

Efficient Machine Learning Model Design

Techniques for Fast Inference

Elham E Khoda

University of British Columbia, Vancouver, Canada

Machine Learning for Fundamental Physics School 2026

June 5, 2026

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

- What is pruning? How to formulate pruning?
- Determine pruning granularity and criteria
- Network performance after pruning

ML on FPGA

hls4ml and Trigger applications

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

- What is pruning? How to formulate pruning?
- Determine pruning granularity and criteria
- Network performance after pruning

ML on FPGA

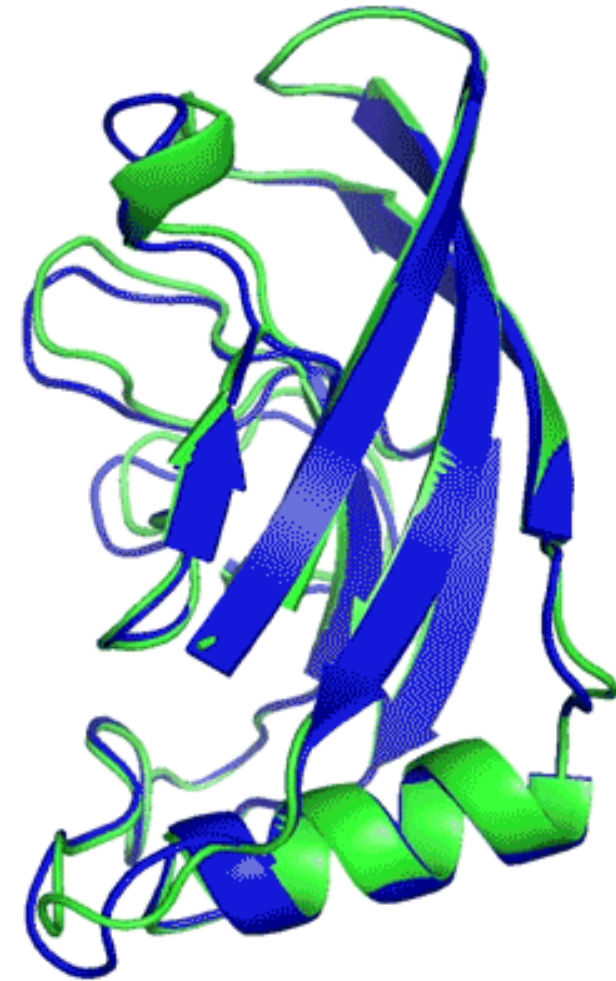
hls4ml and Trigger applications

Large Models are everywhere

Protein folding



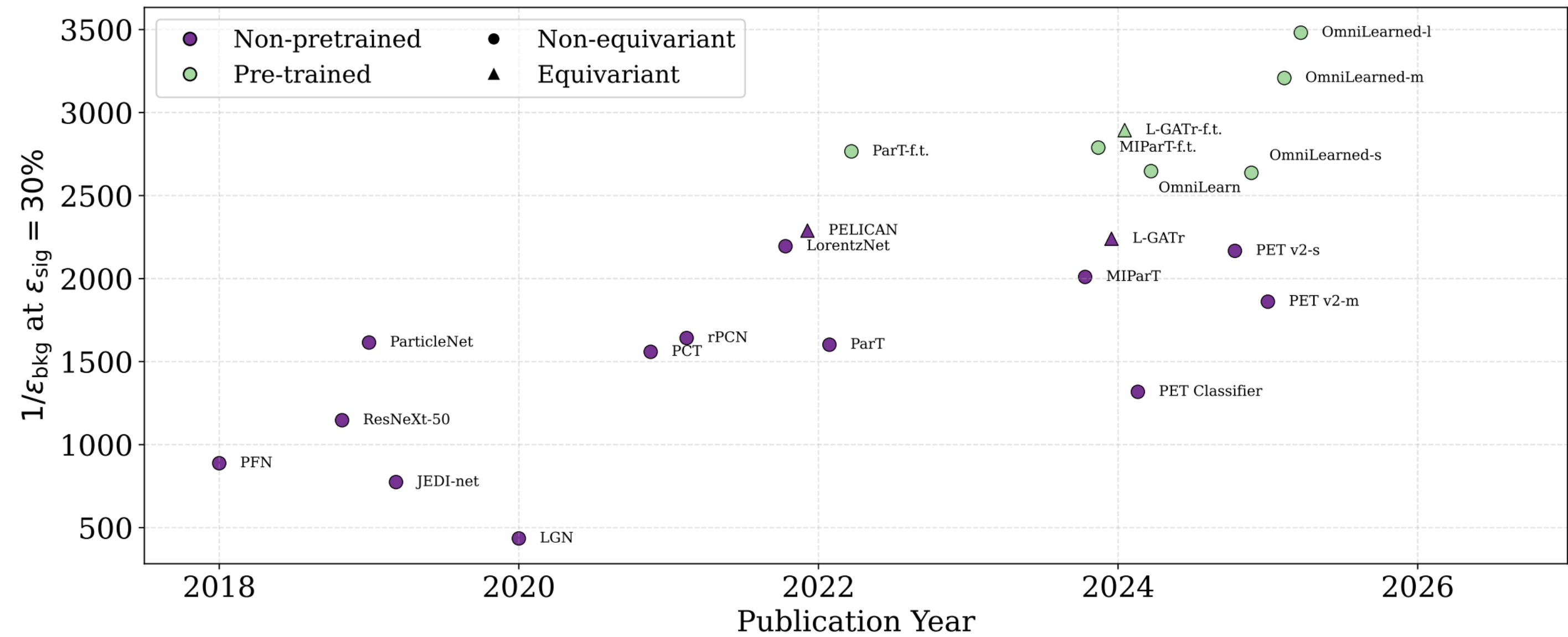
T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



T1049 / 6y4f
93.3 GDT
(adhesin tip)

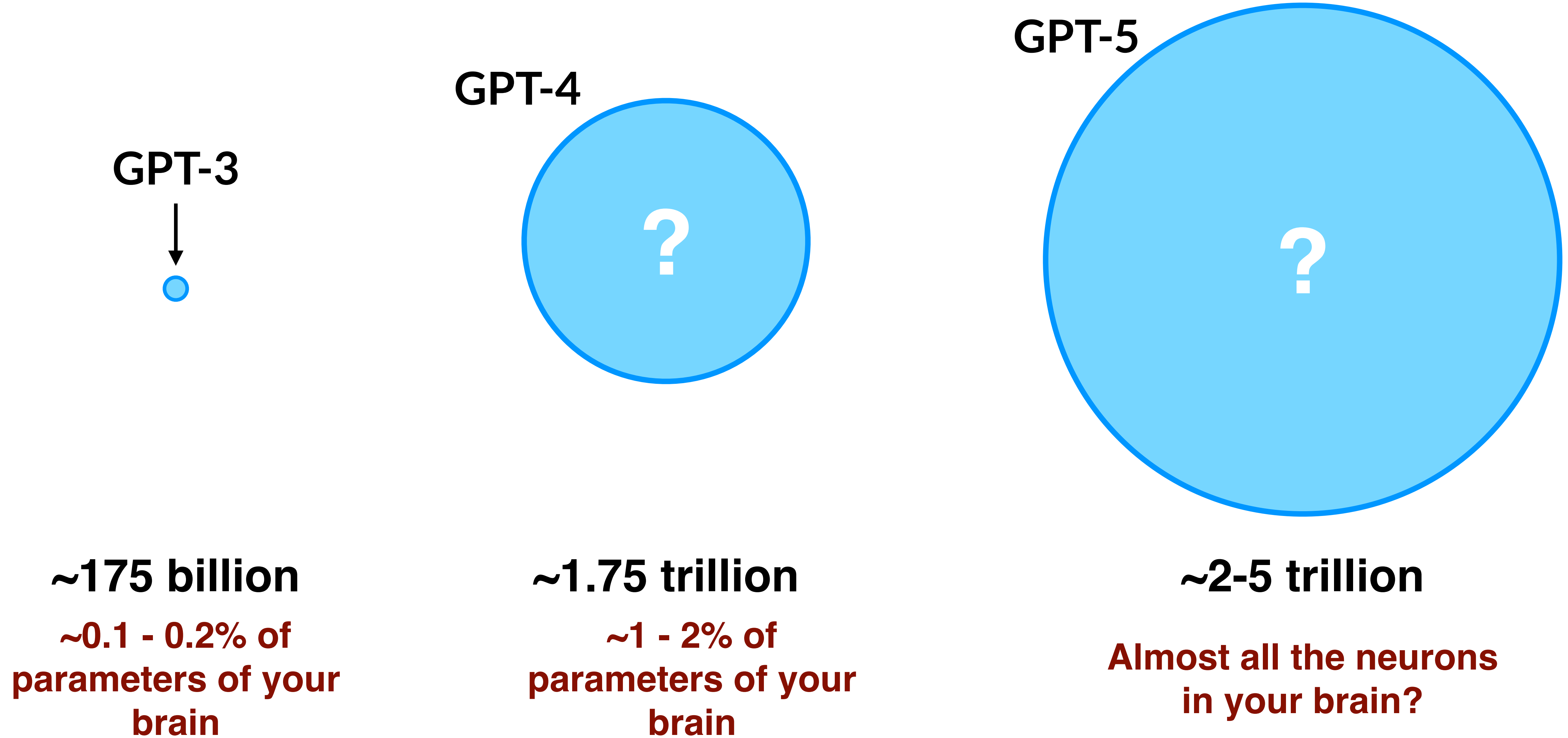
- Experimental result
- Computational prediction

Particle physics jet tagging

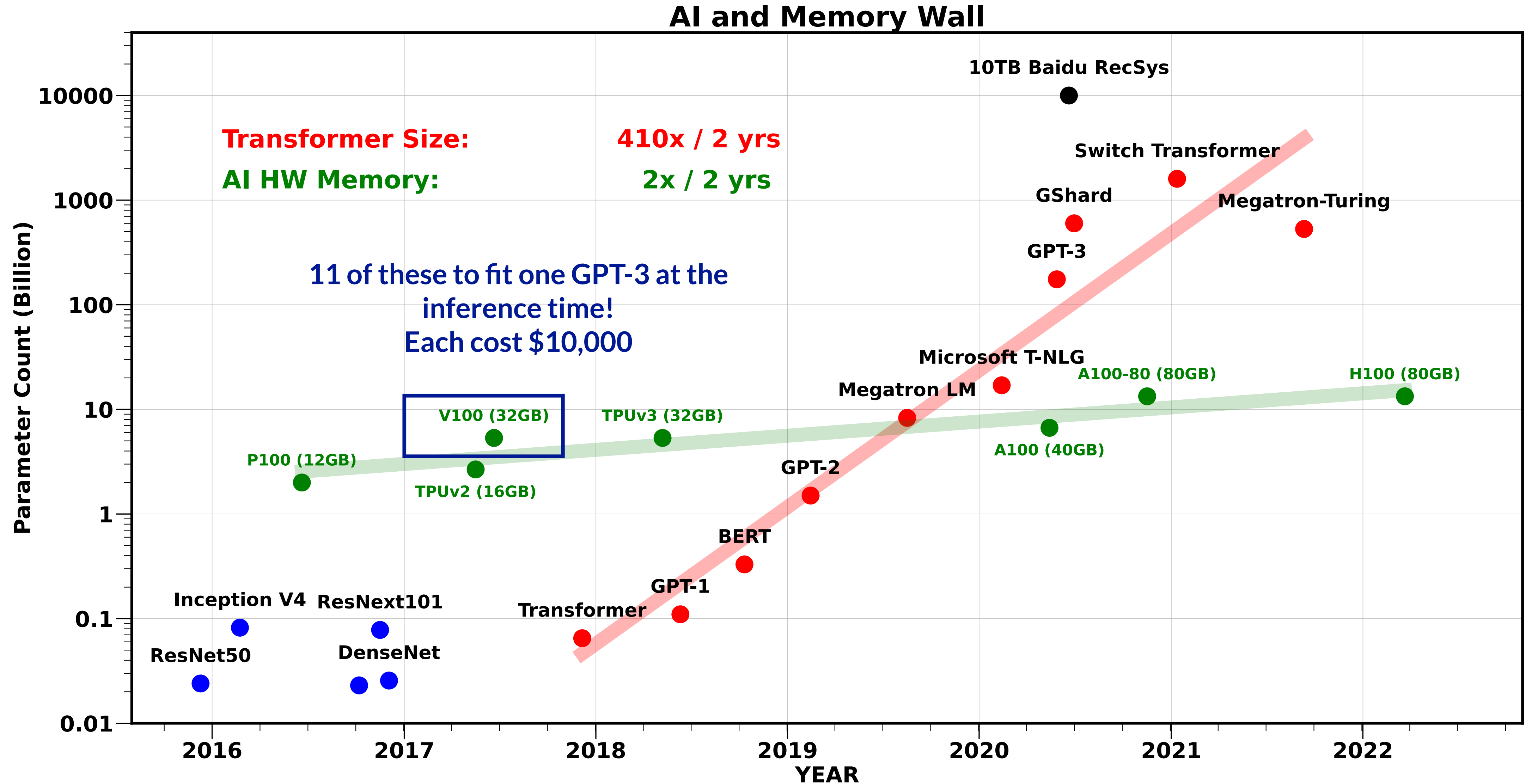


[arXiv:2510.24066](https://arxiv.org/abs/2510.24066)

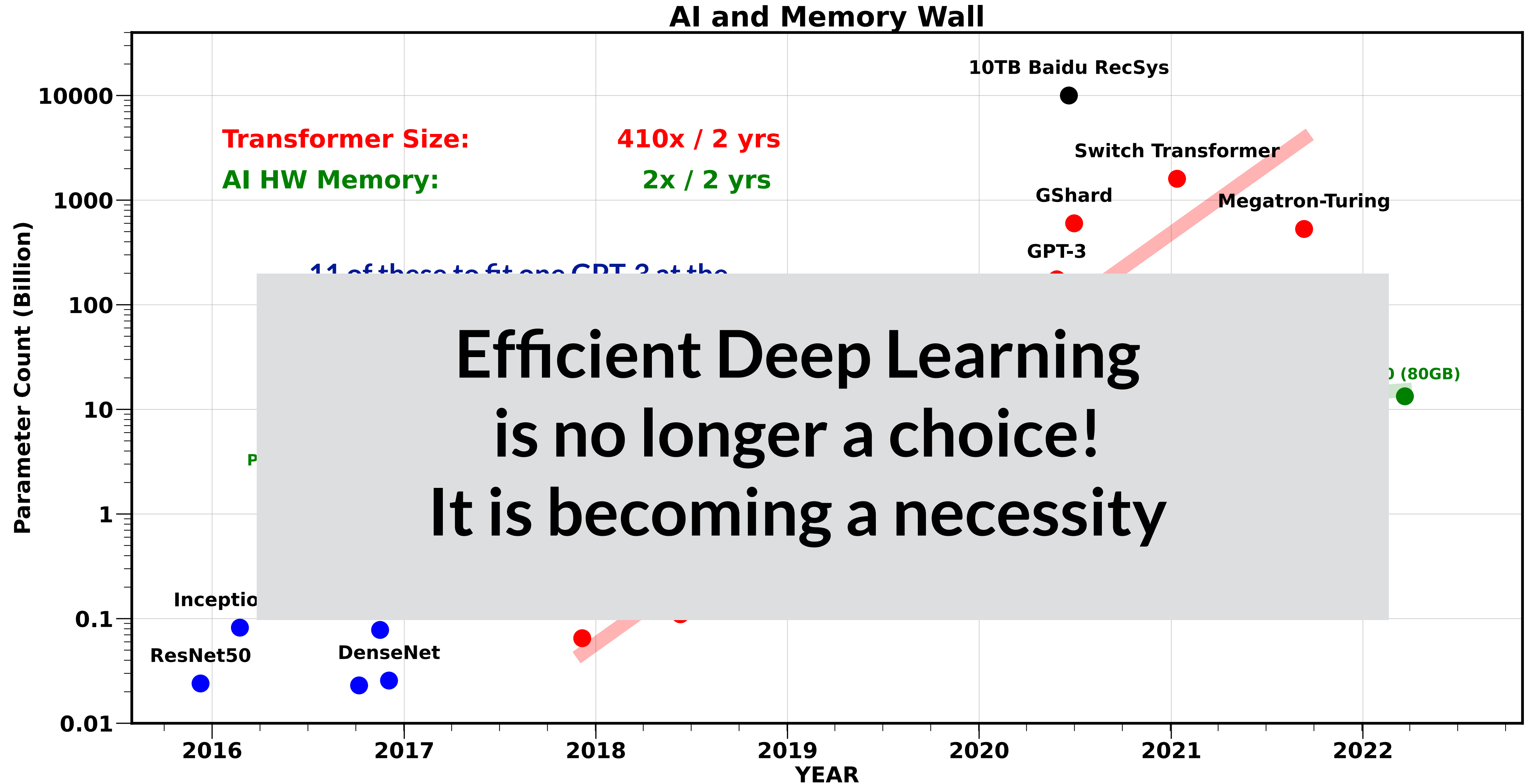
Large Language Model Parameters



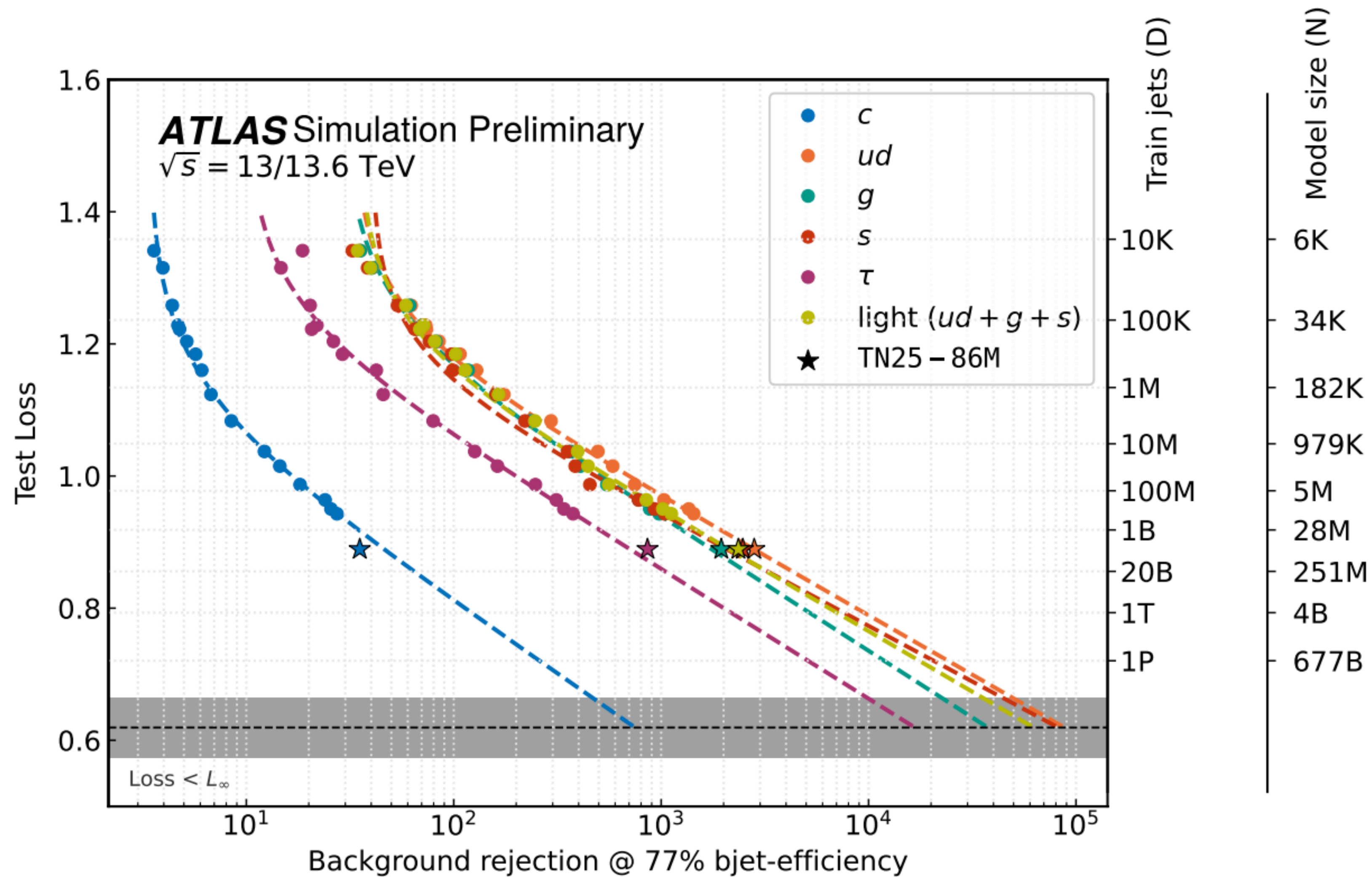
AI and Memory Wall



AI and Memory Wall



Particle Physics Models are also getting larger



ATL-SOFT-PUB-2026-002

Question #1

What challenges arise as machine learning models continue to grow in size?

Think about it for 30 seconds and share your answer

Question #1

What challenges arise as machine learning models continue to grow in size?

