chip
  - Difficult to program - concurrency always challenging

Spatial programming and FPGAs

- Field Programmable Gate Arrays are reprogrammable integrated circuits

- **High speed transceivers** with Tb/s total bandwidth
  PCIe, (Multi) Gigabit Ethernet, Infiniband
- AND: Support **highly parallel** algorithm implementations
- **Low power per Op** (relative to CPU/GPU)

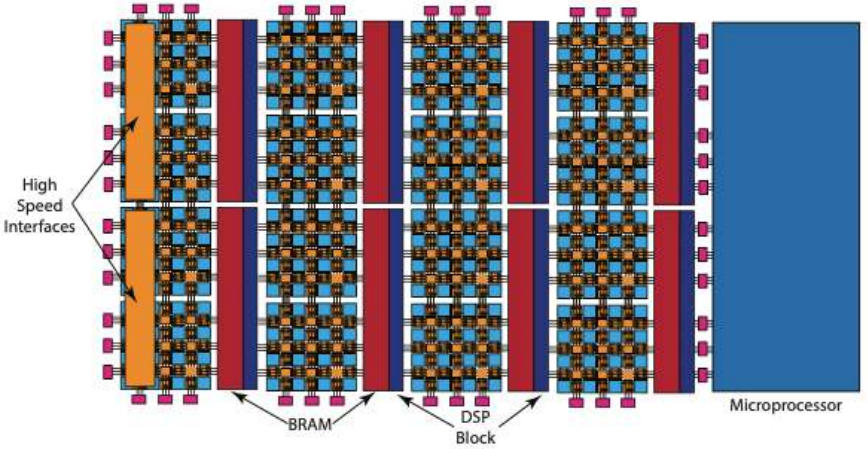
- Con
  - L
  - D

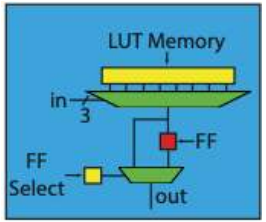


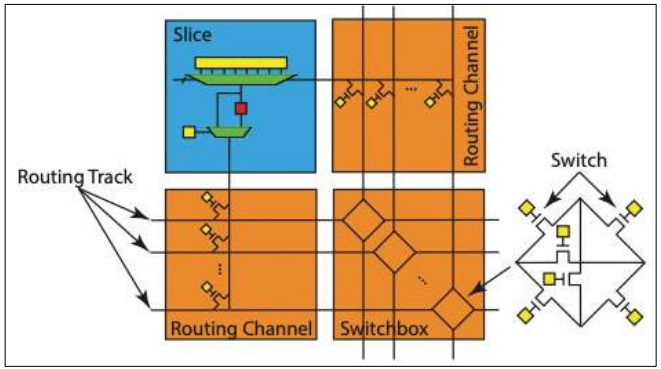
enging

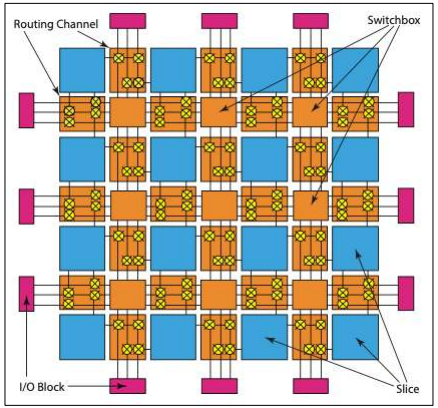
Fermilab

Spatial programming and FPGAs





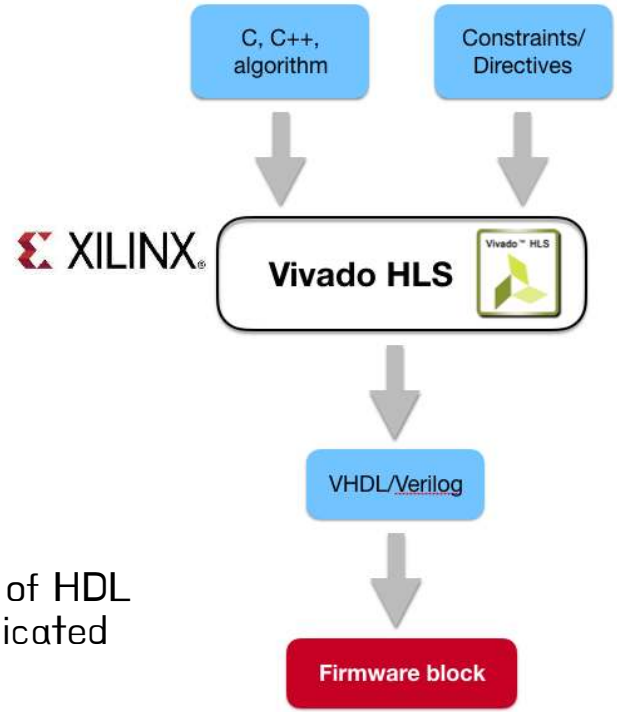




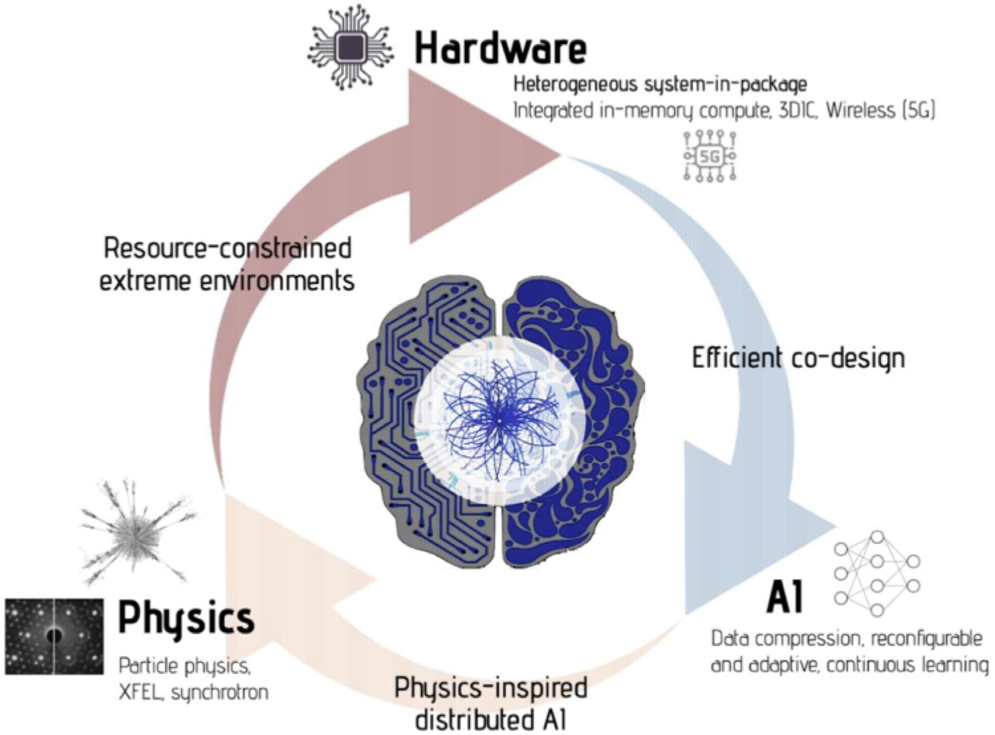
Fermilab

How are FPGAs programmed?