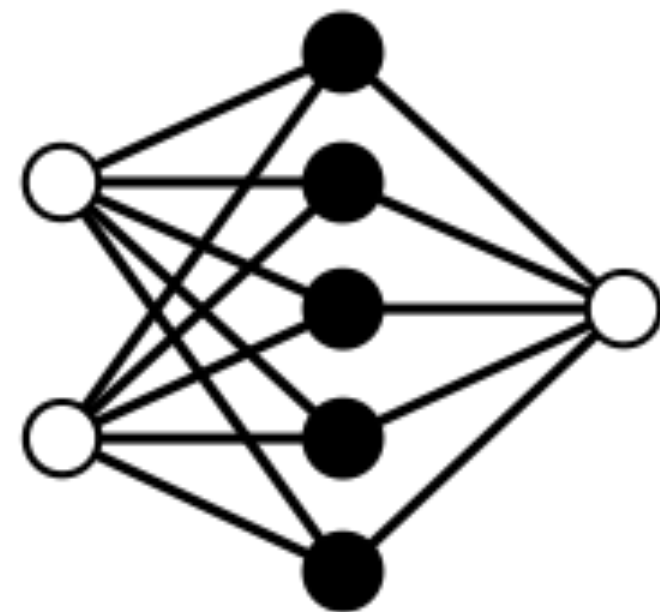
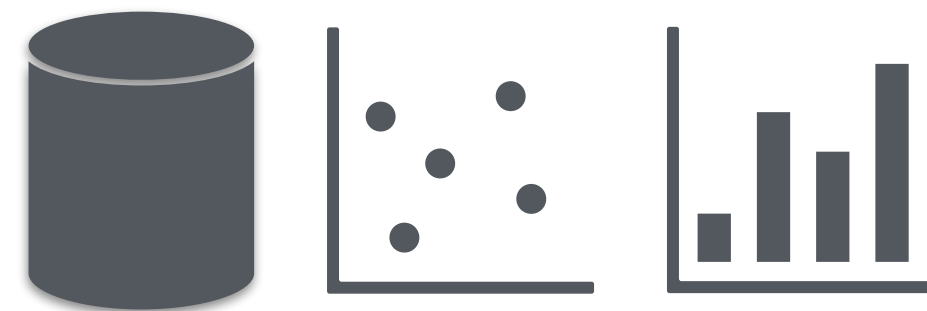
Think about it for 30 seconds and share your answer

- High computing cost
- Greater memory requirement
- Slower Inference
- Increased Energy Consumption
- Deployment Challenges

ML Model Training vs Inference

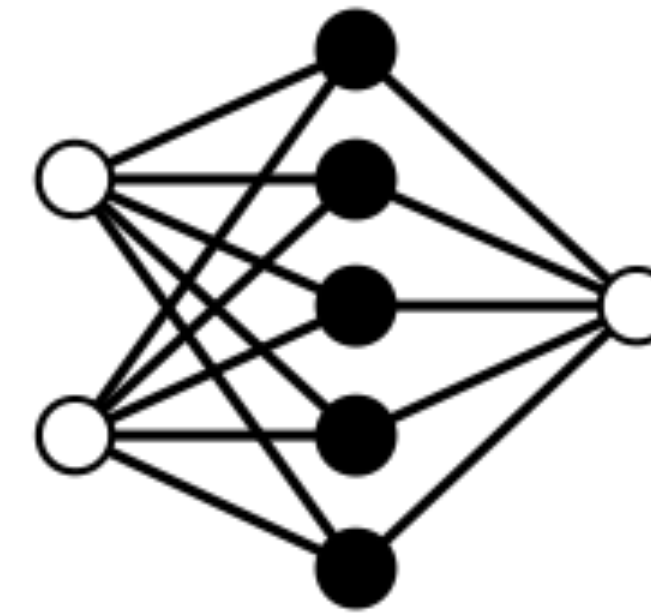
Model training

Learning the model parameters



Model inference

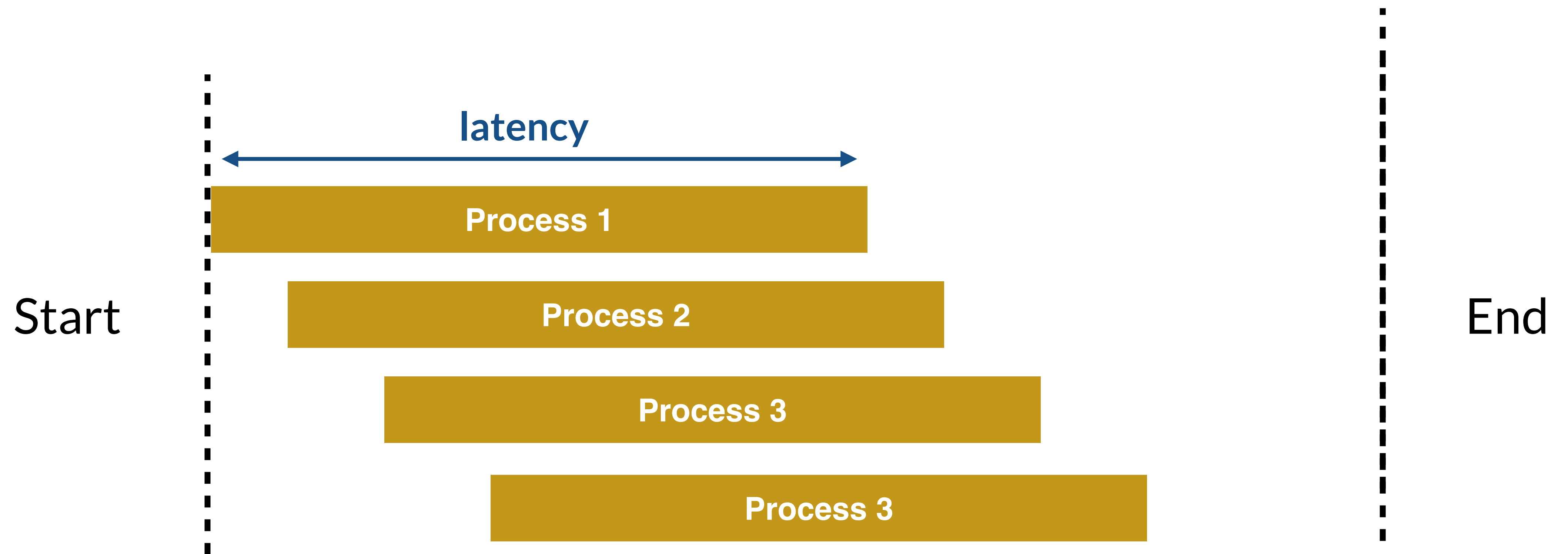
prediction from a trained model



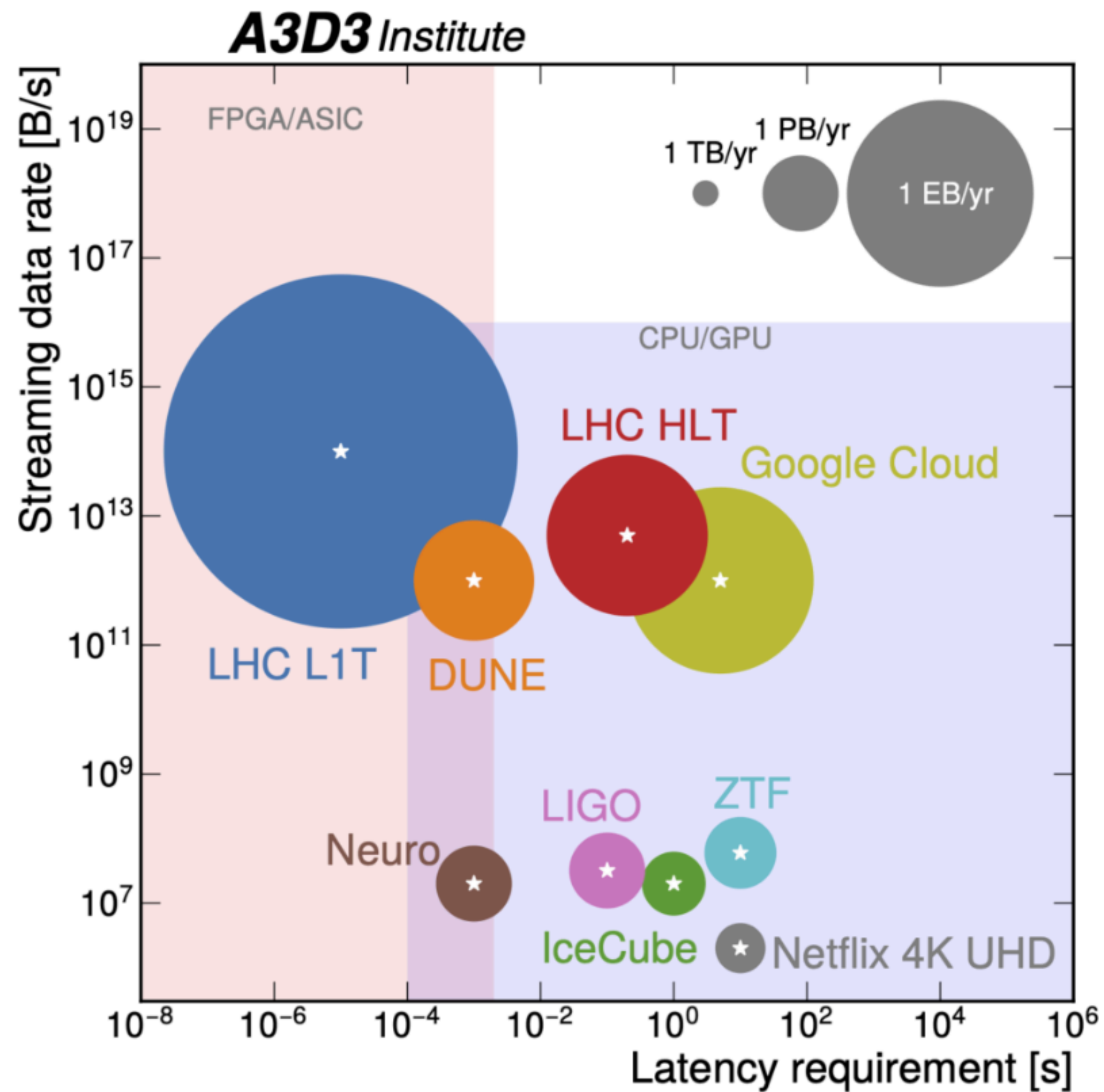
Latency vs Throughput

Latency: Time taken to process a single item / event

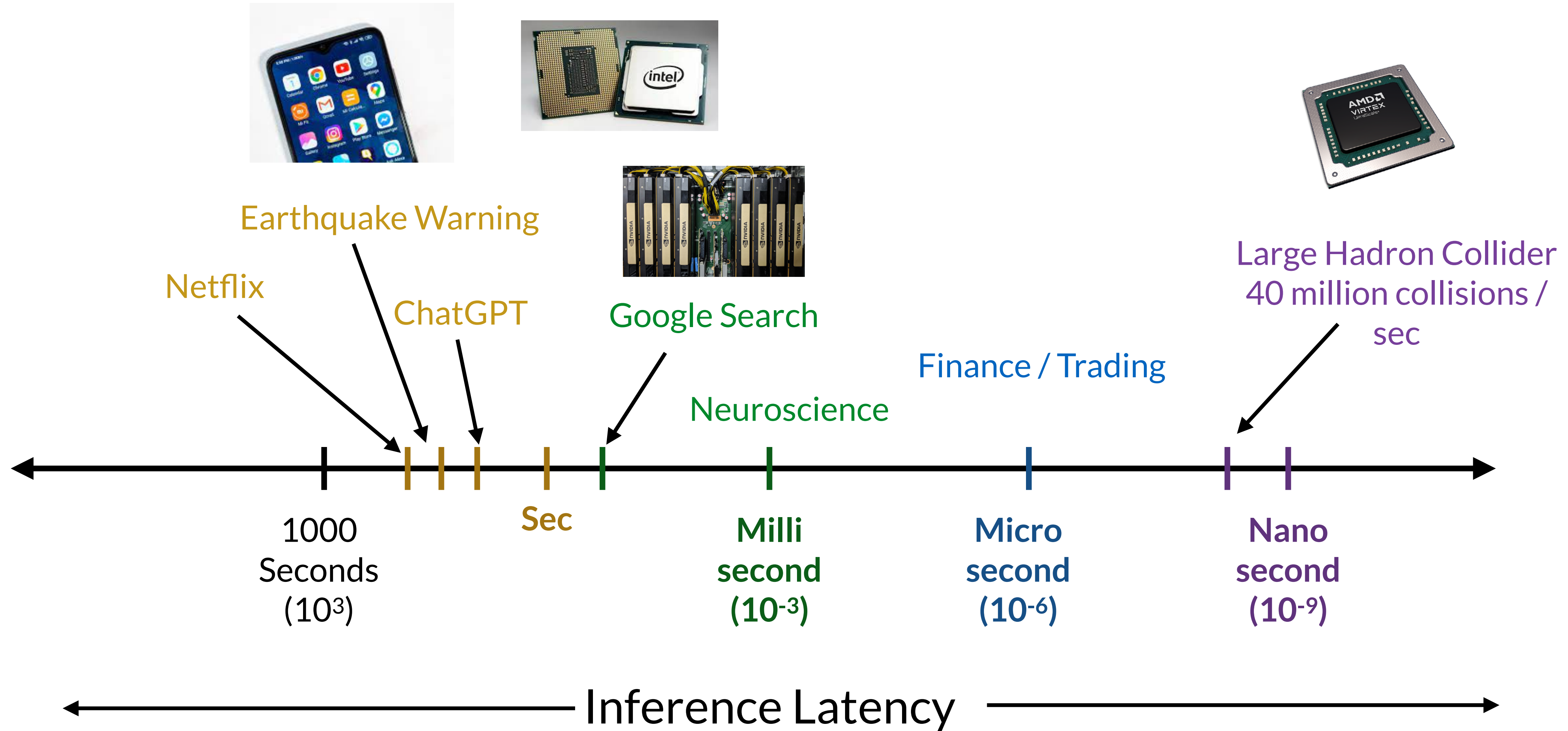
Throughput: total number of tasks / events get processed over a specific period of time. (Events / sec)



Low Latency and High Throughput



Machine Learning Inference



Computing Hardware

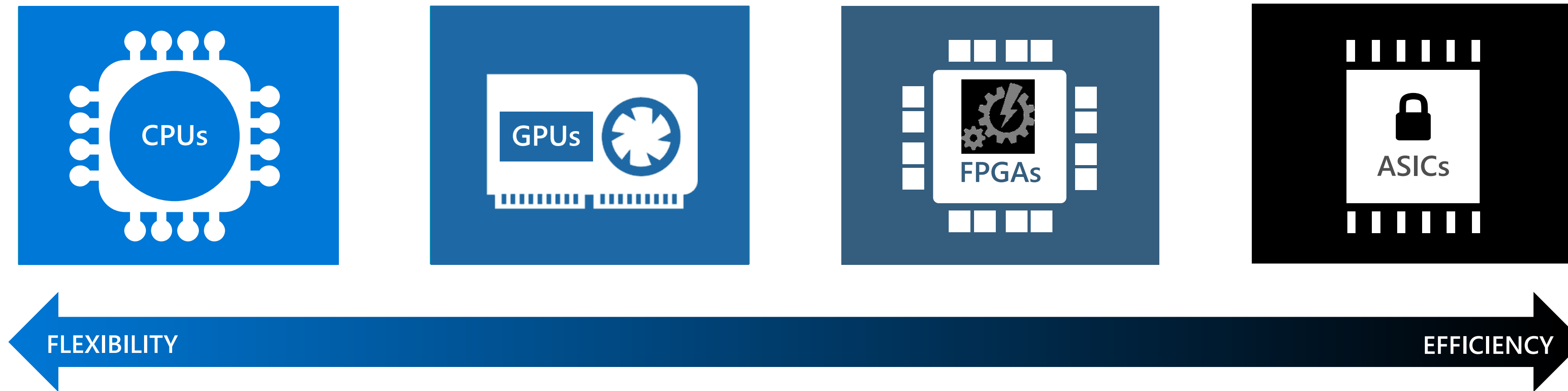


Image: [Microsoft](#)

ML Inference in Particle Physics

GPT-3 Inference (prediction)

Resource: 11 interconnected GPUs

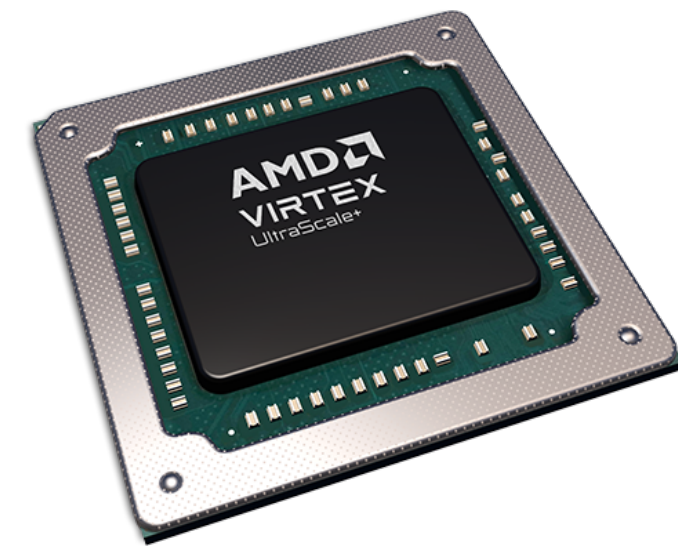
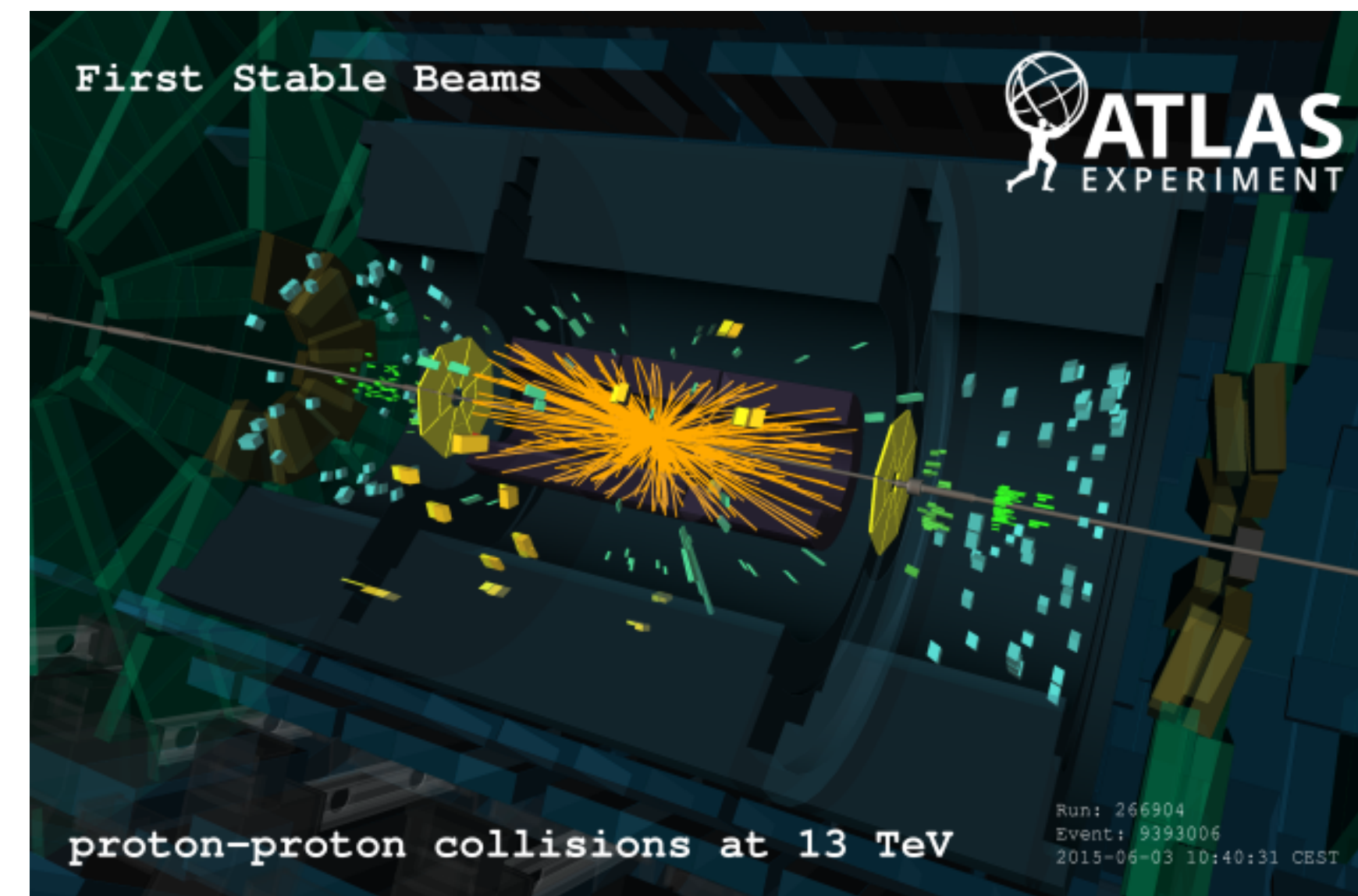
Median Latency = time it takes give you output
Median Latency ~ 4-10 seconds



Large Hadron Collider Real-time event filter

Resource: One single chip

Median Latency: 10^{-9} seconds



Computing Hardware

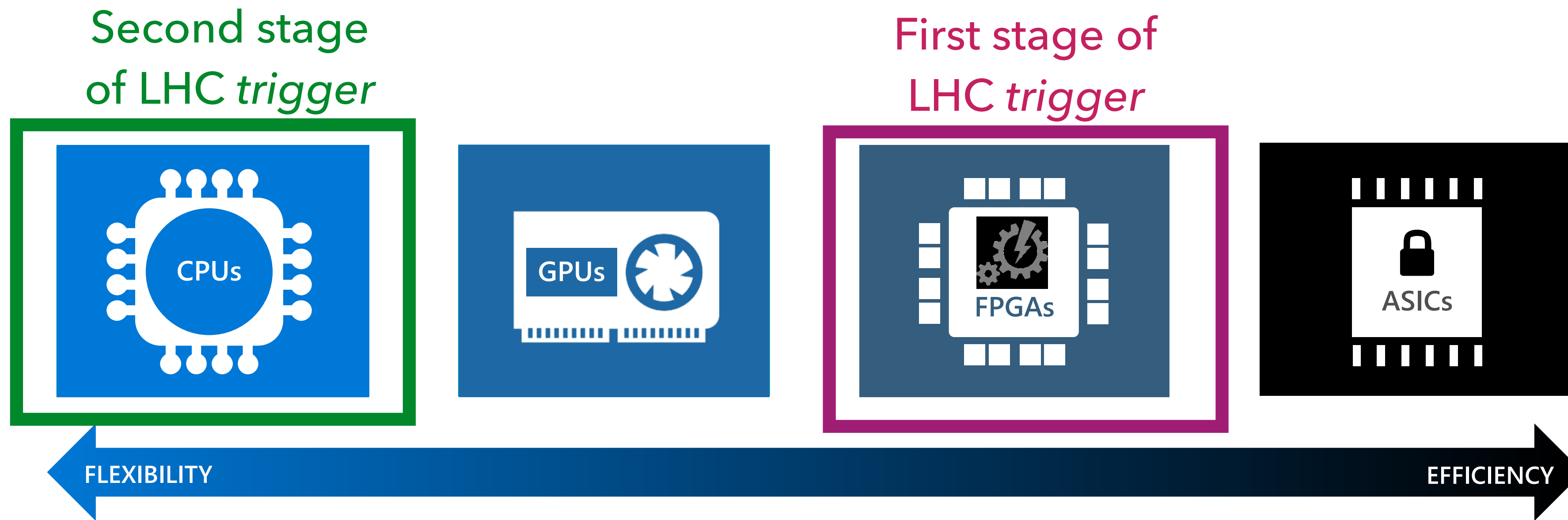
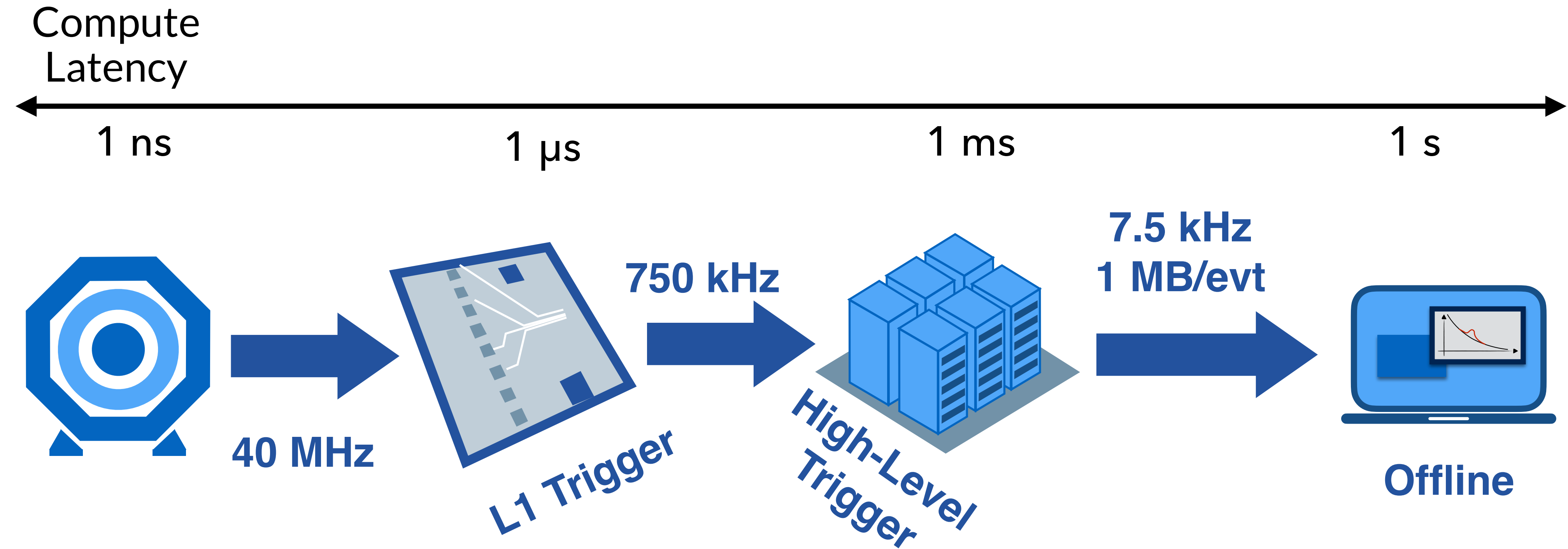


Image: [Microsoft](#)

HL-LHC Data Processing



ASICs

FPGAs

CPUs

GPUs

FPGAs

Other processors:
IPU, TPU ..

CPUs

GPUs

FPGAs

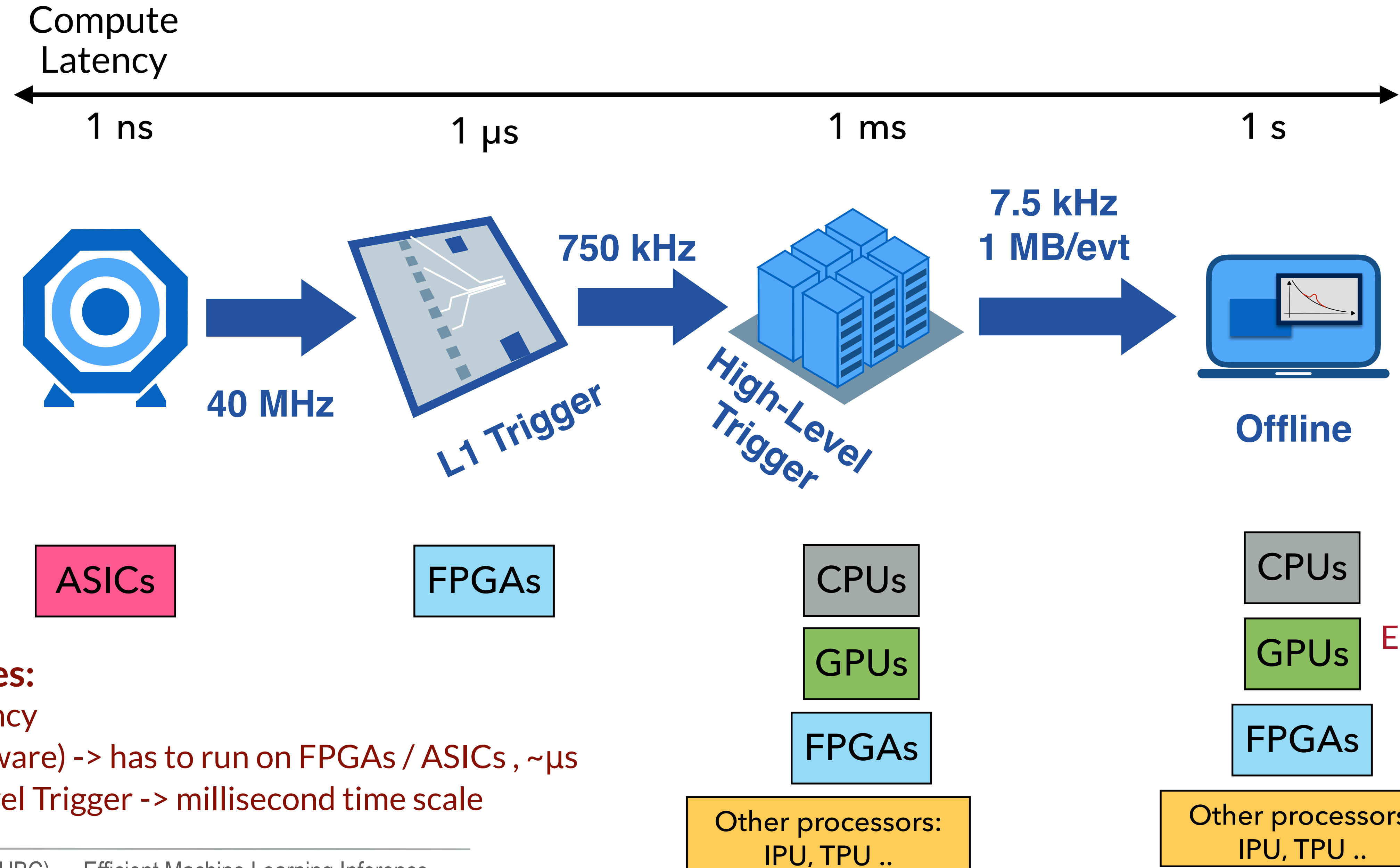
Other processors:
IPU, TPU ..

Exabyte-scale datasets

Challenges:

- Each collision produces $\mathcal{O}(10^3)$ particles
- The detectors have $\mathcal{O}(10^8)$ sensors
- Extreme data rates of $\mathcal{O}(100 \text{ TB/s})$

ML in the Trigger



Challenges:

- Low latency
- L1 (Hardware) -> has to run on FPGAs / ASICs, $\sim \mu$ s
- High-level Trigger -> millisecond time scale

Question #2

What features / properties do you expect in an “efficient” ML model?

What comes to your mind when you think about efficient ML

Think about it for 30 seconds and share your answer

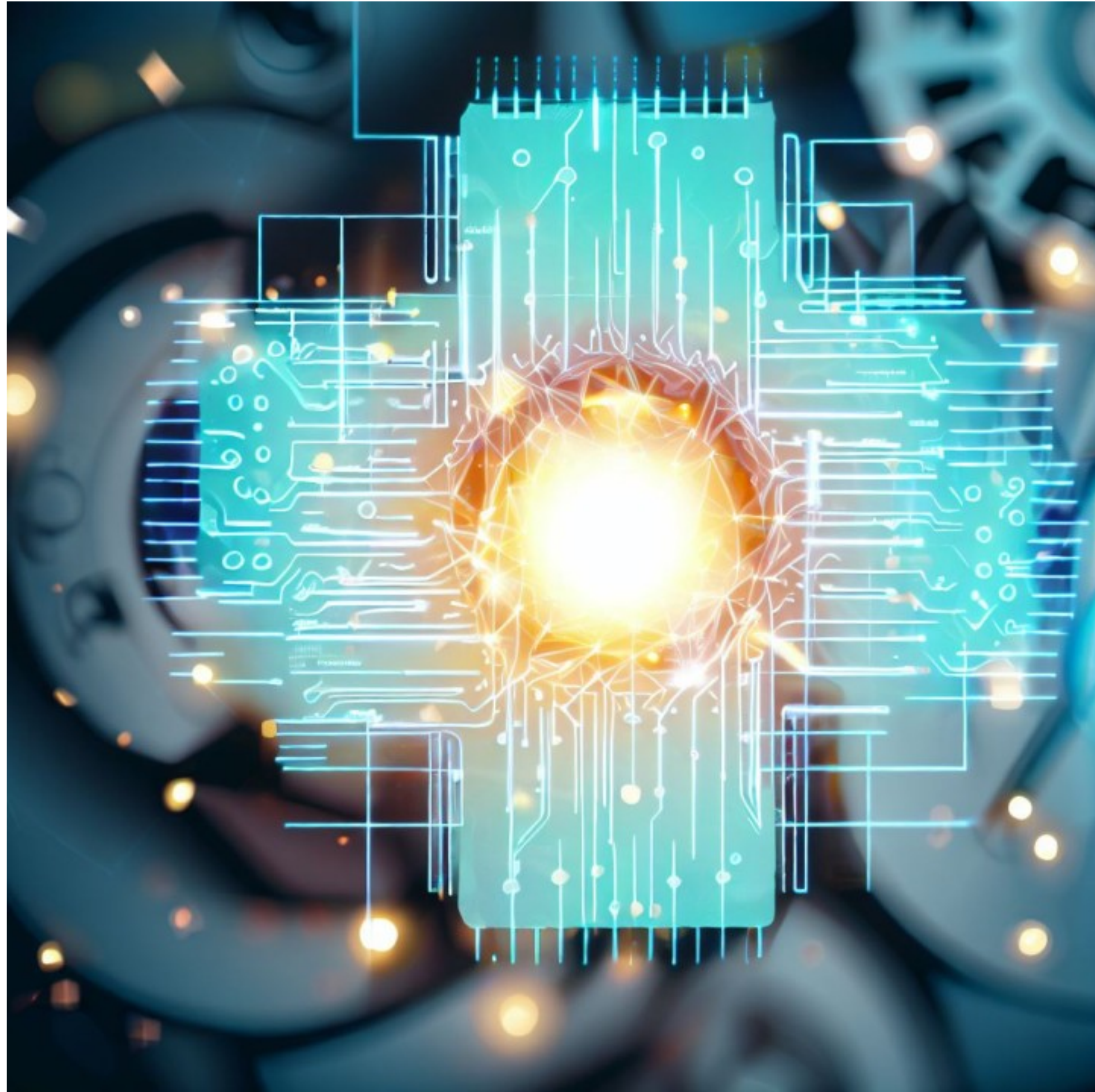
Question #2

What features / properties do you expect in an “efficient” ML model?

What comes to your mind when you think about efficient ML

Think about it for 30 seconds and share your answer

- Smaller in size
- Requires less computing resources for training and / or inference
- Runs faster during inference (prediction stage)
- Uses less power
- Inference should scale well with increasing data volume



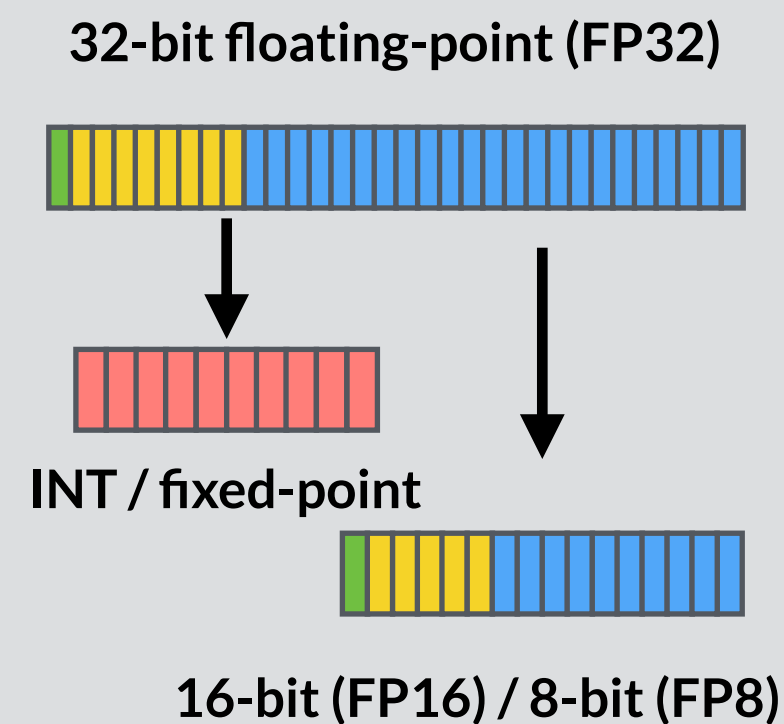
Lets discuss

*some of the techniques to design
Efficient Deep Learning models*

Efficient Model Design

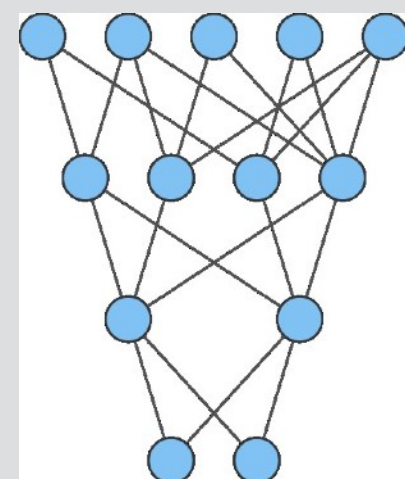
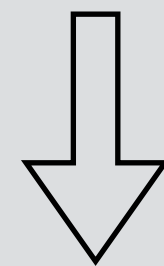
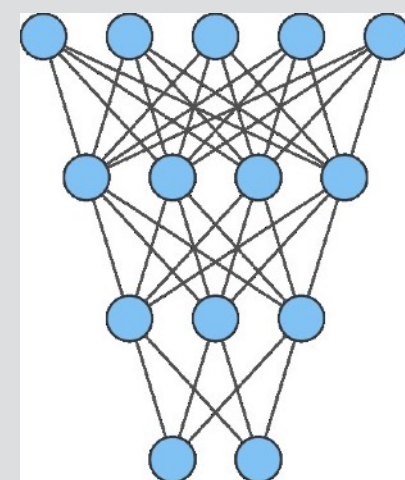
Quantization