- Hardware Description Languages
  - HDLs are programming languages which describe electronic circuits
- High Level Synthesis
  - Compile from C/C++ to VHDL
  - Pre-processor directives and constraints used to optimize the design
  - Drastic decrease in firmware development time!
- Not totally rainbows and sunshine, often projects are mixes of HDL and HLS but HLS can be used to make kernels or IPs of dedicated algorithms



Efficient codesign

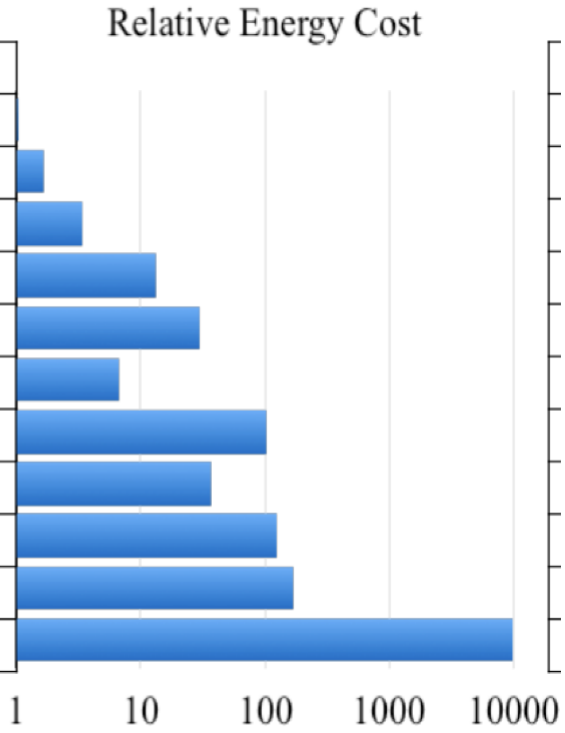


Fermilab

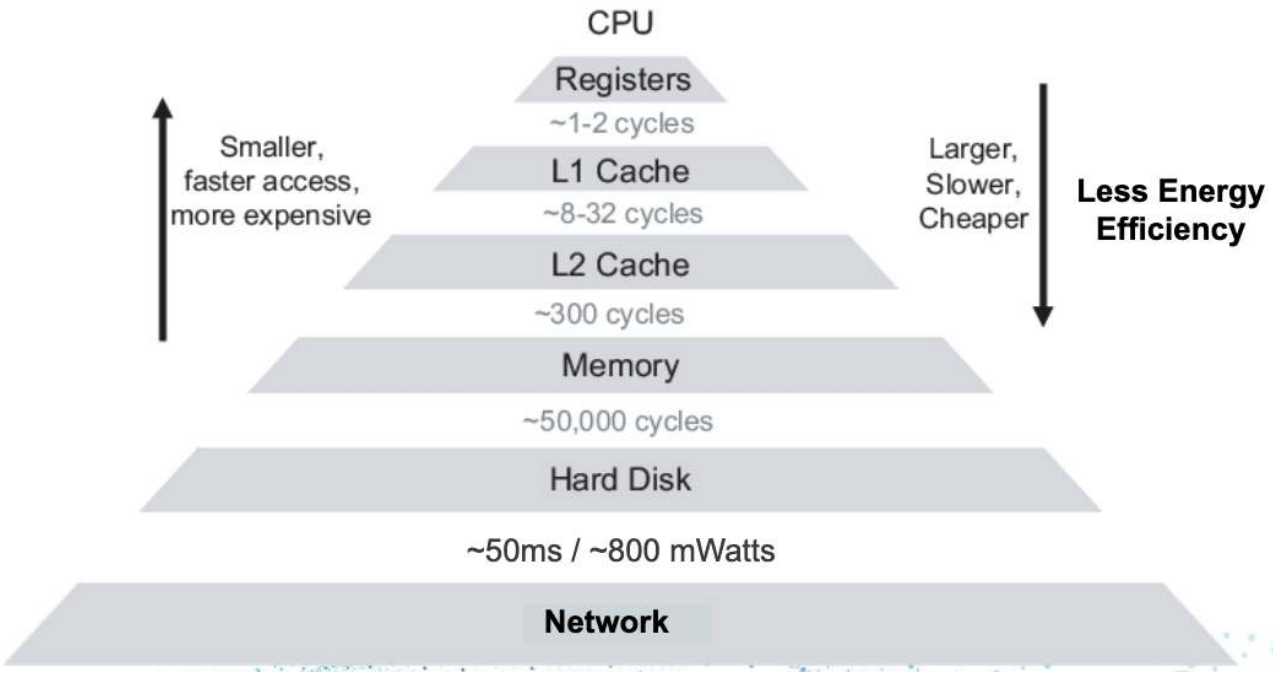
Moving data expensive, computing cheap

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

Adapted from Horowitz

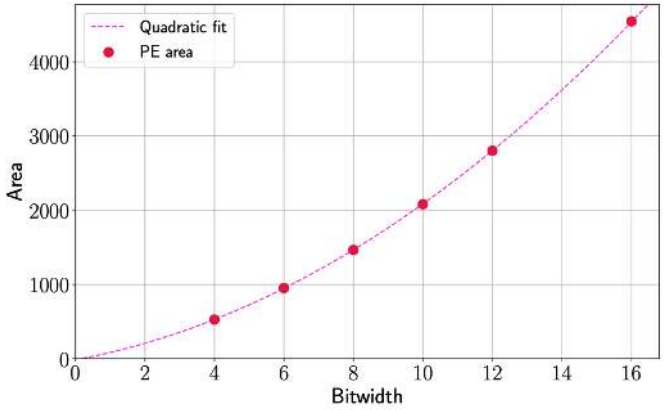


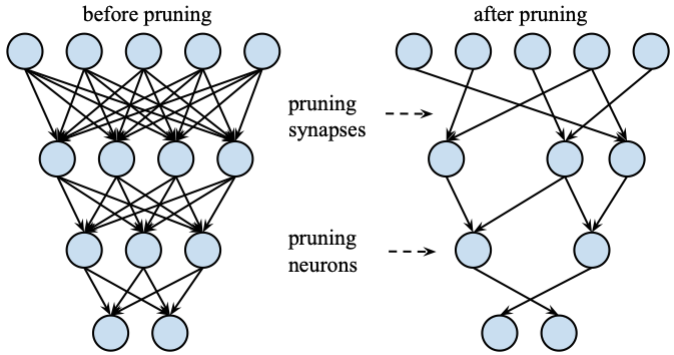
Moving data expensive, computing cheap



Efficient machine learning

- **Computation parallelization/ vectorization and in-memory compute (architecture)**
- **Quantization, reduced precision**
  - For ML, 32-bit floating point is often overkill
  - Integer/fixed-point math at 16,8,7,6,5…1 bits
- **Compression, pruning**
  - maintain the same performance while removing low weight synapses and neurons