Finding the optimal model architecture



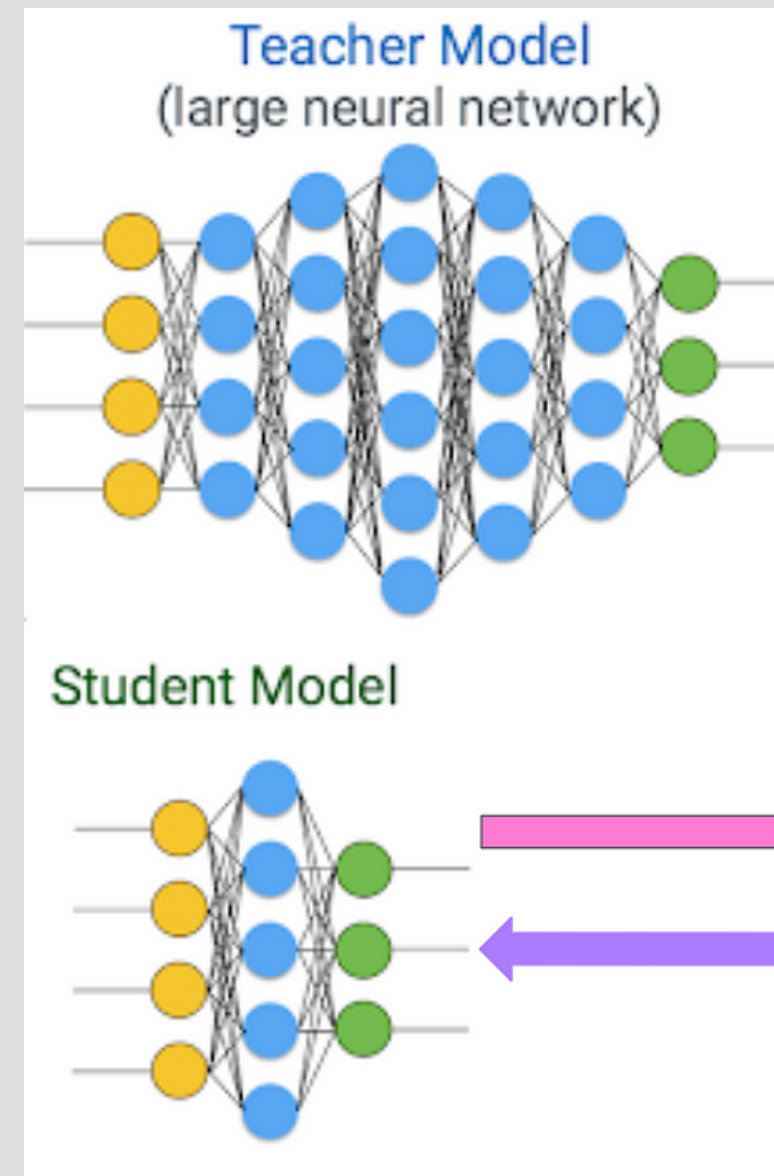
Pruning

Remove synapses and neurons



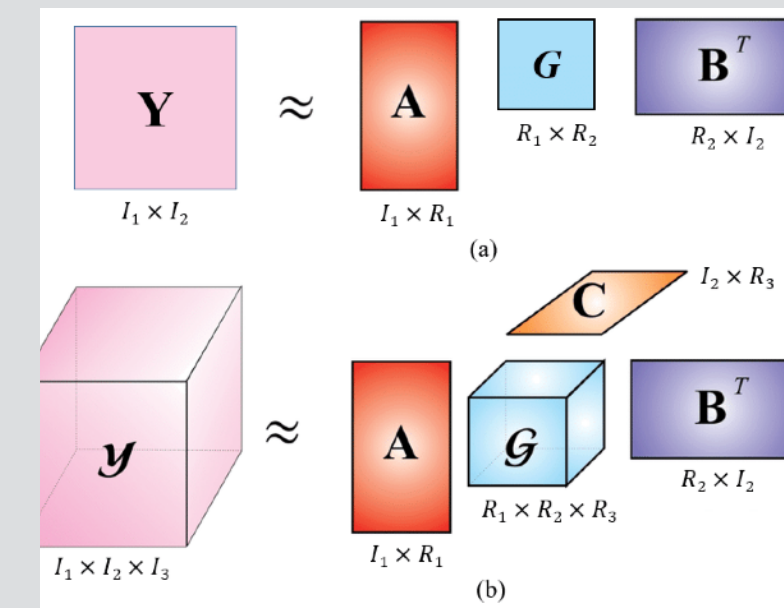
Knowledge Distillation

Train a smaller model using a bigger model



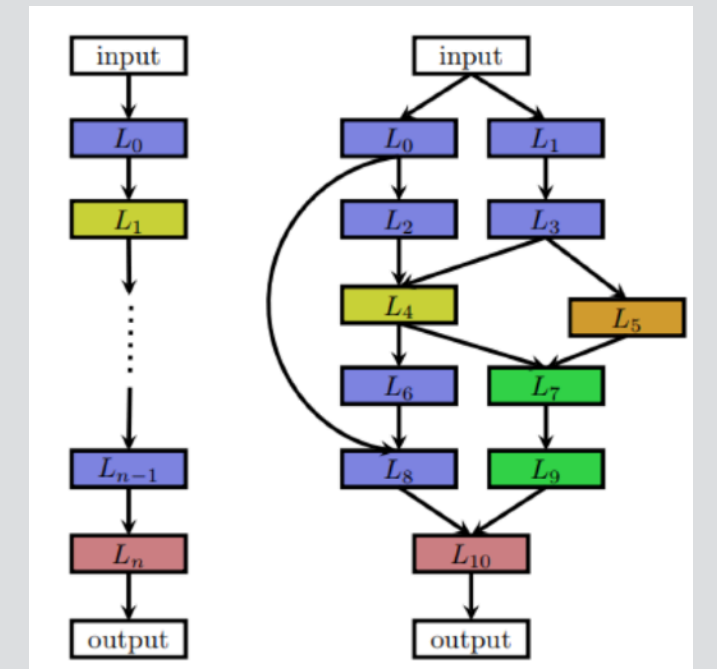
Tensor Decomposition

Reduce the dimension of the weight matrix / tensor



Neural Architecture Search

Finding the optimal model architecture



Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

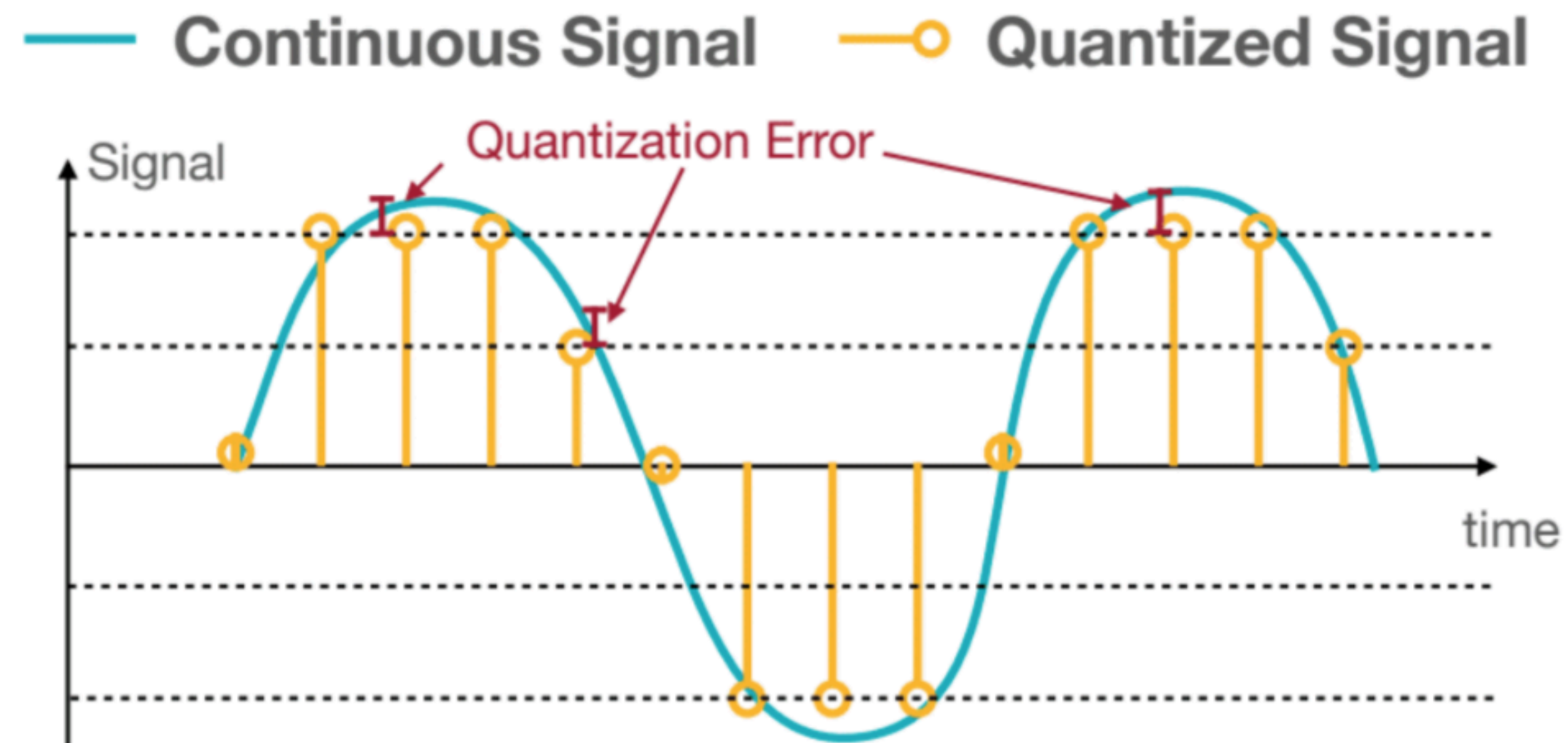
- What is pruning? How to formulate pruning?
- Determine pruning granularity and criteria
- Network performance after pruning

ML on FPGA

hls4ml and Trigger applications

Quantization

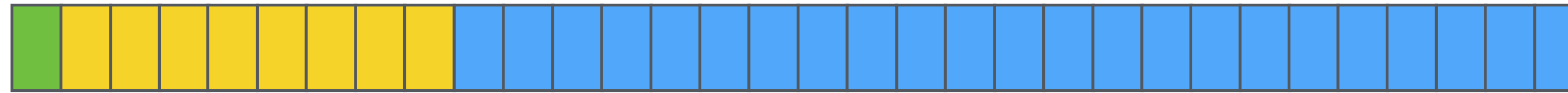
Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set



Numerical Data Types

32-bit floating-point (FP32)

Range: $10^{-38} - 10^{38}$



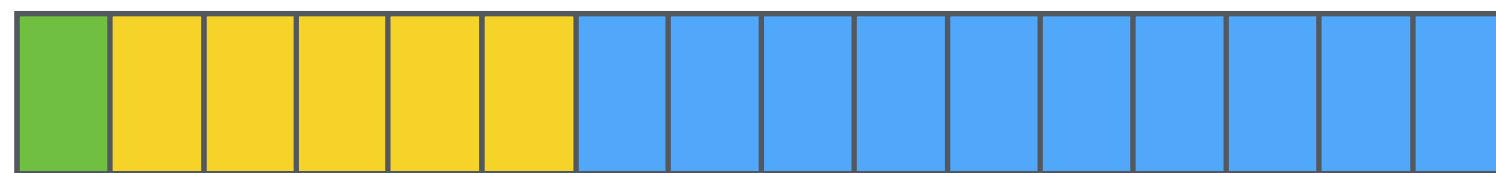
Sign bit

8 bit Exponent

23 bit Mantissa

16-bit floating-point (FP16)

Range: $6 \times 10^{-5} - 6 \times 10^4$



Sign bit

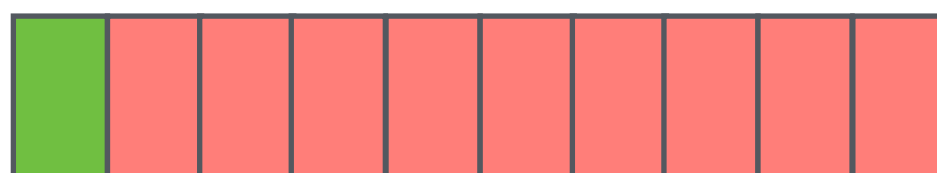
5 bit Exponent

10 bit Mantissa

Unsigned Integer



Signed Integer



Sign bit

Fixed-point Number



Integer . Fraction

"Decimal" Point



$$\begin{matrix} \times & \times & \times & \times & \times & \times & \times & \times \\ -2^3 & +2^2 & +2^1 & +2^0 & +2^{-1} & +2^{-2} & +2^{-3} & +2^{-4} \end{matrix} = 3.0625$$



$$(-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \times 2^{-4} = 49 \times 0.0625 = 3.0625$$

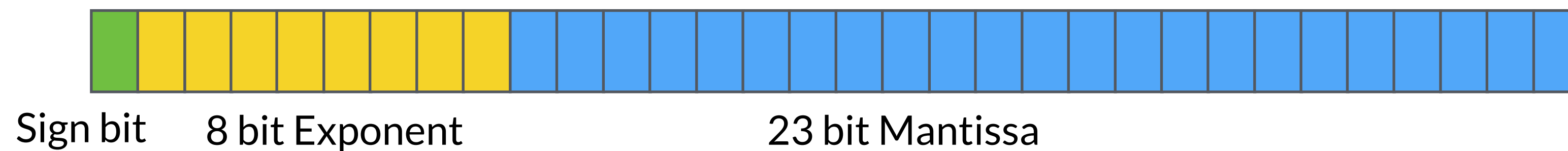
(using 2's complement representation)

GPT-3 Memory

GPT-3
↓
~175,000,000,000
●

32-bit floating-point (FP32)

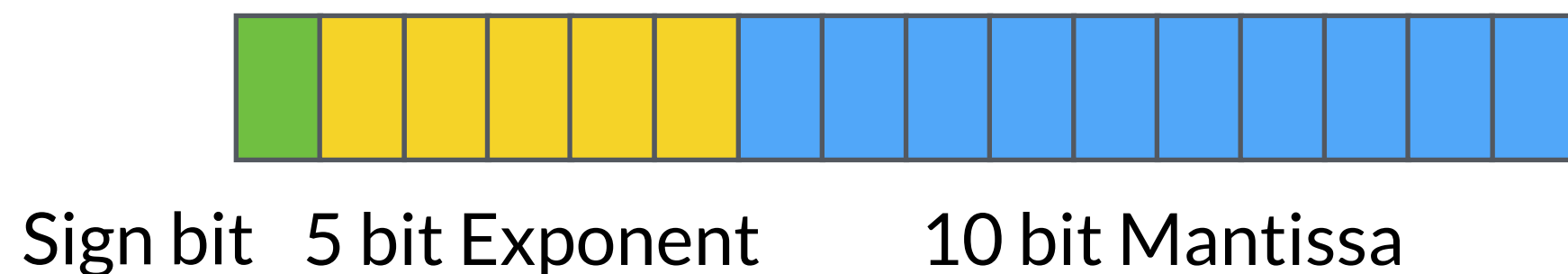
Range: 10^{-38} - 10^{38}



~700 GB of memory
(175 B par x 4 bytes/par)
~10x larger than max memory in a single Nvidia A100 GPU

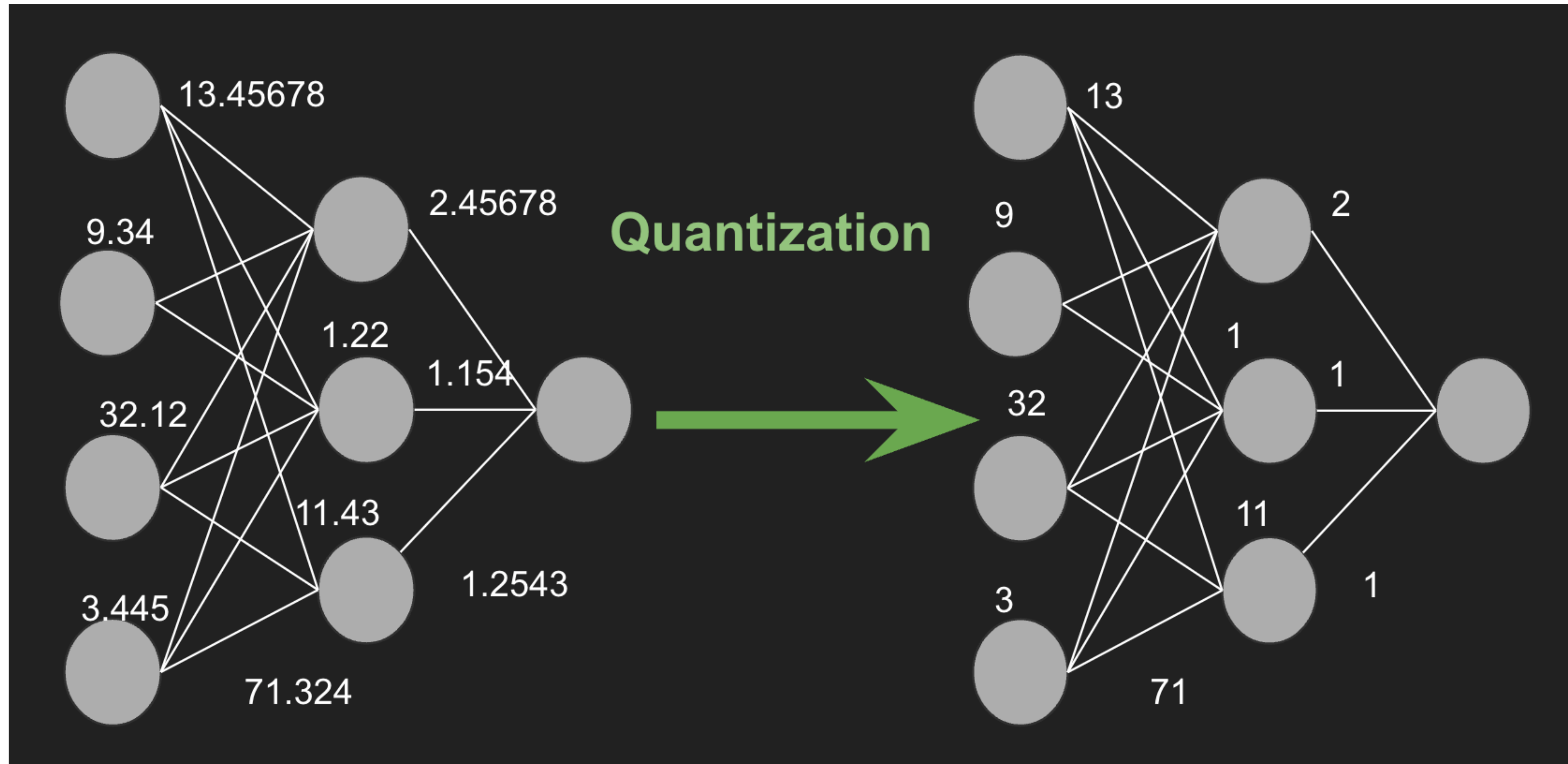
16-bit floating-point (FP16)

Range: 6×10^{-5} - 6×10^4



~350 GB of memory
(175 B par x 2 bytes/par)
~ 5 Nvidia A100 GPUs
~ 11 Nvidia V100 GPUs

Weight Quantization



Quantization Types

Quantization: using reduced precision for parameters and operations

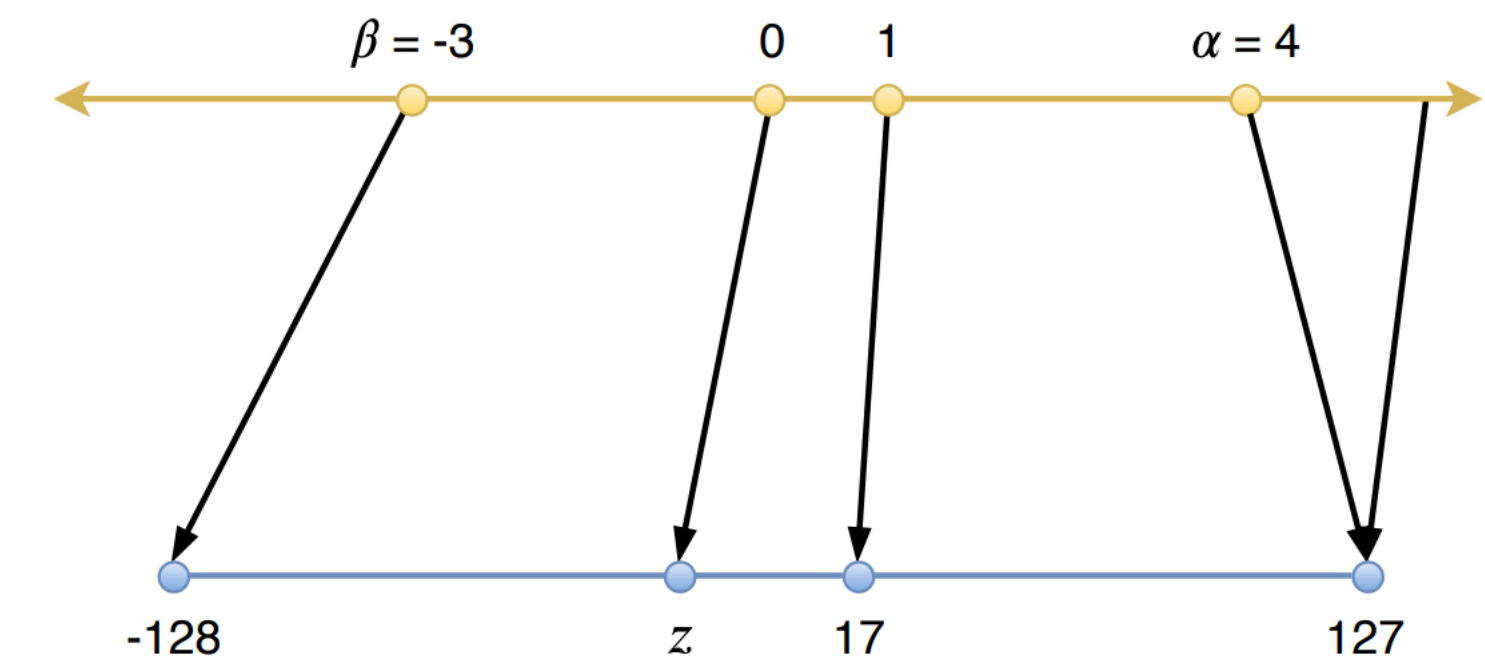
Fixed-point precision

ap_fixed<width bits, integer bits>

0101.1011101010



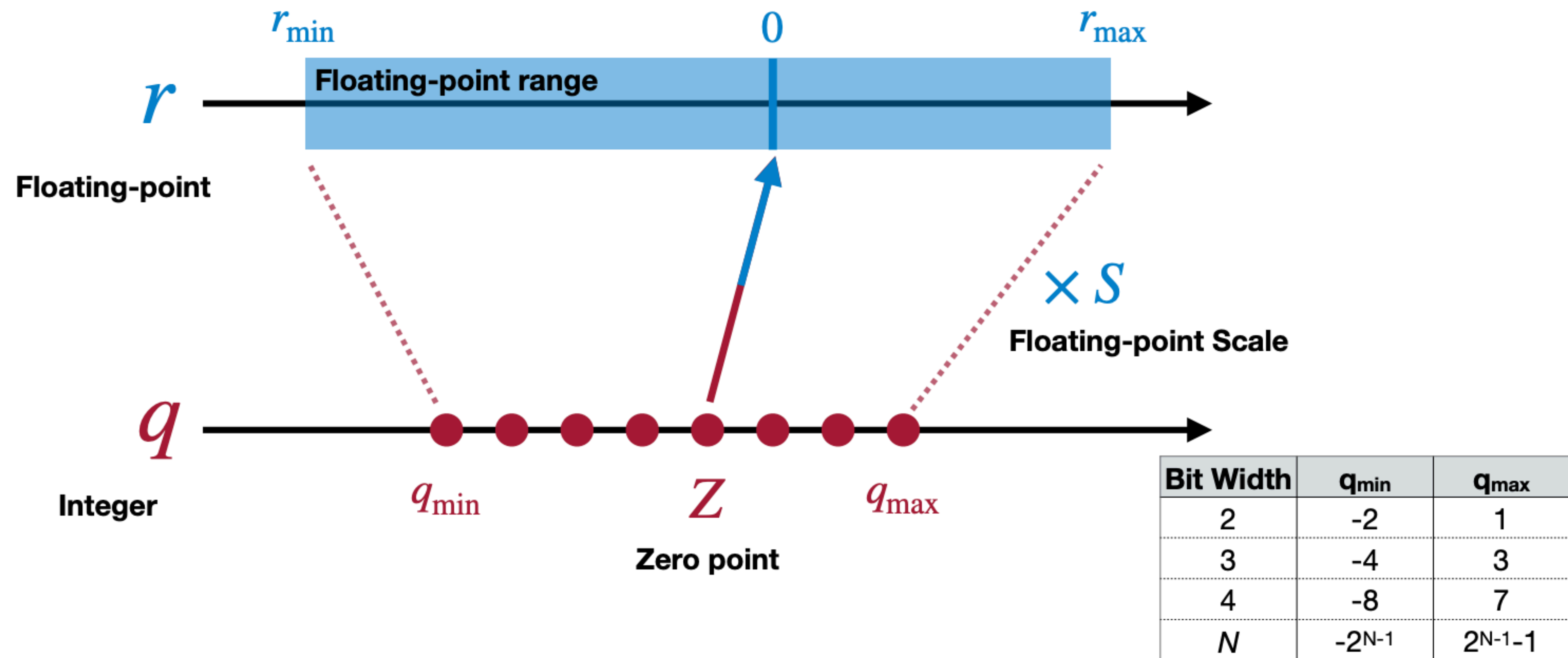
Affine Integer Quantization



[arXiv:2004.09602](https://arxiv.org/abs/2004.09602)

Affine Integer Quantization

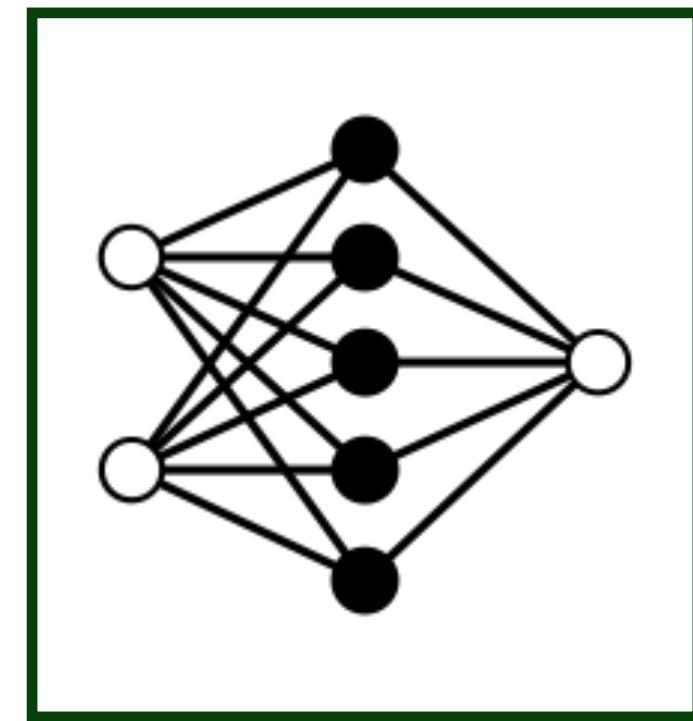
An affine mapping of integers to real numbers $r = S(q - Z)$



Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Quantization Strategies

Post-training Quantization



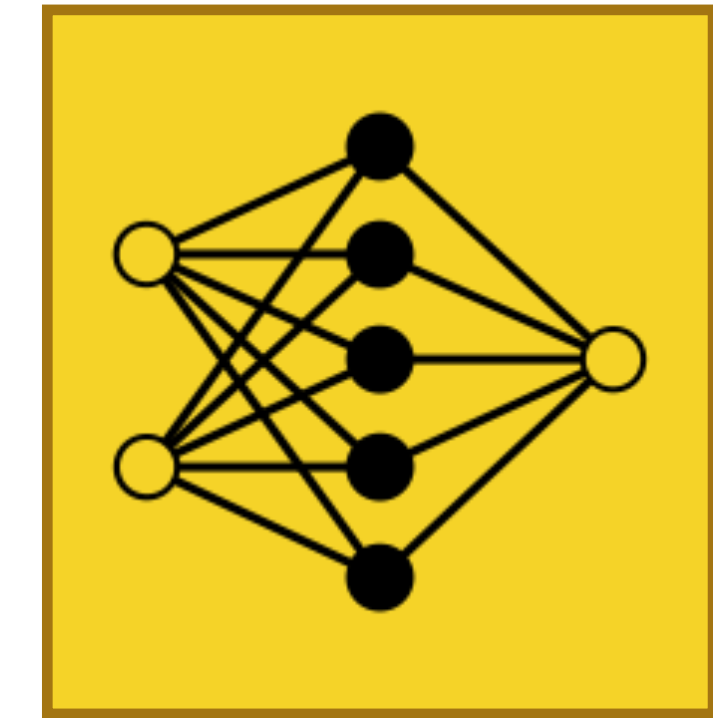
trained model



Quantize Weights
/ Activations

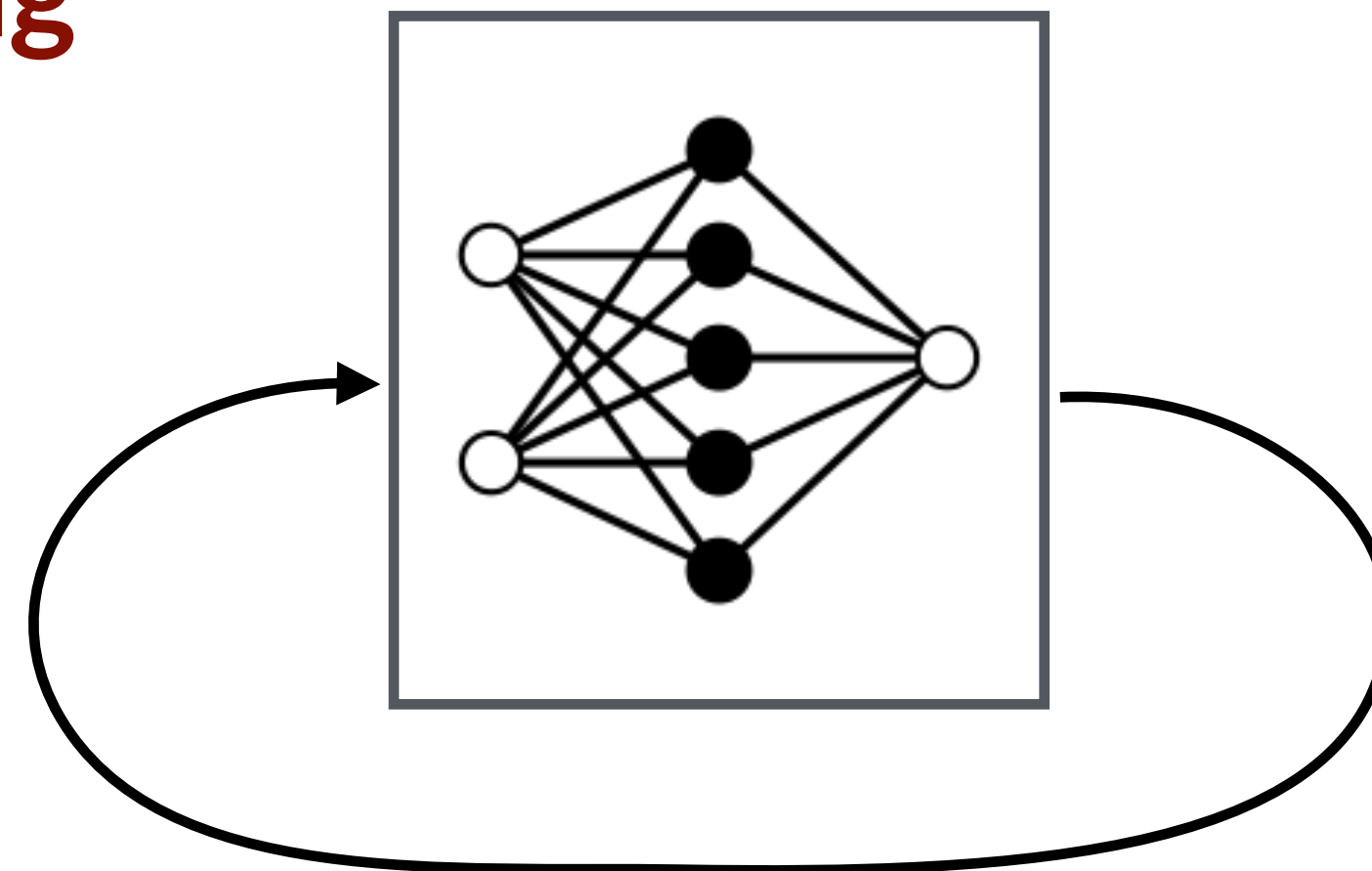


Quantized model

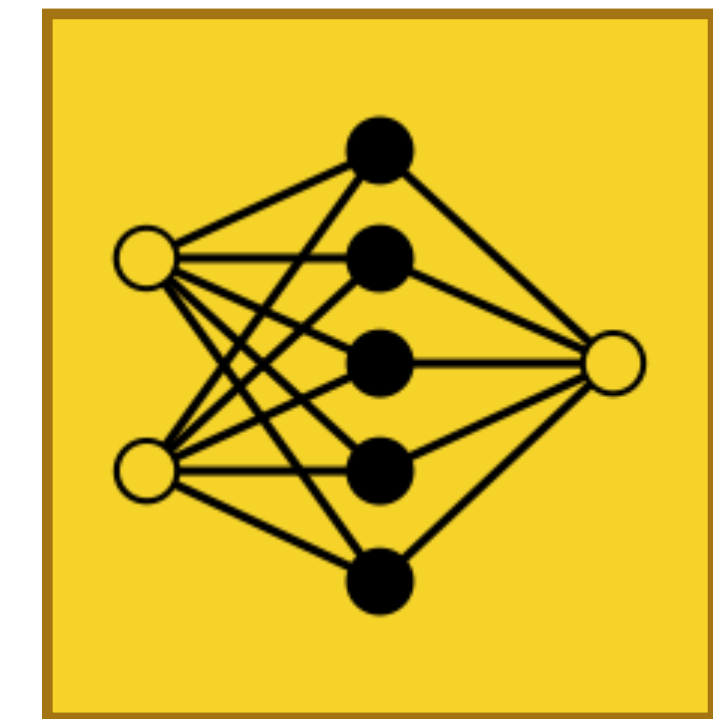


Quantization-aware training

Training loop



Quantized model



Post-training Quantization

- QKeras
- Bravitas (PyTorch Models)
- PyTorch (limited options)
- TensorFlow (limited options)
- ONNX (in development)

Quantization-aware training

- QKeras
- PyTorch (limited options)
- TensorFlow (limited options)
- QONNX (in development)

- PyTorch and TensorFlow gives very limited options for quantization

32 bit float → 16 bit / 8 bit / 4 bit
(INT or FLOAT)

- PyTorch is actively developing TorchAO

- Often HEP applications need more granular optimization. Example: 5 bits

Question #3

Post-training Quantization (PTQ) vs Quantization-Aware Training (QAT)

Advantages and disadvantages of PTQ?

Post-training Quantization (PTQ) vs Quantization-Aware Training (QAT)

PTQ	QAT
Usually fast	Slow
No re-training of the model	Model needs to be trained / finetuned
Plug and play of quantization schemes	Plug and play of quantization schemes (requires re-training)
Less control over final accuracy of the model	More control over final accuracy since <i>q-params</i> are learned during training.

Quantization References

Quantization Survey paper

<https://arxiv.org/pdf/2103.13630>

MIT Efficient ML Course:

<https://hanlab.mit.edu/courses/2023-fall-65940>

TensorFlow model_optimization:

https://www.tensorflow.org/lite/performance/model_optimization

PyTorch Quantization:

<https://pytorch.org/docs/stable/quantization.html>

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

- **What is pruning? How to formulate pruning?**
- Determine pruning granularity and criteria
- Network performance after pruning

ML on FPGA

hls4ml and Trigger applications

Neural Network Pruning

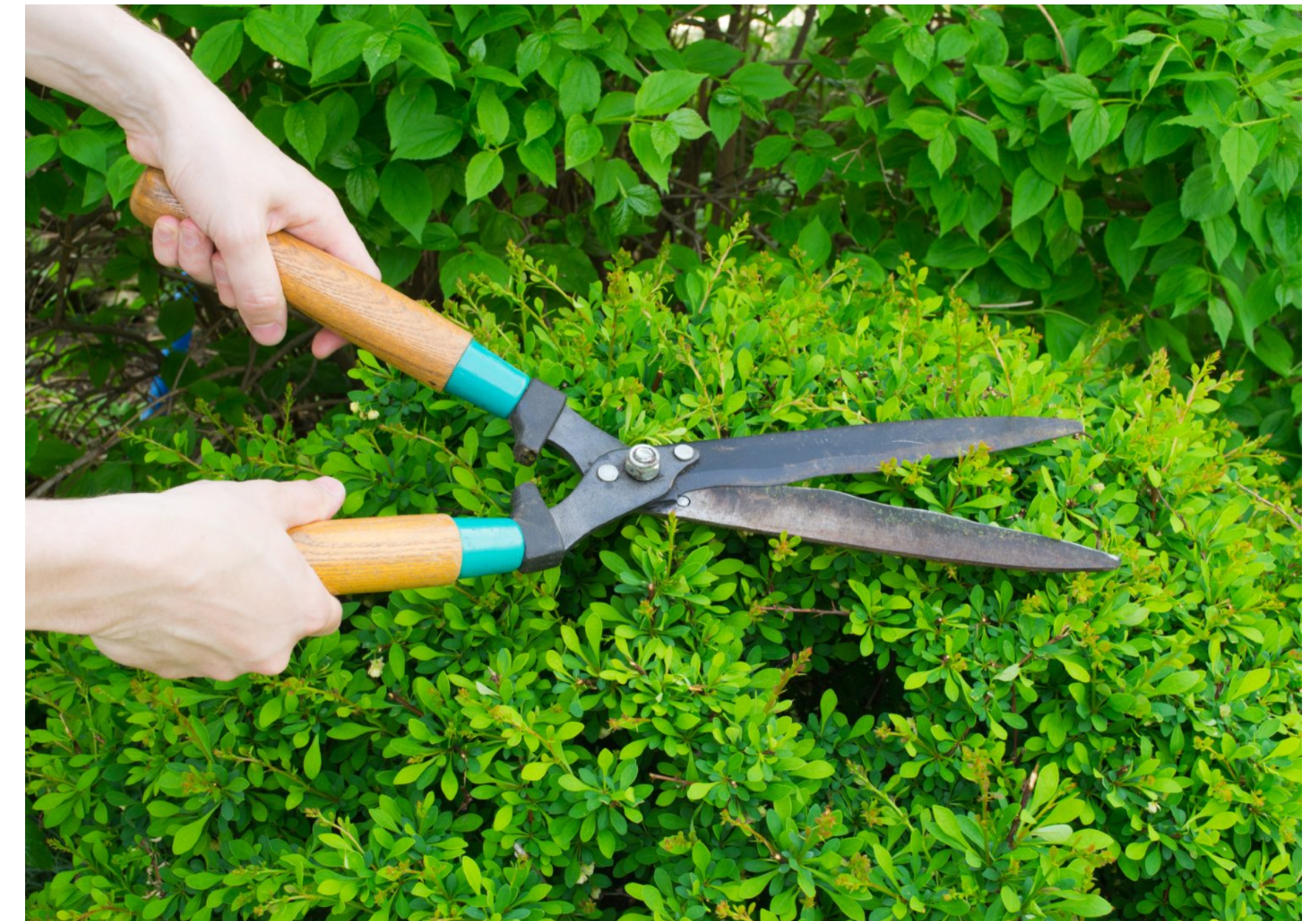
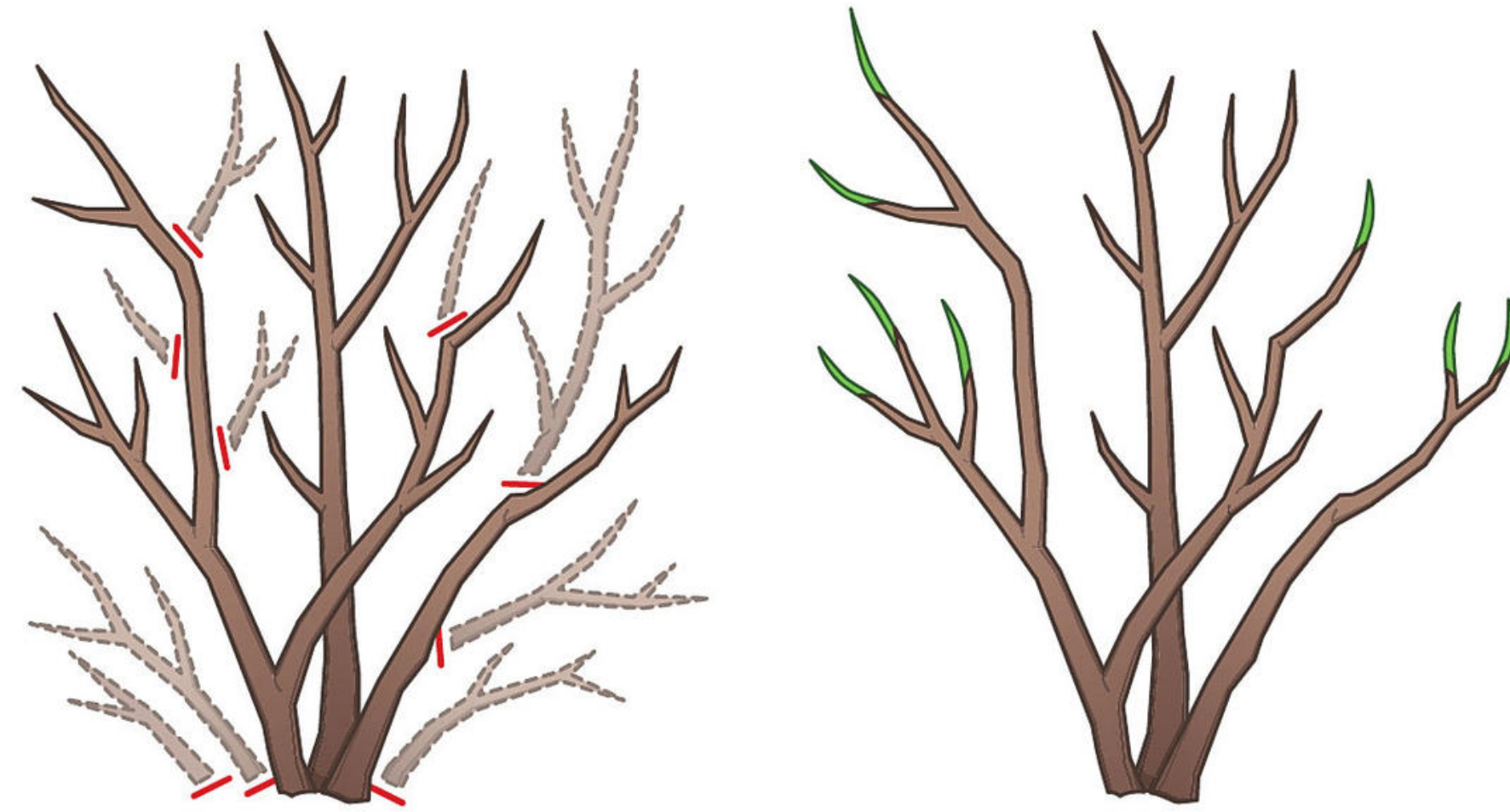
- > 90% reduction in parameter count
- Decreasing the storage requirements
- Improving computation efficiency

Model accuracy loss was negligible

Neural Network	#Parameters		
	Before Pruning	After Pruning	Reduction
AlexNet	61 M	6.7 M	9 ×
VGG-16	138 M	10.3 M	12 ×
GoogleNet	7 M	2.0 M	3.5 ×
ResNet50	26 M	7.47 M	3.4 ×

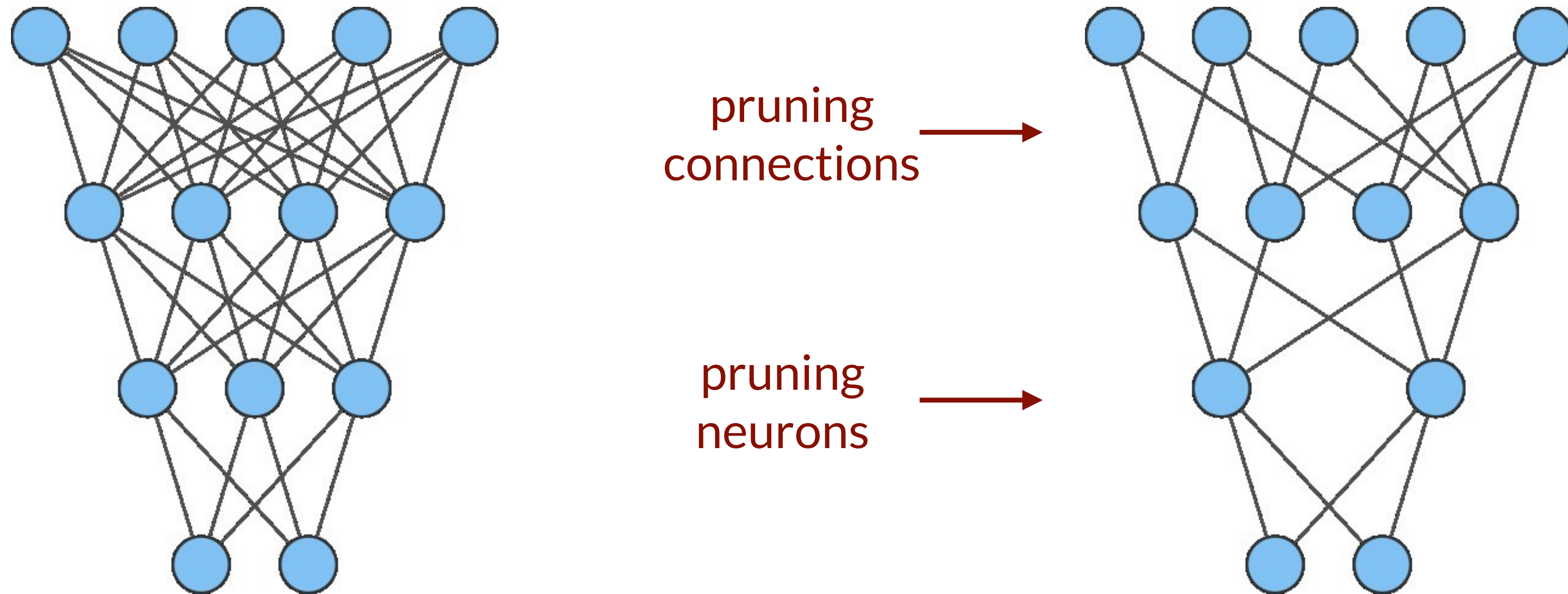
Results are from [Efficient Methods and Hardware for Deep Learning \[Han, S, Stanford University\]](#)

What is pruning?