Ps and Qs: Quantization-Aware Pruning for Efficient Low Latency Neural Network Inference

Benjamin Hawks [1], **Javier Duarte** [2], **Nicholas J. Fraser** [3], **Alessandro Pappalardo** [3], **Nhan Tran** [1,4], **Yaman Umuroglu** [3]

[1] *Fermi National Accelerator Laboratory, Batavia, IL, United States* [2] *University of California San Diego, La Jolla, CA, United States,* [3] *Xilinx Research, Dublin, Ireland,* [4] *Northwestern University, Evanston, IL, United States*

Model	Precision	BN or L_1	Pruned [%]	BOPs	Accuracy [%]	$\langle \epsilon_b^{\epsilon_s=0.5} \rangle$ [%]	$\langle$AUC$\rangle$ [%]
Nominal	32-bit floating-point	L_1 + BN	0	4,652,832	**76.977**	**0.00171**	**94.335**
Pruning + PTQ	16-bit fixed-point	L_1 + BN	70	631,791	75.01	0.00210	94.229
QAT	6-bit fixed-point	L_1 + BN	0	412,960	76.737	0.00208	94.206
QAP	6-bit scaled-integer	L_1 + BN	**80**	**189,672**	76.602	0.00211	94.197

Developed Quantization-aware pruning procedure:

- Used BOPS as hardware efficiency metric
- Fine-tuning vs. Lottery ticket pruning
- Effect of Batch Norm and L1 reg
- Explored generalizability of QAP-ed models including metrics like neural efficiency
- Bayesian Optimization/structured pruning vs. unstructured pruning

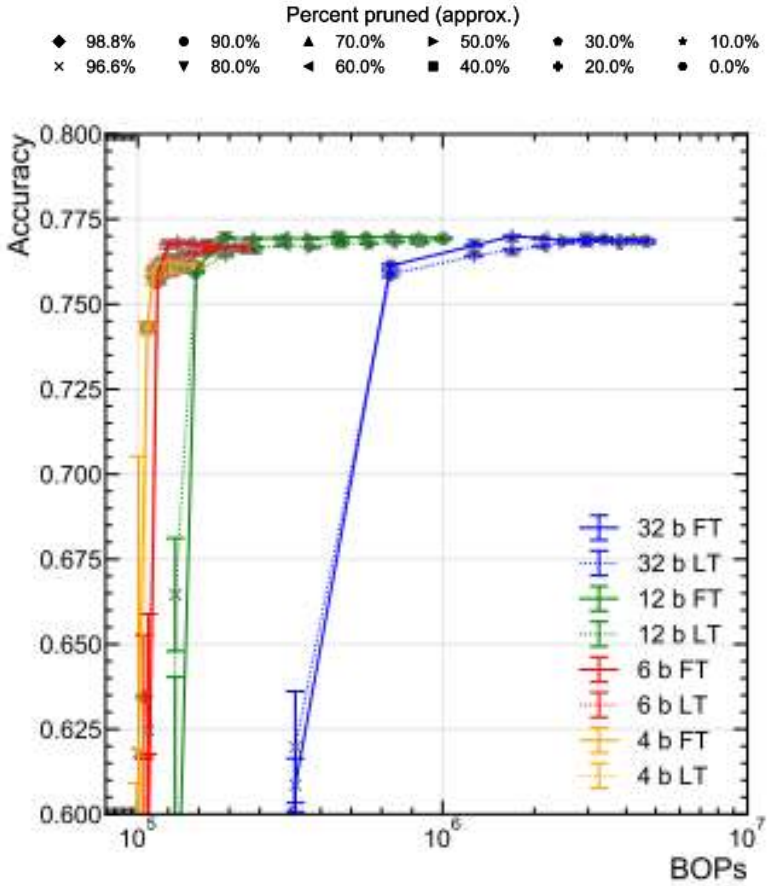
Ps and Qs: Quantization-Aware Pruning for Efficient Low Latency Neural Network Inference

Benjamin Hawks [1], **Javier Duarte** [2], **Nicholas J. Fraser** [3], **Alessandro Pappalardo** [3], **Nhan Tran** [1,4], **Yaman Umuroglu** [3]

[1] *Fermi National Accelerator Laboratory, Batavia, IL, United States* [2] *University of California San Diego, La Jolla, CA, United States,* [3] *Xilinx Research, Dublin, Ireland,* [4] *Northwestern University, Evanston, IL, United States*

Developed Quantization-aware pruning procedure:

- Used BOPS as hardware efficiency metric
- Fine-tuning vs. Lottery ticket pruning
- Effect of Batch Norm and L1 reg
- Explored generalizability of QAP-ed models including metrics like neural efficiency
- Bayesian Optimization/structured pruning vs. unstructured pruning



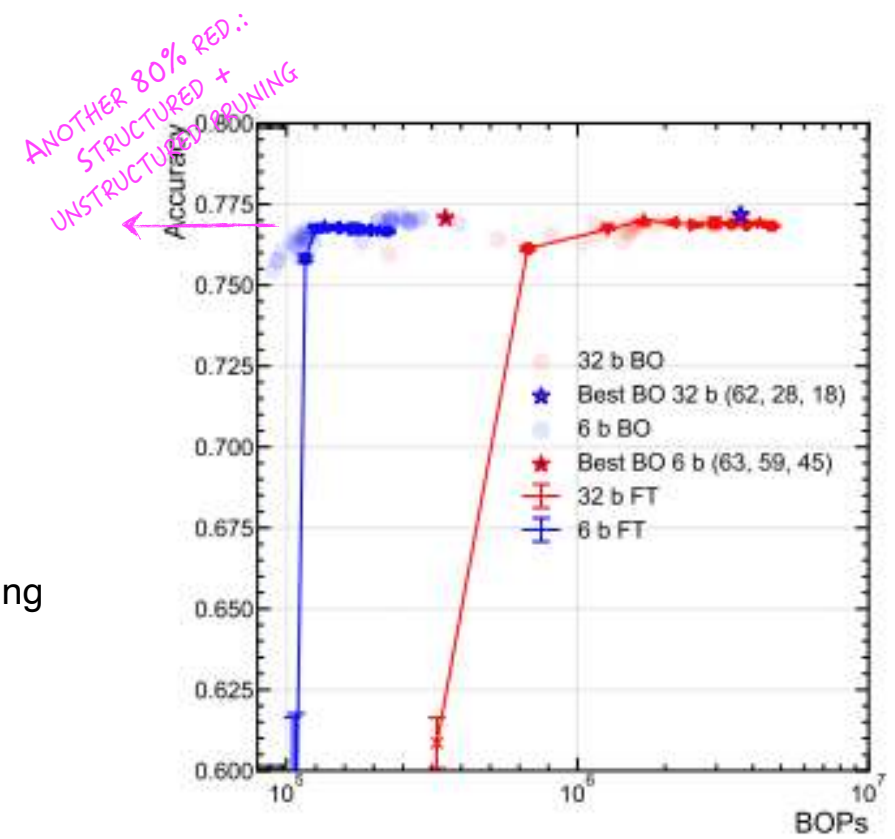
Ps and Qs: Quantization-Aware Pruning for Efficient Low Latency Neural Network Inference

Benjamin Hawks [1], **Javier Duarte** [2], **Nicholas J. Fraser** [3], **Alessandro Pappalardo** [3], **Nhan Tran** [1,4], **Yaman Umuroglu** [3]

[1] *Fermi National Accelerator Laboratory, Batavia, IL, United States* [2] *University of California San Diego, La Jolla, CA, United States,* [3] *Xilinx Research, Dublin, Ireland,* [4] *Northwestern University, Evanston, IL, United States*

Developed Quantization-aware pruning procedure:

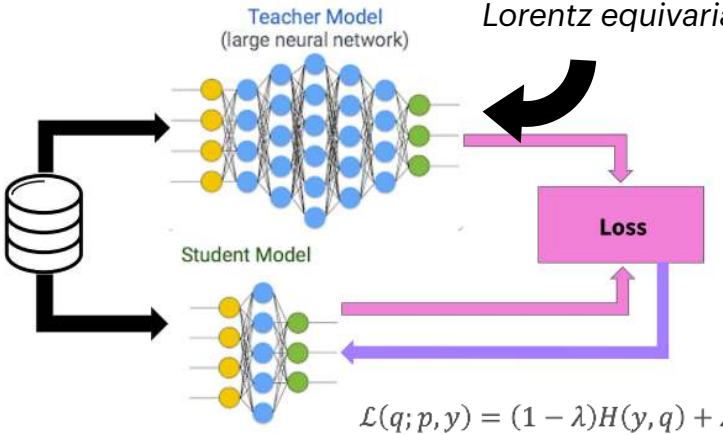
- Used BOPS as hardware efficiency metric
- Fine-tuning vs. Lottery ticket pruning
- Effect of Batch Norm and L1 reg
- Explored generalizability of QAP-ed models including metrics like neural efficiency
- Bayesian Optimization/structured pruning vs. unstructured pruning



Efficient algorithm codesign

More interesting directions — distillation and inductive bias

Teacher model knows about Lorentz equivariance



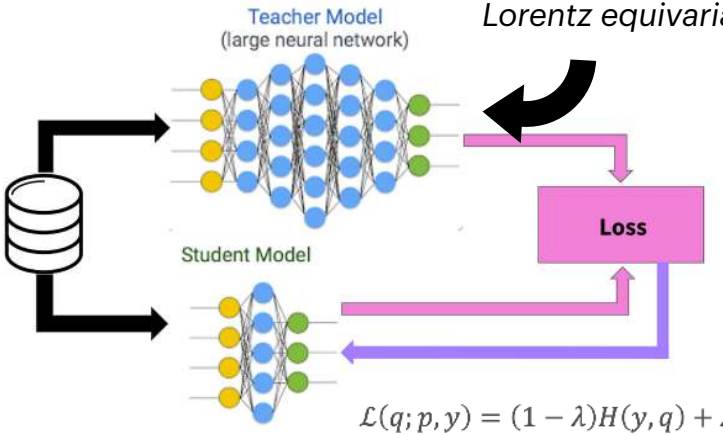
$$\mathcal{L}(q; p, y) = (1 - \lambda)H(y, q) + \lambda D_{KL}(\tilde{p}||\tilde{q})$$

q: student's prediction, p: teacher's prediction, y: ground truth, $\tilde{p}$: temperature soften distribution.

Efficient algorithm codesign

More interesting directions — distillation and inductive bias

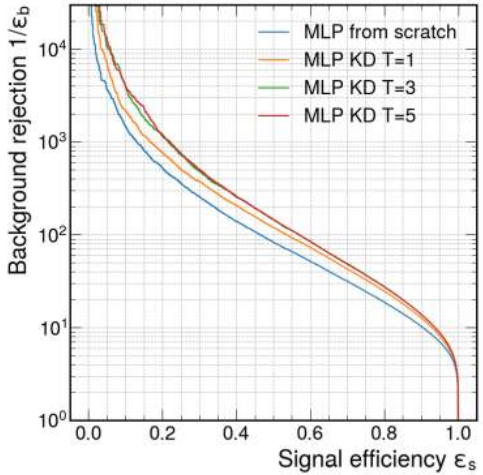
Teacher model knows about Lorentz equivariance



$$\mathcal{L}(q; p, y) = (1 - \lambda)H(y, q) + \lambda D_{KL}(\tilde{p}||\tilde{q})$$