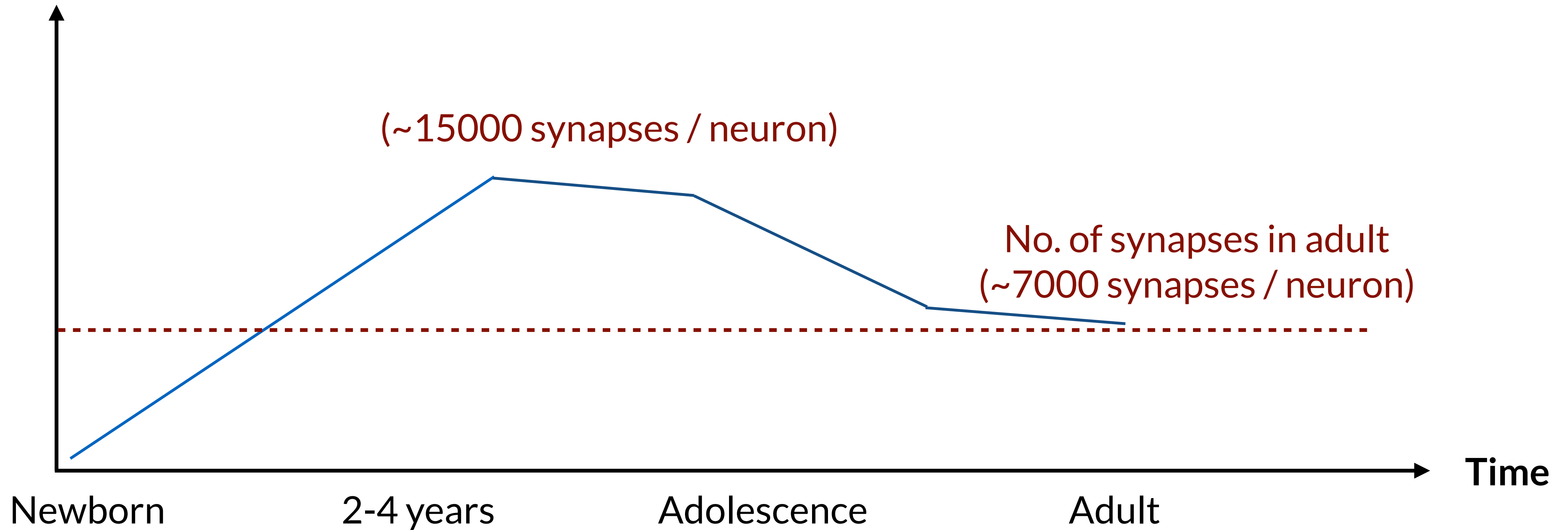
Neural Network Pruning

Pruning is the technique to remove less important connections and neurons

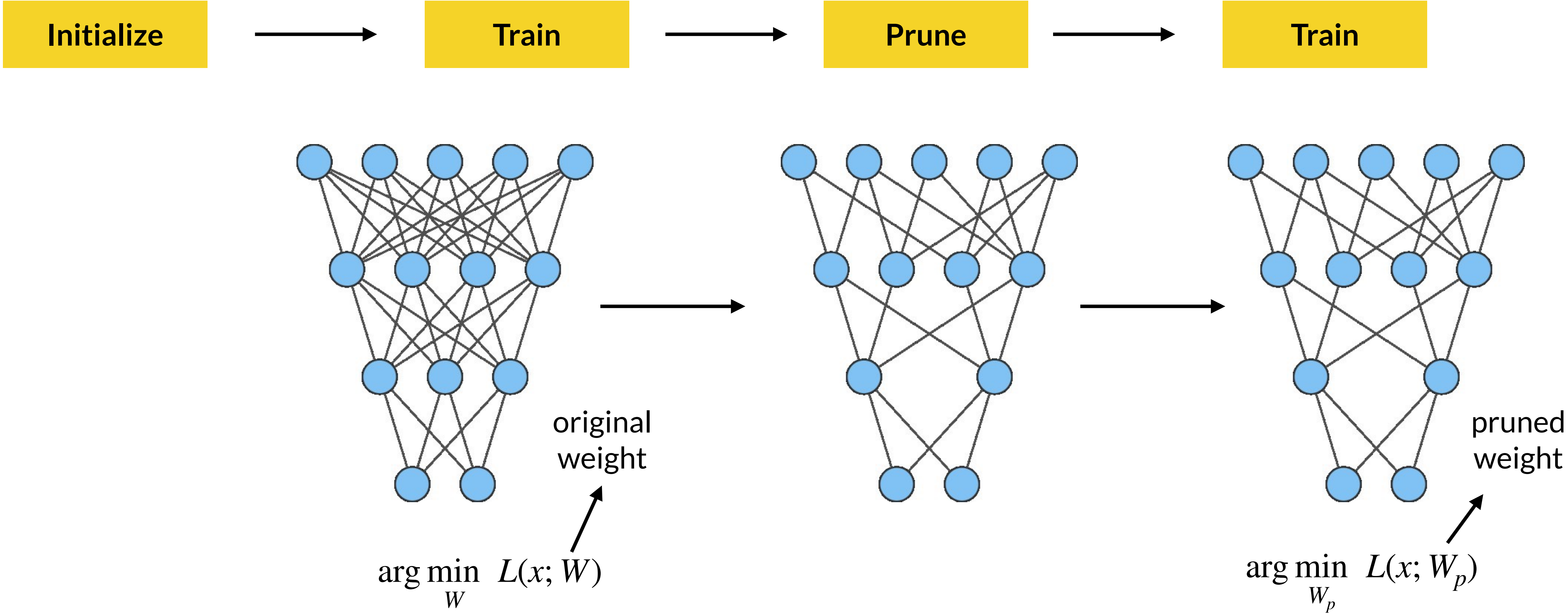


Pruning happens in Human Brain

No. Of Synapses



Pruning workflow



Weight Pruning: Formulation

Remove least important connections such that $\|W_p\|_0 < N$

W_p is the pruned weight matrix

N = target #non-zero weights

$$\text{Sparsity} = \frac{\# \text{ of non-zero weights}}{\text{Total \# of weights}} = \frac{\|W_p\|_0}{\|W_p\|}$$

Example

50% sparsity means half of the weight are pruned

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

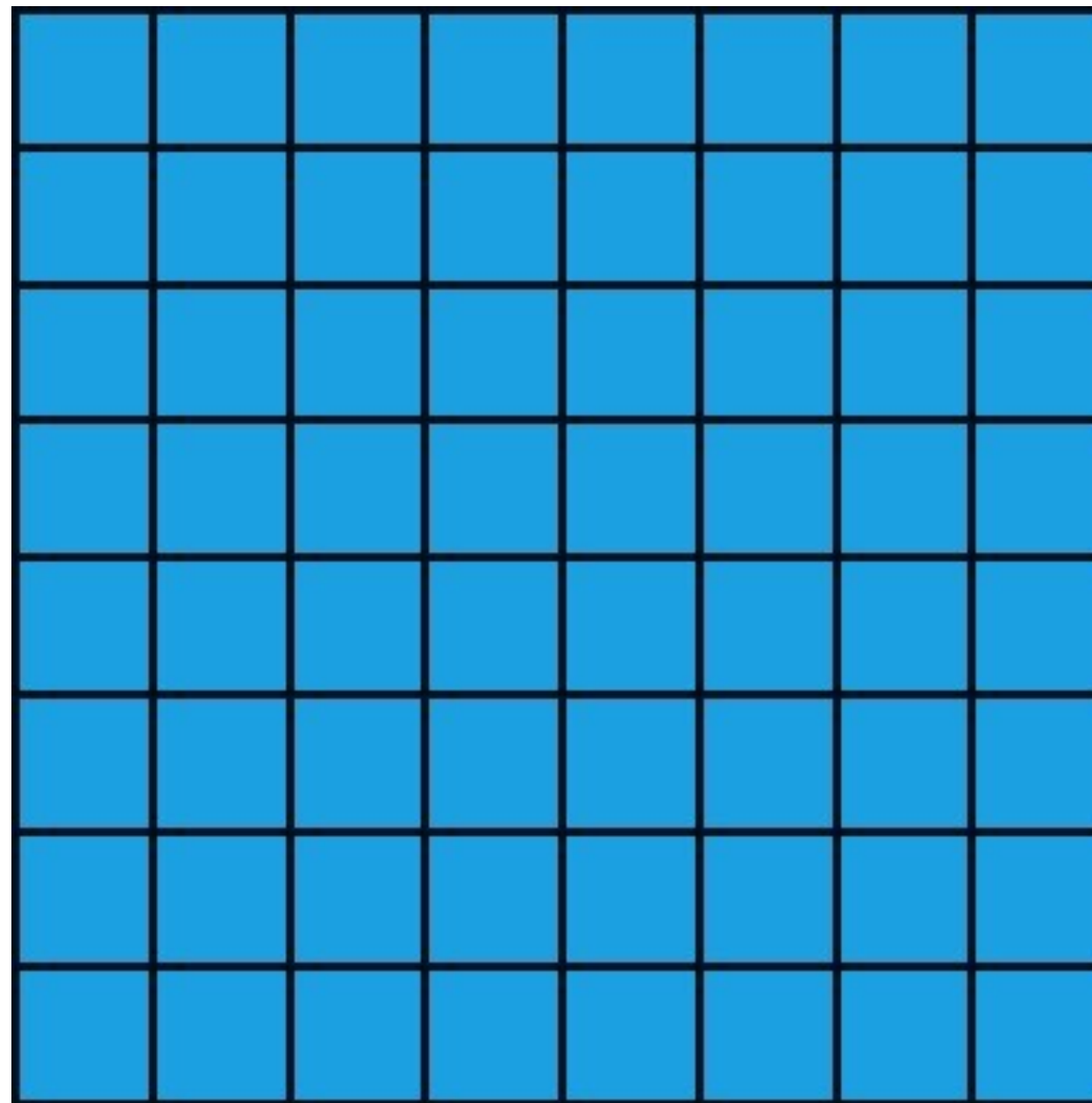
- What is pruning? How to formulate pruning?
- **Determine pruning granularity and criteria**
- Network performance after pruning

ML on FPGA

hls4ml and Trigger applications

Different Pruning Granularity: Structured and Unstructured

Example: 2D weight matrix (8x8)



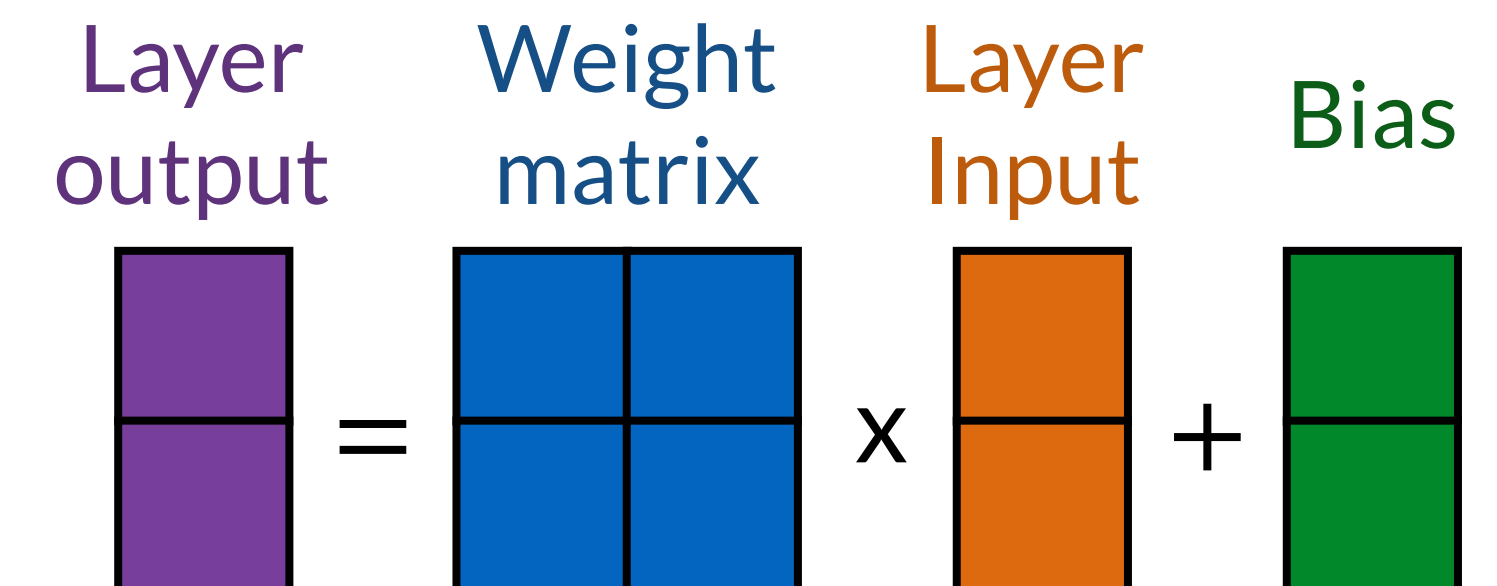
NN layer computation

Layer input = $x_1 \in \mathbb{R}^{m_1}$

Weight = $W_1 \in \mathbb{R}^{m_2 \times m_1}$, Bias = $b_1 \in \mathbb{R}^{m_2}$

Layer output = $x_2 \in \mathbb{R}^{m_2}$

$$x_2 = W_1 \cdot x_1 + b_1$$



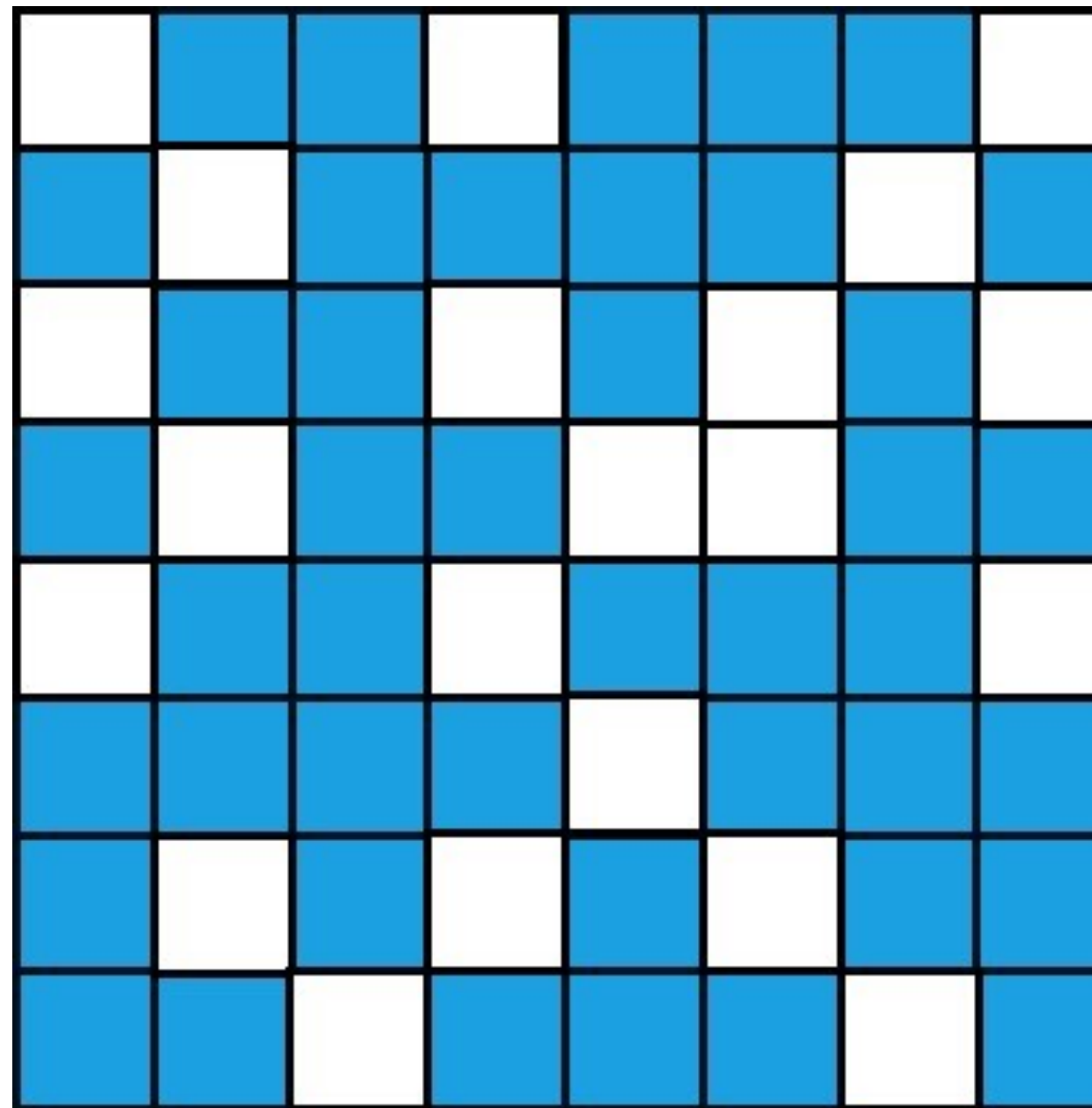
Different Pruning Granularity: Structured and Unstructured

Example: 2D weight matrix (8x8)

□ Pruned
■ Preserved

Fine-grained / Unstructured Pruning

- Flexible pruning index



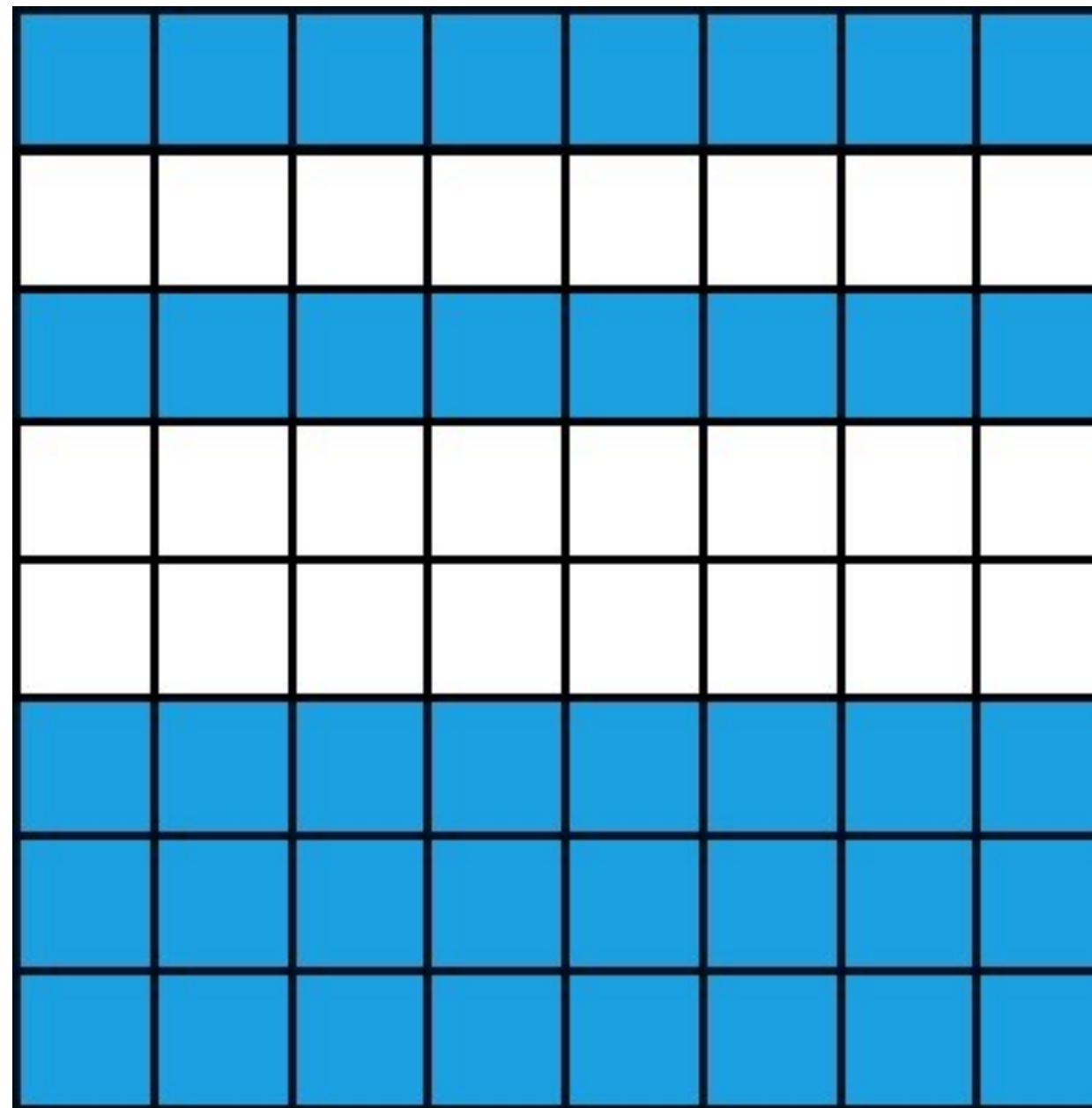
Different Pruning Granularity: Structured and Unstructured

Example: 2D weight matrix (8x8)

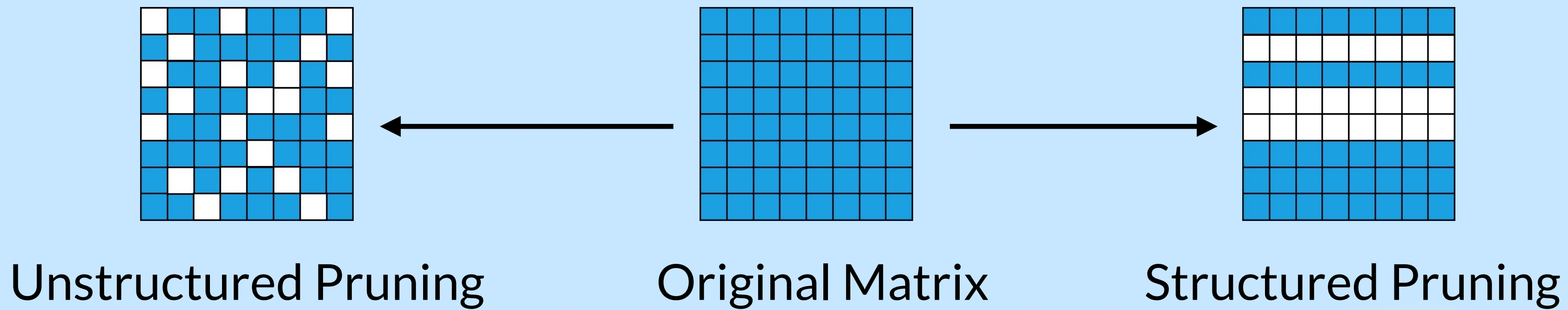
□ Pruned
■ Preserved

Coarse-grained / Structured Pruning

- Less flexible pruning index



Question #4



Which statement is more accurate?

Context: neural network (NN) inference / prediction stage

NN layer computation

Layer input = $x_1 \in \mathbb{R}^{m_1}$

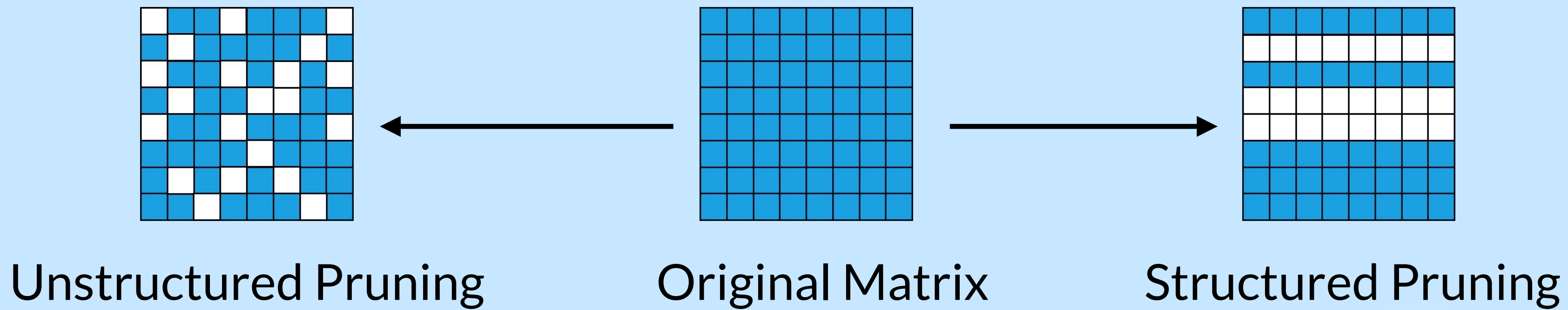
Weight = $W_1 \in \mathbb{R}^{m_2 \times m_1}$, Bias = $b_1 \in \mathbb{R}^{m_2}$

Layer output = $x_2 \in \mathbb{R}^{m_2}$

$$x_2 = W_1 \cdot x_1 + b_1$$

- A. Accelerating NN models is easier after unstructured pruning (weight matrix)
- B. Accelerating NN models is easier after structured pruning (weight matrix)
- C. Both methods offer similar ease of acceleration
- D. The original network (before pruning) will be faster

Question #4



Which statement is more accurate?

Context: neural network (NN) inference / prediction stage

NN layer computation

Layer input = $x_1 \in \mathbb{R}^{m_1}$

Weight = $W_1 \in \mathbb{R}^{m_2 \times m_1}$, Bias = $b_1 \in \mathbb{R}^{m_2}$

Layer output = $x_2 \in \mathbb{R}^{m_2}$

$$x_2 = W_1 \cdot x_1 + b_1$$

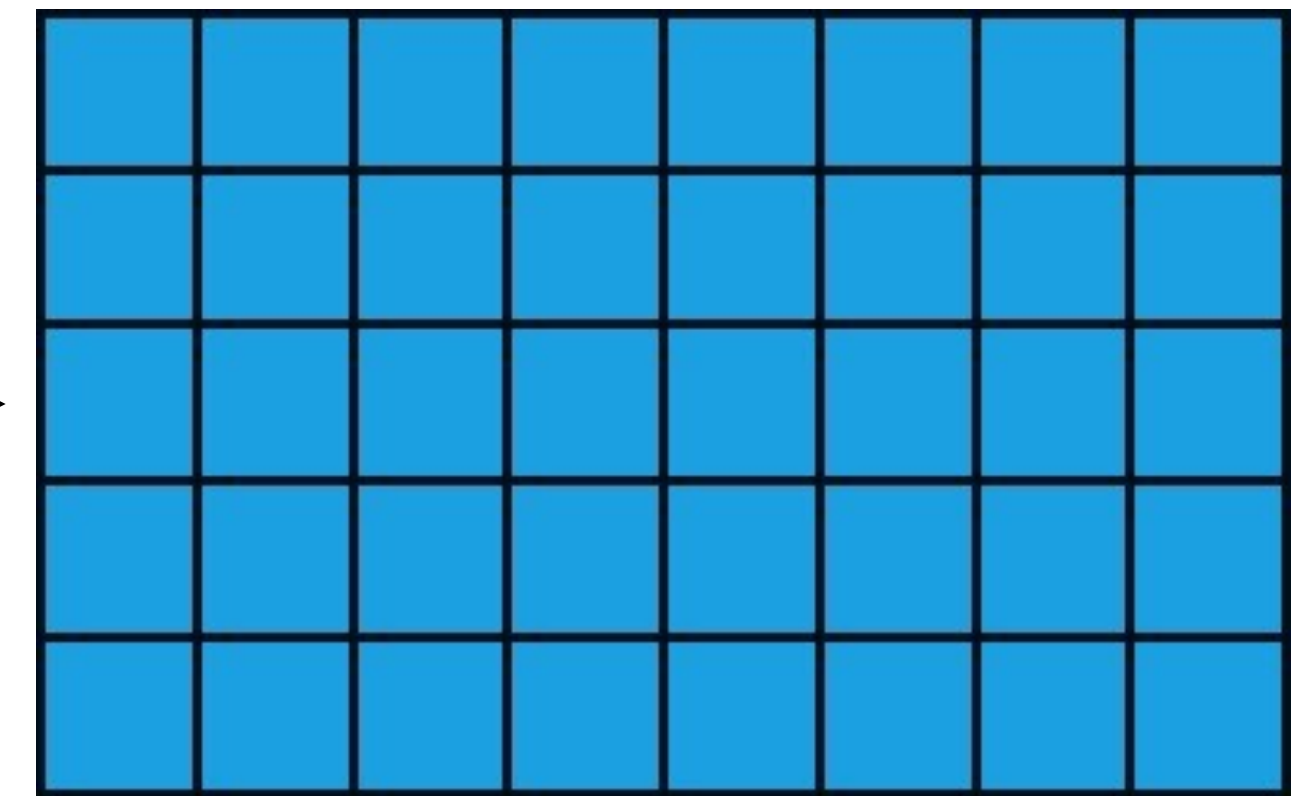
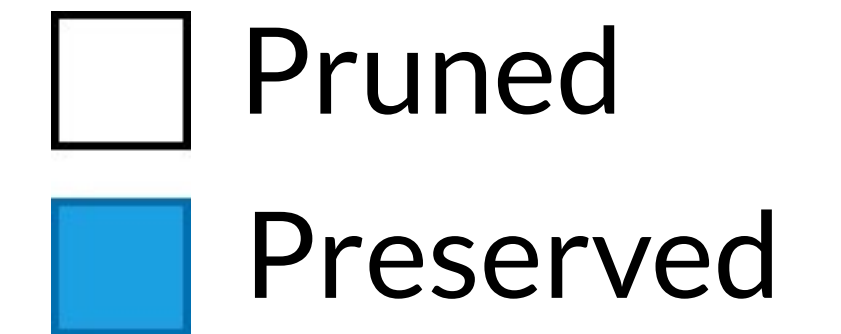
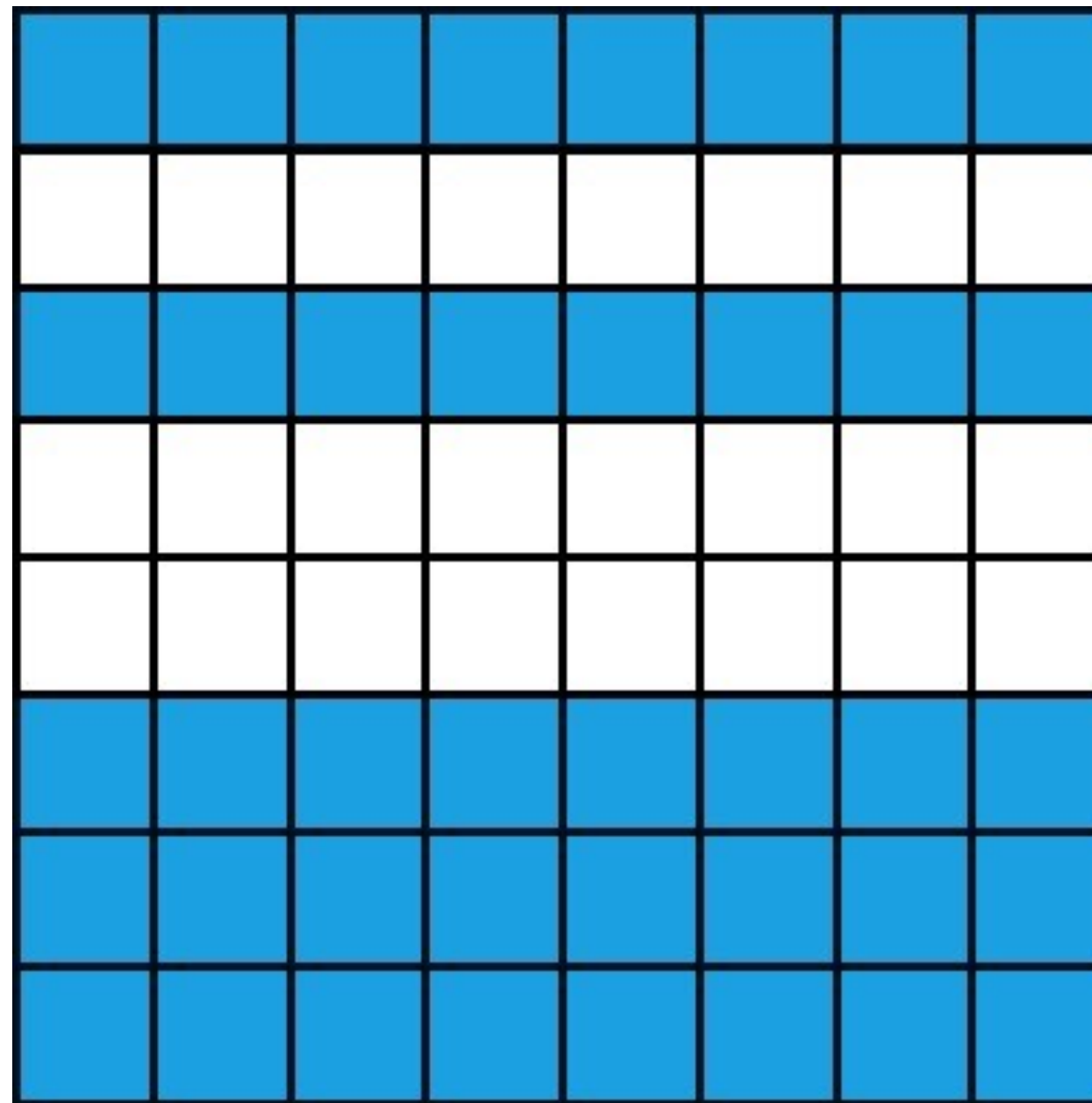
- A. Accelerating NN models is easier after unstructured pruning (weight matrix)
- B. Accelerating NN models is easier after structured pruning (weight matrix)**
- C. Both methods offer similar ease of acceleration
- D. The original network (before pruning) will be faster

Different Pruning Granularity: Structured and Unstructured

Example: 2D weight matrix (8x8)

Coarse-grained / Structured Pruning

- Less flexible pruning index
- Small regular matrix →
easy to accelerate



Mention hardware

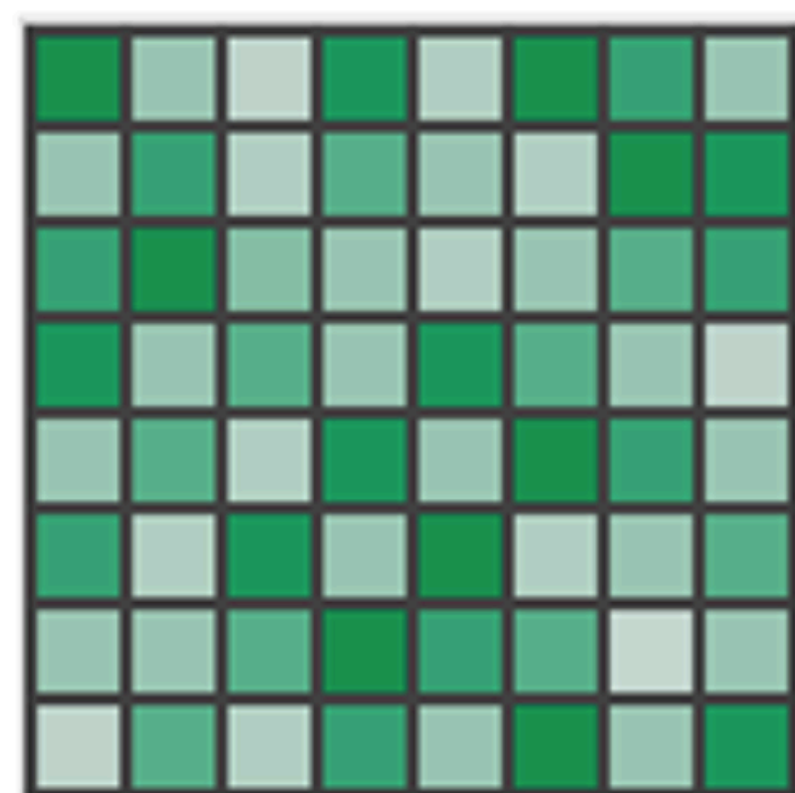
Pattern-based Pruning

Pruning with N:M sparsity = In each M contiguous elements, N of them are pruned

Example: 2:4 sparsity (50% sparsity)

It is supported by Nvidia's Ampere GPU architecture (eg A100)

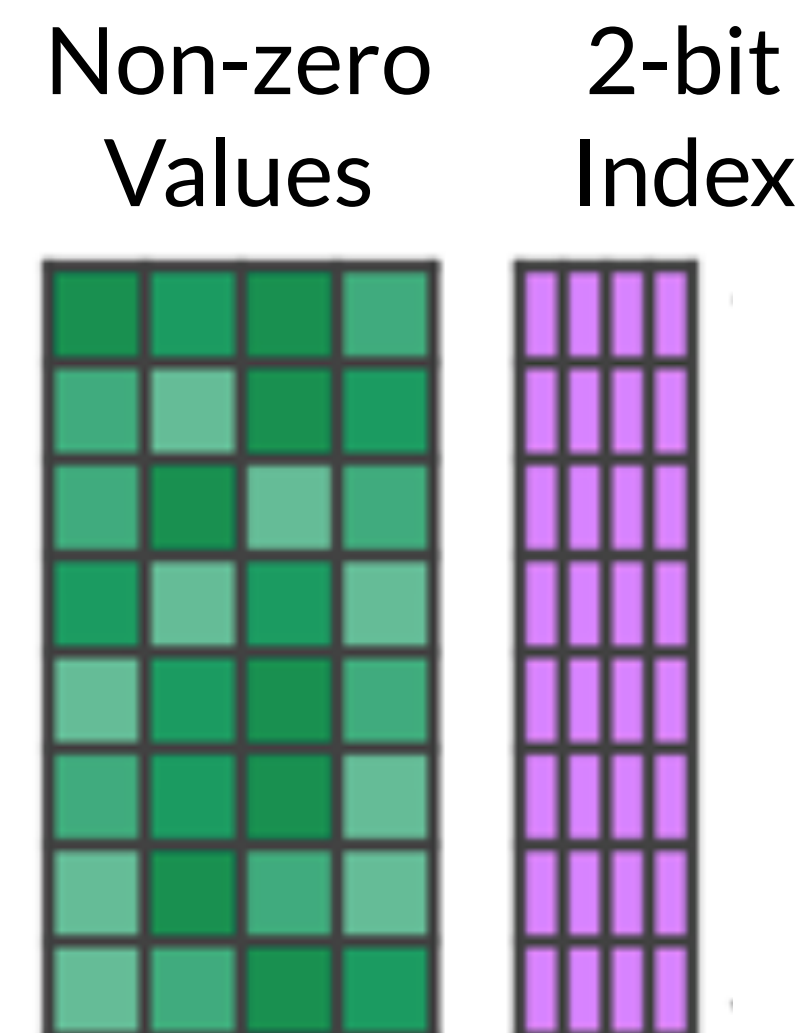
$$\text{Sparsity} = \frac{\text{\# of non-zero weights}}{\text{Total \# of weights}}$$



Dense Matrix



2:4 Sparse Matrix



Compressed Matrix

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

- What is pruning? How to formulate pruning?
- **Determine pruning granularity and criteria**
- Network performance after pruning

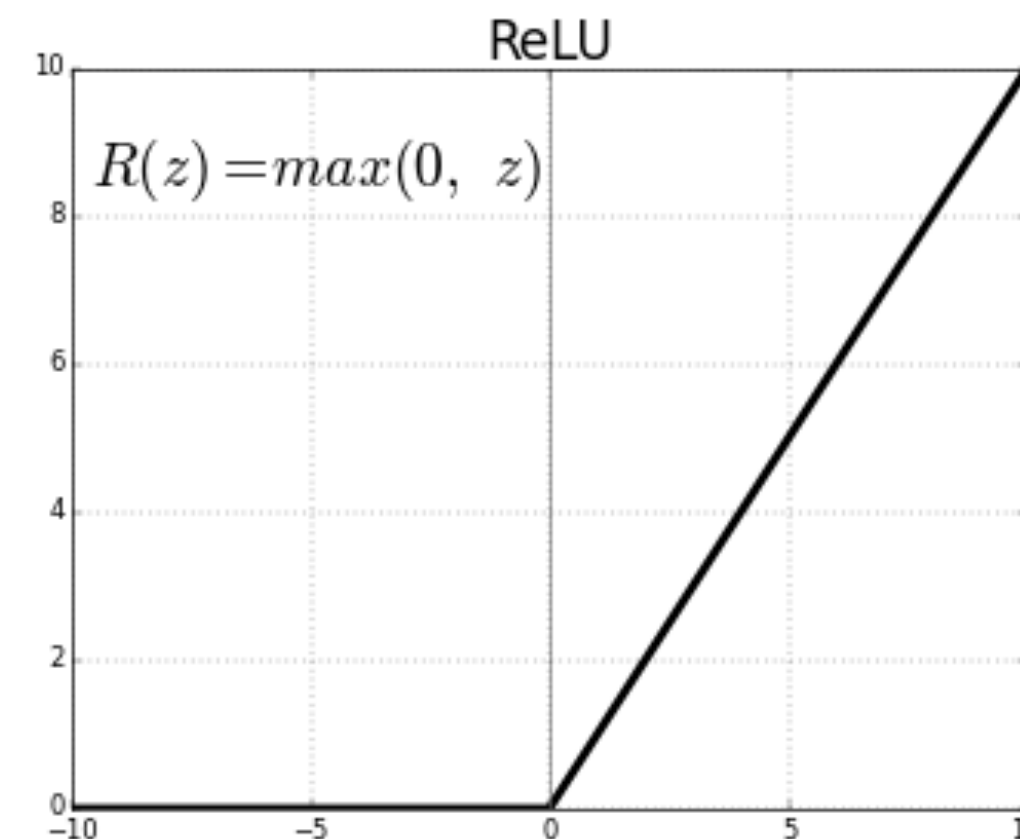
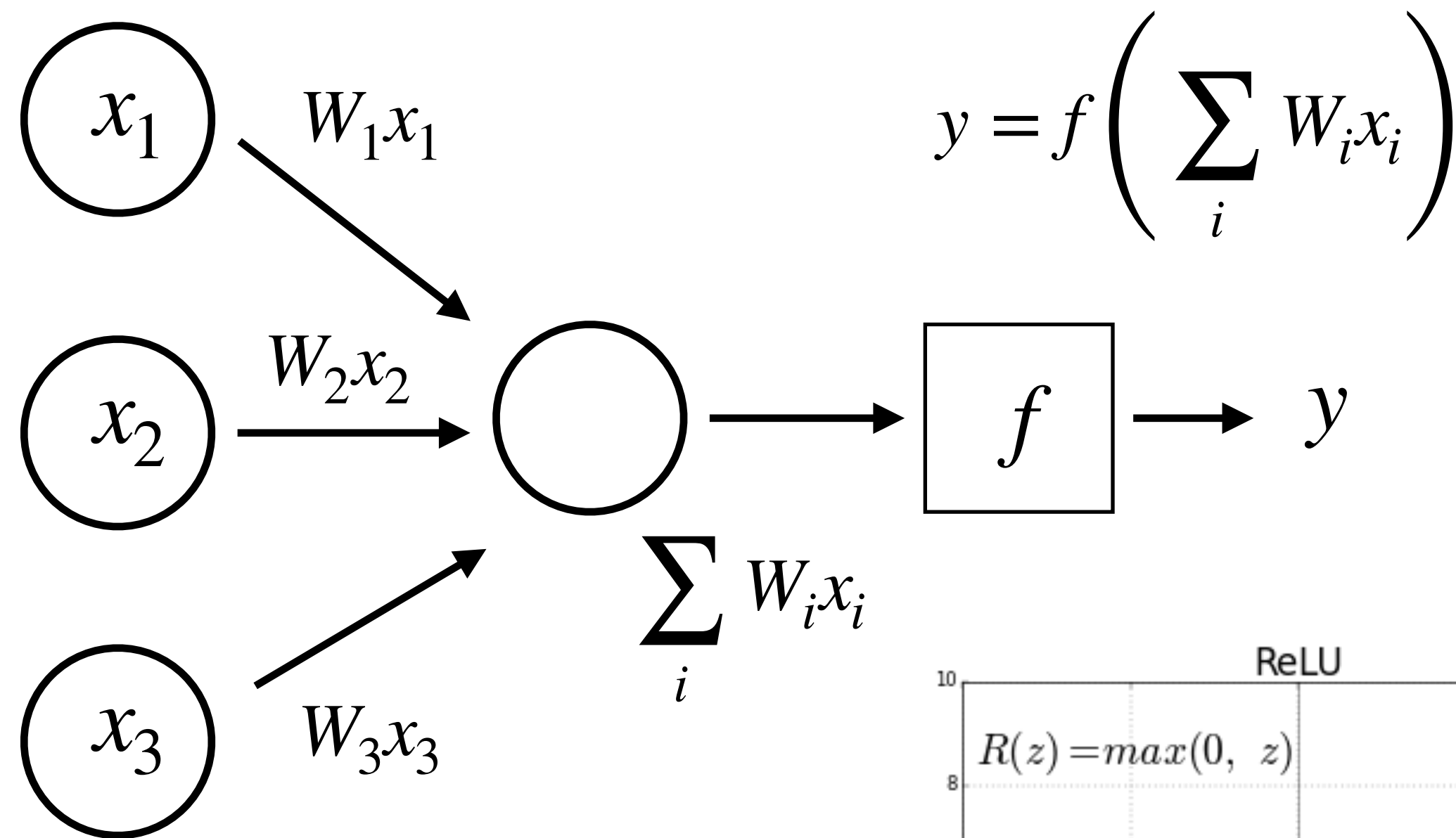
ML on FPGA

hls4ml and Trigger applications

Pruning Criteria

Goal:

Remove less important parameters from a neural network



Example

ReLU Activation

$$f(\cdot) = \text{ReLU}(\cdot)$$

Weights: $W = [-5, 12, 0.2]$

$$y = \text{ReLU}(-5x_1 + 12x_2 + 0.2x_3)$$

If we want to remove one weight, then which one?