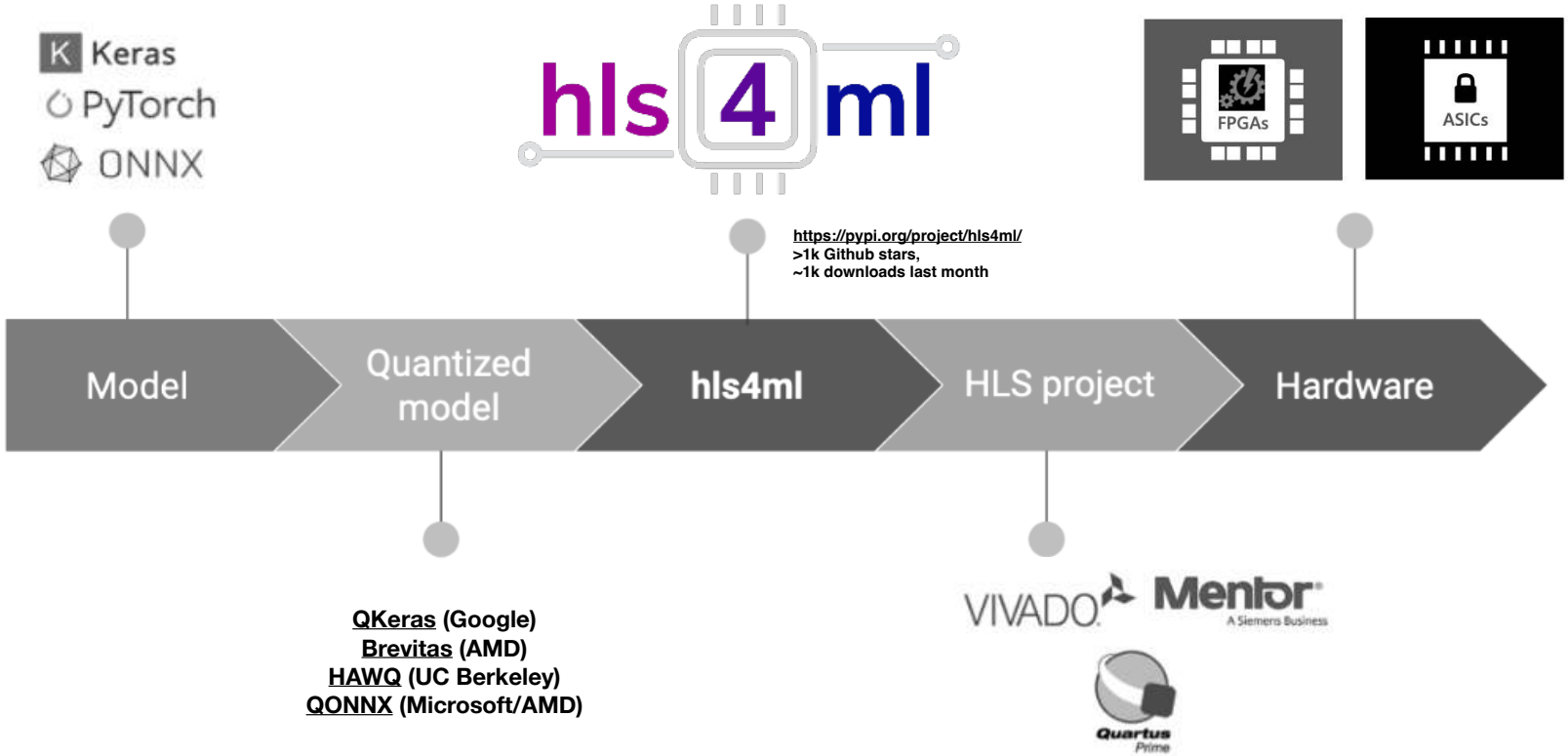
q: student's prediction, p: teacher's prediction, y: ground truth, $\tilde{p}$: temperature soften distribution.

Model performance improves with distillation of expert knowledge, and more robust (see talk)



Fermilab

Efficient hardware - algorithm codesign



Hardware - algorithm codesign

Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

What kind of platform?

Latency?
Pipeline Interval?

How many resources?

Area/power?
Radiation?
Cryo?

Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

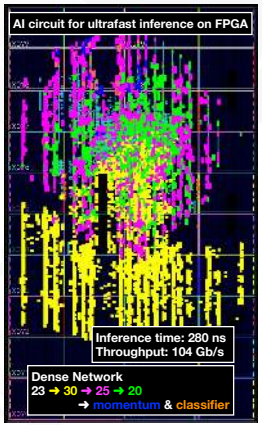
Neural architecture search/
Hyperparameter optimization

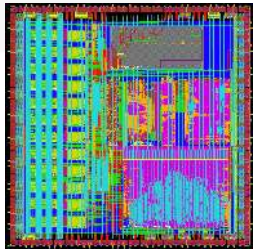
What kind of platform?

Latency?
Pipeline Interval?

How many resources?

Area/power?
Radiation?
Cryo?





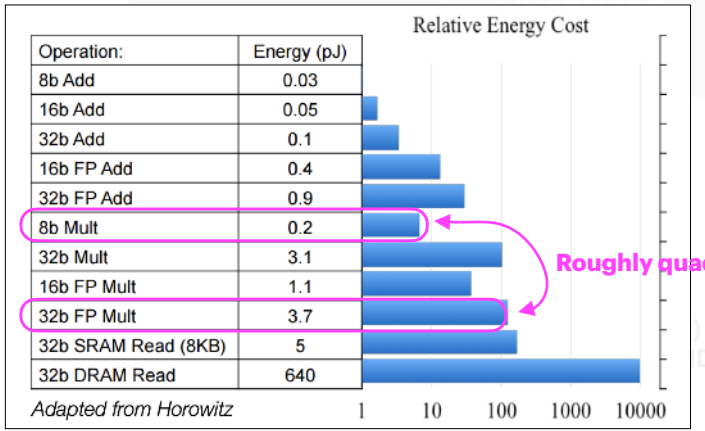
Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Quantize network



See tools like:
QKeras
HAWQ
Brevitas

What kind of platform?

Latency?
Pipeline Interval?

How many resources?

Area/power?
Radiation?
Cryo?

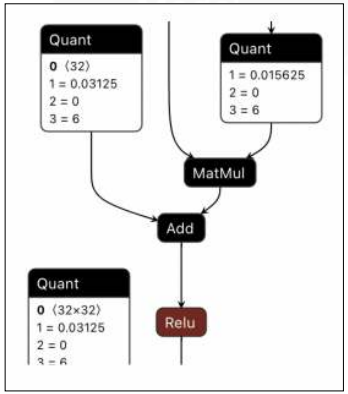
Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Quantize network



Intermediate (quantized) representations

See proposal for QONNX

What kind of platform?

Latency?
Pipeline Interval?

How many resources?

Area/power?
Radiation?
Cryo?

Hardware - algorithm codesign

Physics requirements

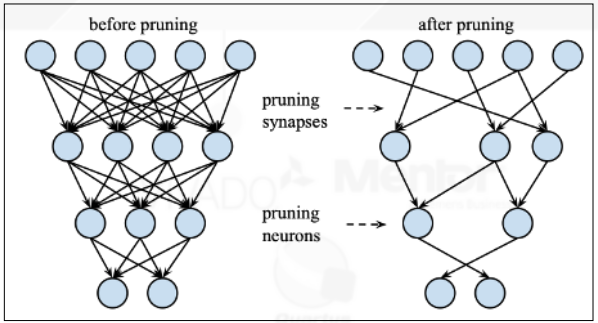
Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Quantize network

Intermediate (quantized)
representations

Pruning/sparsity?



What kind of platform?

Latency?
Pipeline Interval?

How many
resources?

Area/power?
Radiation?
Cryo?

Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Quantize network

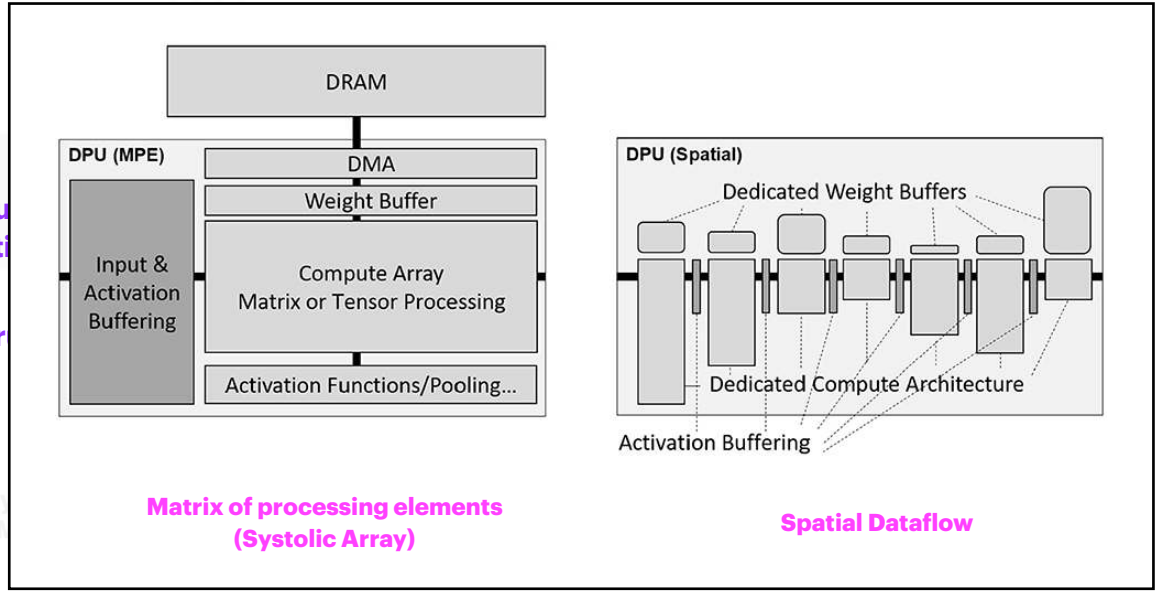
Intermediate (qu
representati

Pr

What kind of platform?

Latency?
Pipeline Interval?

Microarchitecture



Matrix of processing elements (Systolic Array)

Spatial Dataflow

Hardware - algorithm codesign

Physics requirements

Data representation → ML architecture

Neural architecture search/ Hyperparameter optimization

Microarchitecture

Quantize network

Intermediate (quantized) representations

Pruning/sparsity?

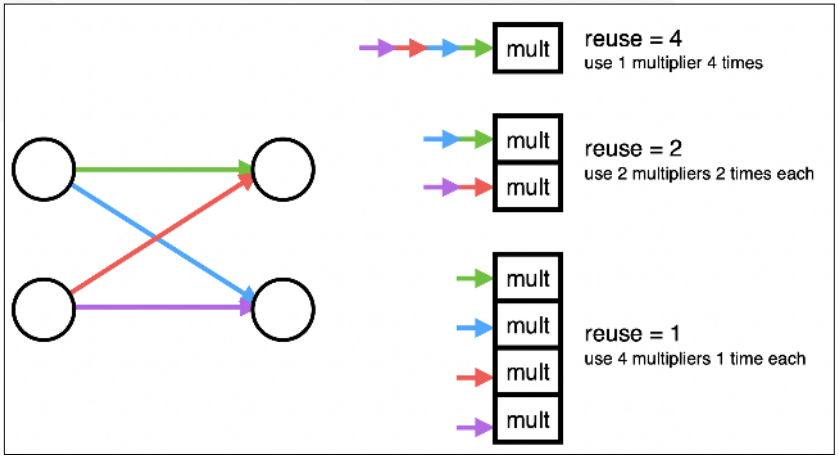
Parallelization

What kind of platform?

Latency? Pipeline Interval?

How many resources?

Area/power? Radiation? Cryo?



Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Quantize network

Intermediate (quantized)
representations

Pruning/sparsity?

Microarchitecture

Parallelization

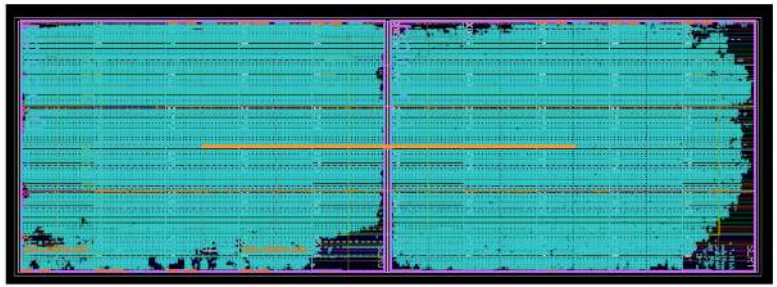
What kind of platform?

Latency?
Pipeline Interval?

How many
resources?

Area/power?
Radiation?
Cryo?

Synthesize, validate design,
satisfy design rules/timing, integration



BNL711 FELIX Firmware Floorplanning

Hardware - algorithm codesign

Physics requirements

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Quantize network

Intermediate (quantized)
representations

Pruning/sparsity?

Multi-objective
design space optimization

Microarchitecture

Parallelization

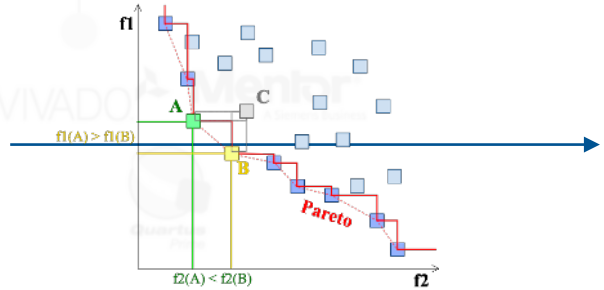
What kind of platform?

Latency?
Pipeline Interval?

How many
resources?

Area/power?
Radiation?
Cryo?

Synthesize, validate design,
satisfy design rules/timing



Efficient codesign

Why hls4ml

- open-source
- Community-supported
- User-driven
- Accessible and usable

https://github.com/fastmachinelearning/hls4ml-tutorial

Check performance

Check the accuracy and make a ROC curve

In [ ]:
```
import plotting
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

y_keras = model.predict(X_test)
print("Accuracy: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_keras, axis=1))))
plt.figure(figsize=(9, 9))
_ = plotting.makeRoc(y_test, y_keras, le.classes_)
```

Convert the model to FPGA firmware with hls4ml

Now we will go through the steps to convert the model we trained to a low-latency optimized FPGA firmware with hls4ml. First, we will evaluate its classification performance to make sure we haven't lost accuracy using the fixed-point data types. Then we will synthesize the model with Vivado HLS and check the metrics of latency and FPGA resource usage.

Make an hls4ml config & model

The hls4ml Neural Network inference library is controlled through a configuration dictionary. In this example we'll use the most simple variation, later exercises will look at more advanced configuration.

In [ ]:
```
import hls4ml

config = hls4ml.utils.config_from_keras_model(model, granularity='model')
print("-----------------------------------")
print("Configuration")
plotting.print_dict(config)
print("-----------------------------------")
hls_model = hls4ml.converters.convert_from_keras_model(
    model, hls_config=config, output_dir='model_1/hls4ml_prj', part='xcu250-figd2104-2L-e'
)
```

Let's visualise what we created. The model architecture is shown, annotated with the shape and data types

In [ ]:
```
hls4ml.utils.plot_model(hls_model, show_shapes=True, show_precision=True, to_file=None)
```

Compile, predict

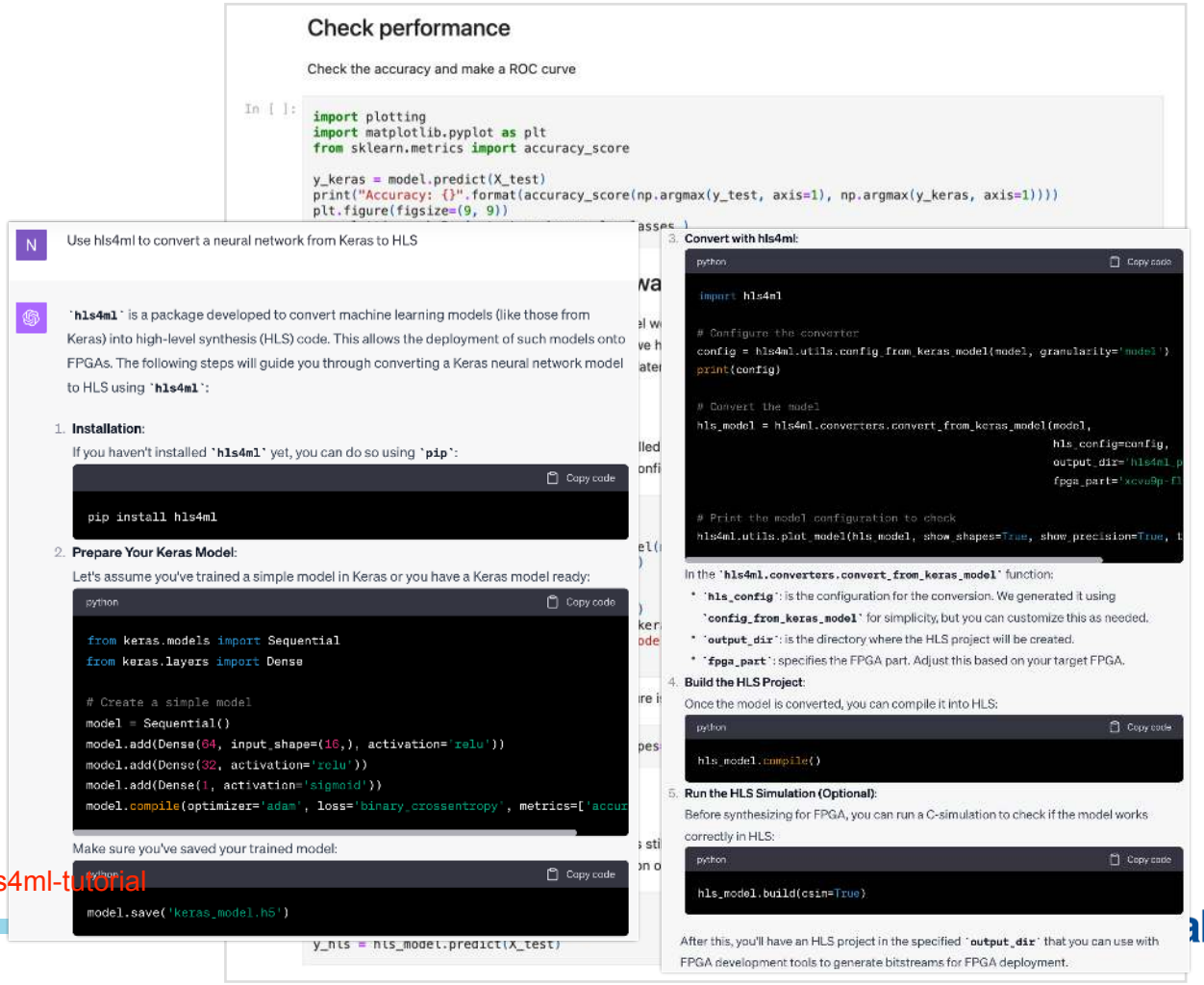
Now we need to check that this model performance is still good. We compile the hls_model, and then use `hls_model.predict` to execute the FPGA firmware with bit-accurate emulation on the CPU.

In [ ]:
```
hls_model.compile()
X_test = np.ascontiguousarray(X_test)
y_hls = hls_model.predict(X_test)
```

Efficient codesign

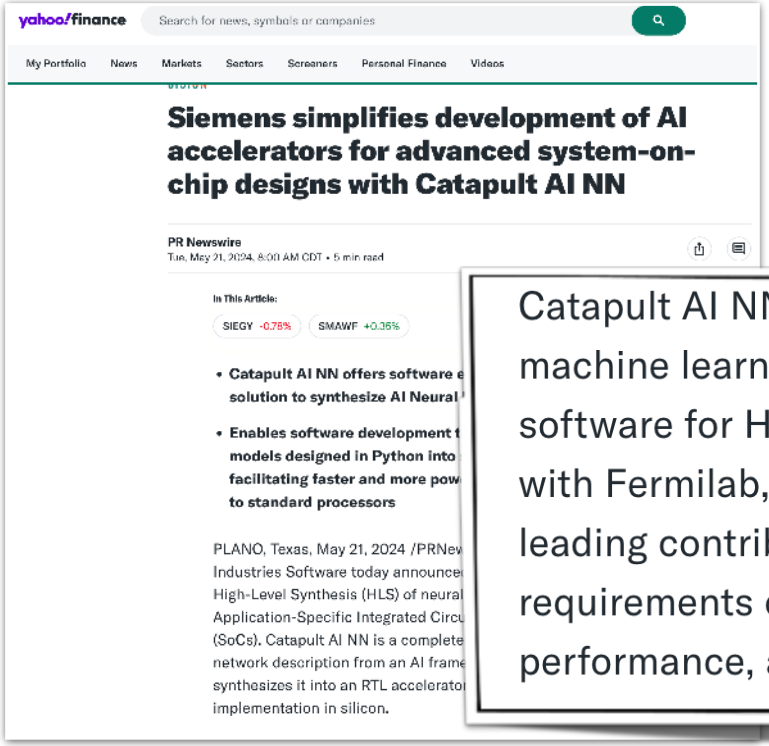
Why hls4ml

- open-source
- Community-supported
- User-driven
- Accessible and usable



https://github.com/fastmachinelearning/hls4ml-tutorial

Efficient codesign



Catapult AI NN brings together hls4ml, an open-source package for machine learning hardware acceleration, and Siemens' Catapult™ HLS software for High-Level Synthesis. Developed in close collaboration with Fermilab, a U.S. Department of Energy Laboratory, and other leading contributors to hls4ml, Catapult AI NN addresses the unique requirements of machine learning accelerator design for power, performance, and area on custom silicon.

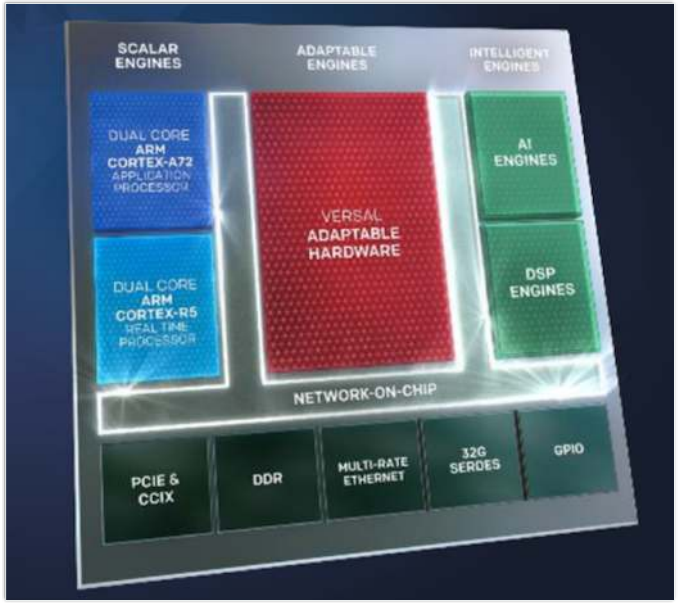
Fermilab

Efficient codesign - emerging methods

- Emerging computing architectures
- New microelectronics technologies
- Efficient neural algorithms, e.g. spiking

Efficient codesign - emerging methods

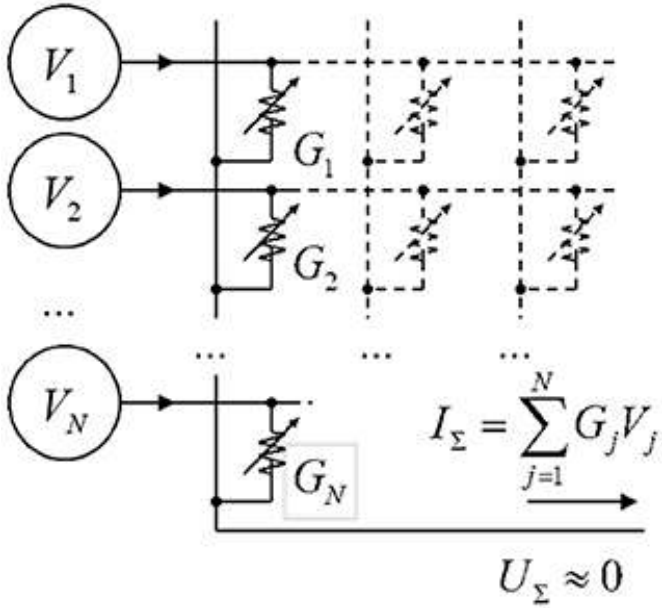
- Emerging computing architectures
- New microelectronics technologies
- Efficient neural algorithms, e.g. spiking



Fermilab

Efficient codesign - emerging methods

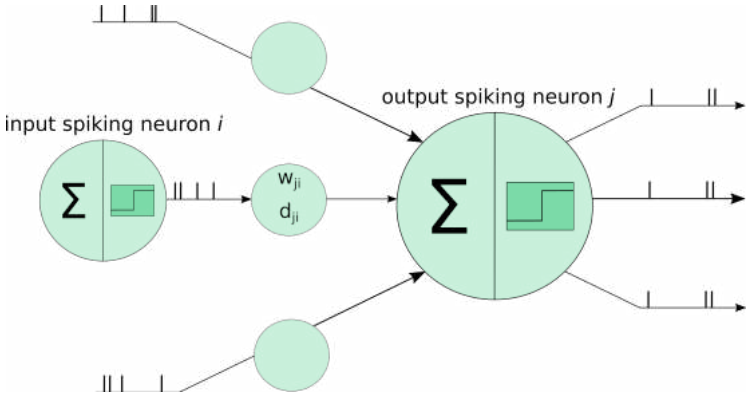
- Emerging computing architectures
- New microelectronics technologies
- Efficient neural algorithms, e.g. spiking



Fermilab

Efficient codesign - emerging methods

- Emerging computing architectures
- New microelectronics technologies
- Efficient neural algorithms, e.g. spiking



Fermilab

Outlook

Fast ML for Science

Embedding ML into our experiments with extreme requirements brings radical new capabilities, accelerates scientific discovery, and spurs technological innovation

Fast ML for Science

Embedding ML into our experiments with extreme requirements brings radical new capabilities, accelerates scientific discovery, and spurs technological innovation

Intelligent edge of tomorrow

We are developing novel ML techniques and accessible tools co-designed with cutting edge hardware for science while collaborating with researchers and industry

Fast ML for Science

Embedding ML into our experiments with extreme requirements brings radical new capabilities, accelerates scientific discovery, and spurs technological innovation

Intelligent edge of tomorrow

We are developing novel ML techniques and accessible tools co-designed with cutting edge hardware for science while collaborating with researchers and industry

Outlook

Powerful intelligent sensing being demonstrated across a wide array of applications; Continuing to advance ML methods and hardware development to enable ultra-fast automated experimentation to enable future ground-breaking discoveries!