Magnitude-based Pruning

Element-wise pruning using absolute magnitude

Magnitude-based pruning considers weights with larger absolute values are more important than other weights.

$$\text{Importance} = |W_i|$$

□ Pruned
■ Preserved

2	-4
7	-1

Original Weight

Element-wise
L1-norm

2	-4
7	-1

Importance

2	4
7	1

0	-4
7	0

Pruned Weight

Magnitude-based Pruning

Row-wise pruning

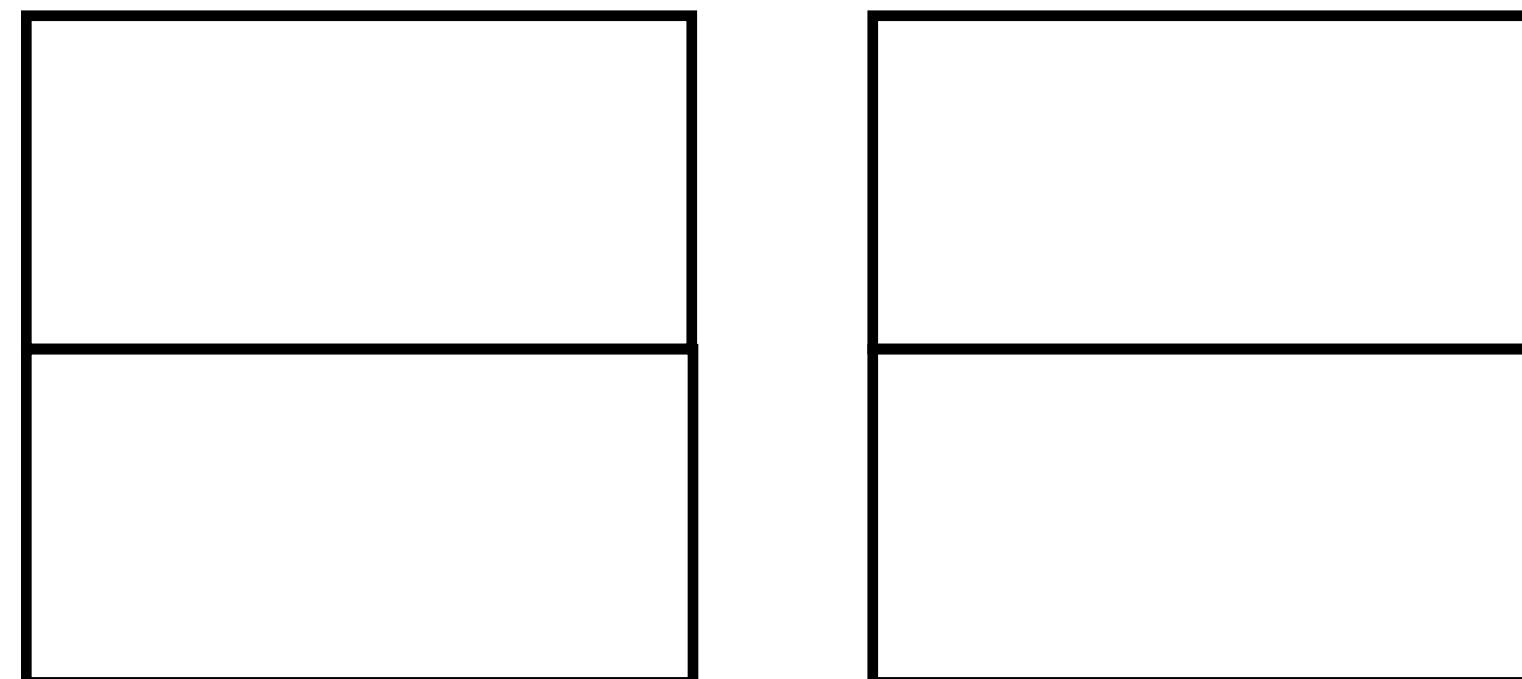
Magnitude-based pruning considers weights with larger absolute values are more important than other weights.

Importance =

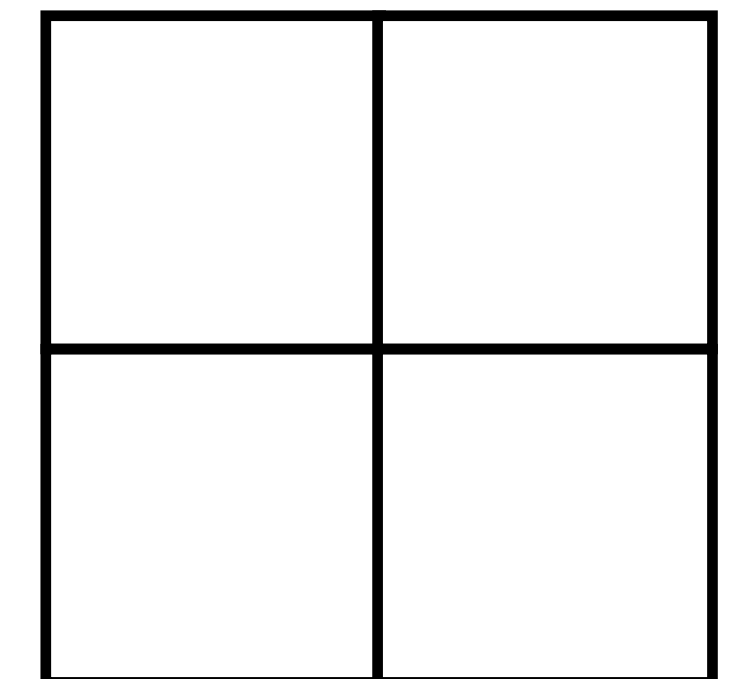
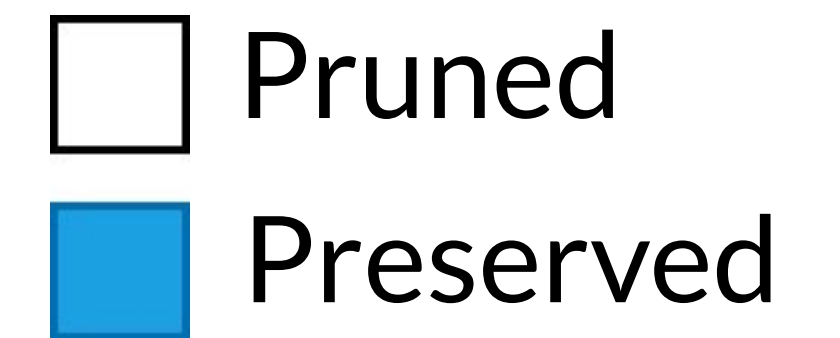
2	-4
7	-1

Original Weight

Row-wise



Importance



Pruned Weight

Question #5

Row-wise pruning

Come up with a strategy to prune a whole row.

2 minutes

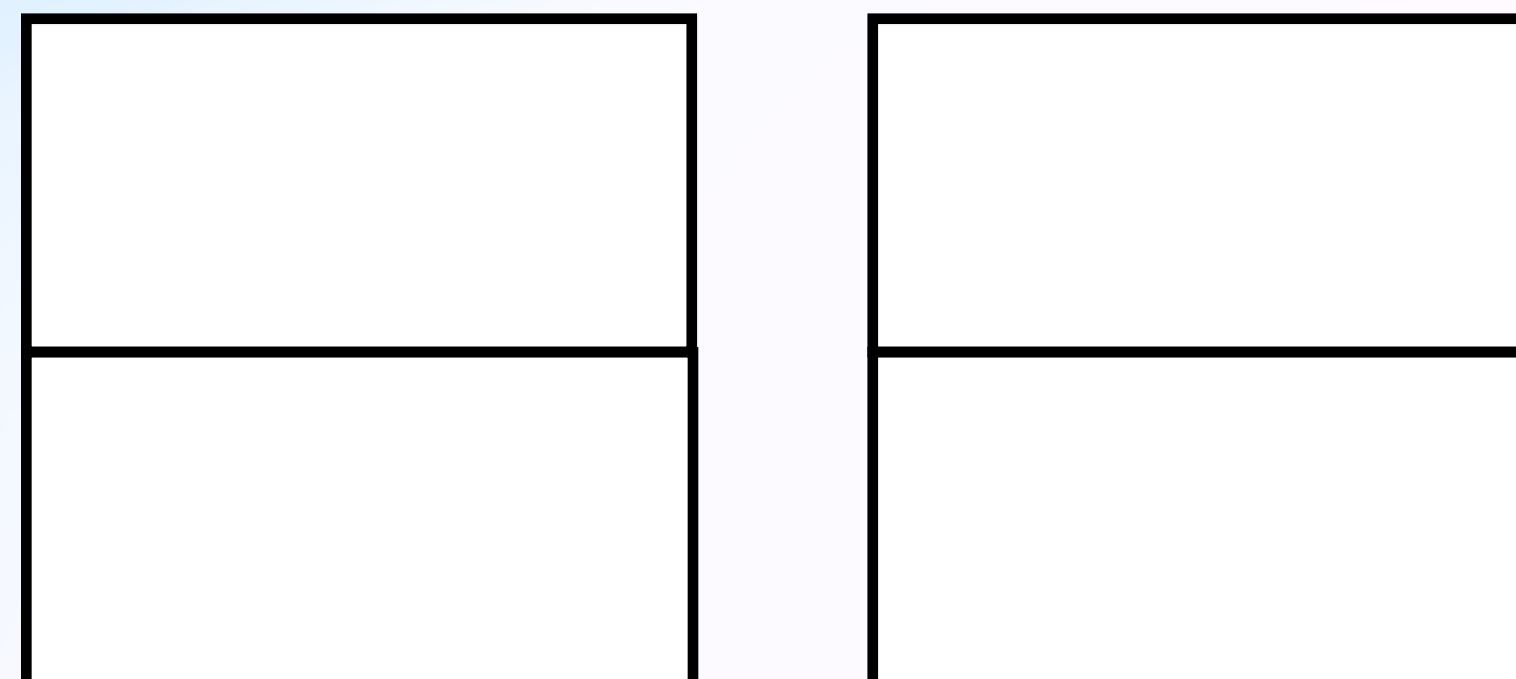
Importance = *define it as a function of the weights*



2	-4
7	-1

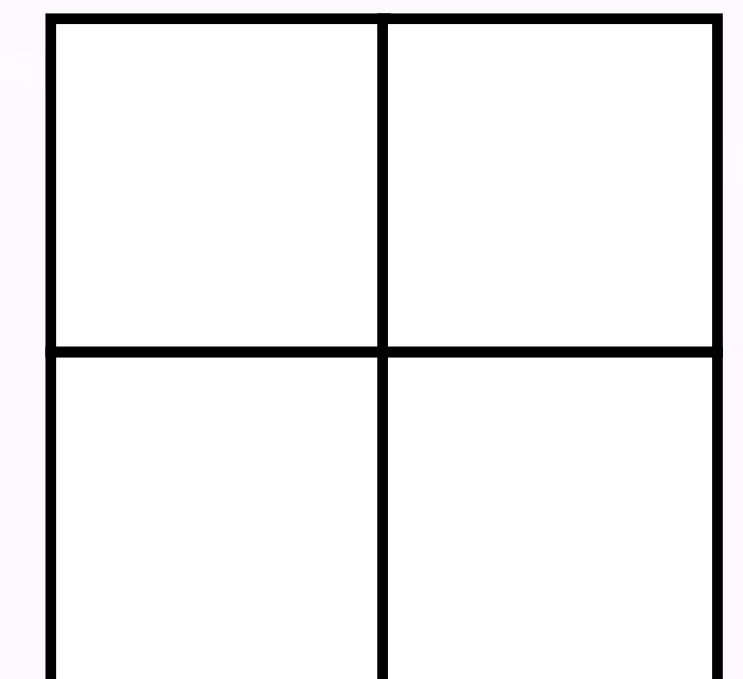
Original Weight

Row-wise →



Importance

→



Pruned Weight

Magnitude-based Pruning

Row-wise pruning using $L1$ -norm

Importance = $\sum_{i \in S} |w_i|$ sum of weights within the structural set (S)

□ Pruned
■ Preserved

2	-4
7	-1

Row-wise

L1-norm

$ 2 + -4 $
$ 7 + -1 $

6
8

0	0
7	-1

Original Weight

Importance

Pruned Weight

Magnitude-based Pruning

Row-wise pruning using L2-norm

$$\text{Importance} = \sqrt{\sum_{i \in S} |w_i|^2}$$

sum of weights square within the structural set (S)

□ Pruned
■ Preserved

2	-4
7	-1

Original Weight

Row-wise
L2-norm

$\sqrt{ 2 ^2 + -4 ^2}$ $= \sqrt{20}$
$\sqrt{ 7 ^2 + -1 ^2}$ $= \sqrt{50}$

Importance

□ $\sqrt{20}$
■ $\sqrt{50}$

→

0	0
7	-1

Pruned Weight

There are many other ways!

We highlighted **magnitude-based weight pruning**

→ You can use any fancy or complicated function that meets your requirements

- **Neuron Pruning**

Some other methods:

- **Second-order derivative-based pruning**

→ Minimizes the error on loss function introduced by pruned synapses

$$\delta L = L(x; W) - L(x; W_p = W - \delta W)$$

- **Channel pruning for convolution neural networks**

- **Regression pruning**

→ Minimize error of a corresponding layer's output: before and after pruning

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

- What is pruning? How to formulate pruning?
- Determine pruning granularity and criteria
- **Network performance after pruning**

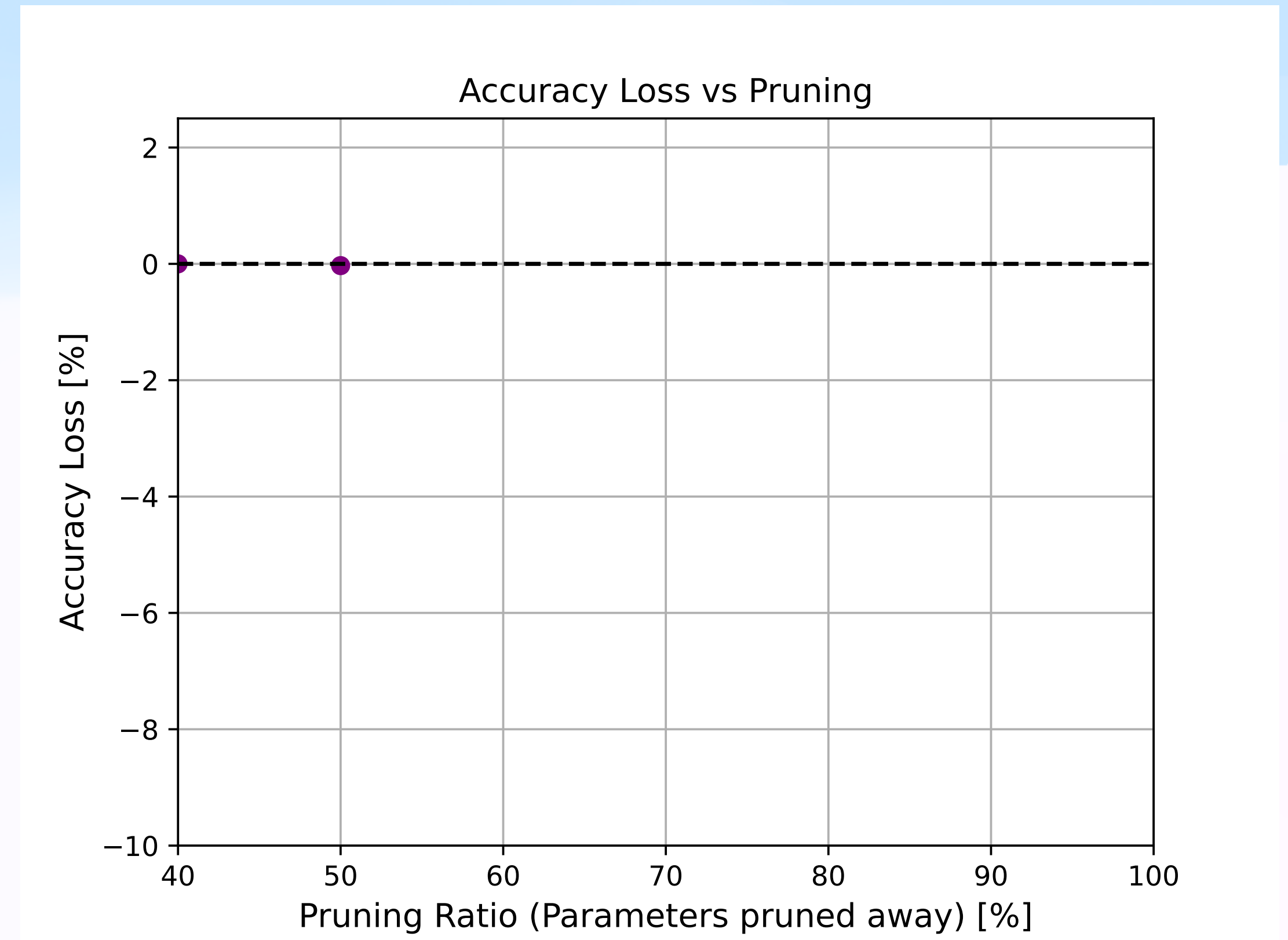
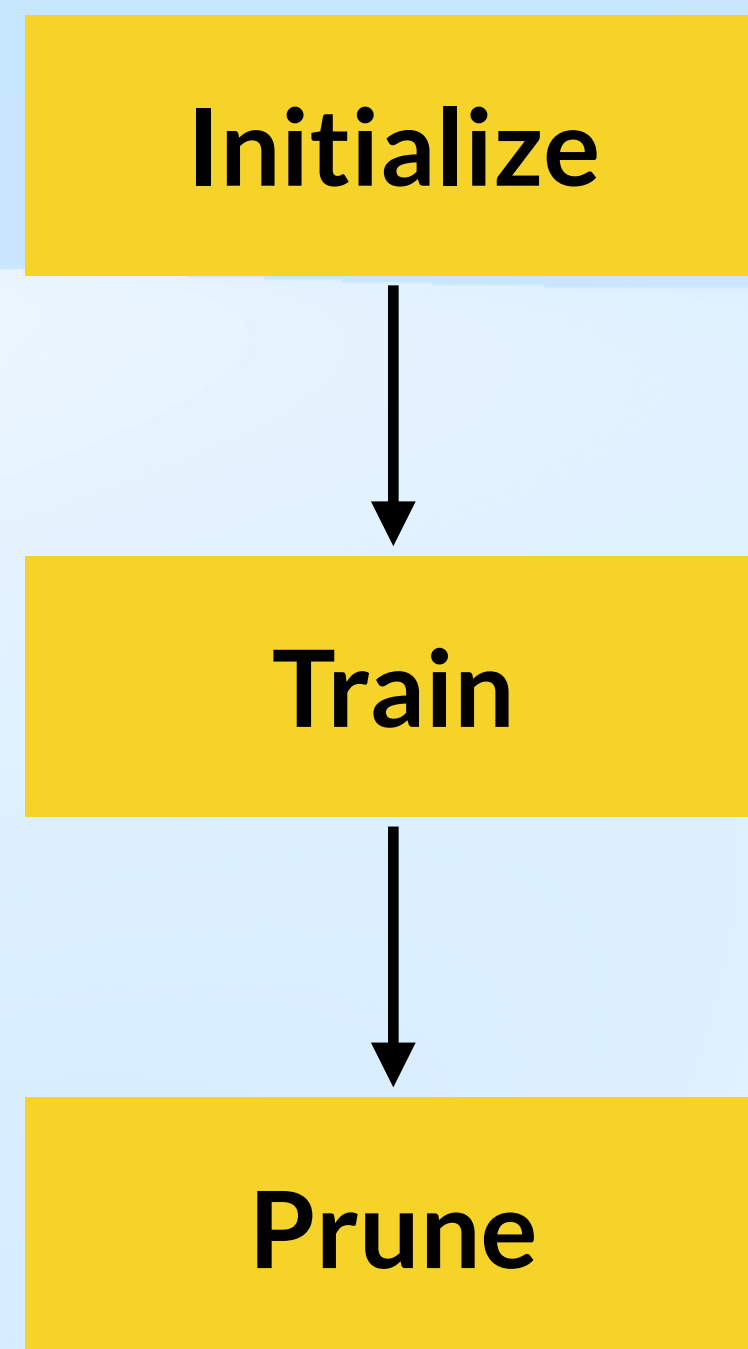
ML on FPGA

hls4ml and Trigger applications

Question #6

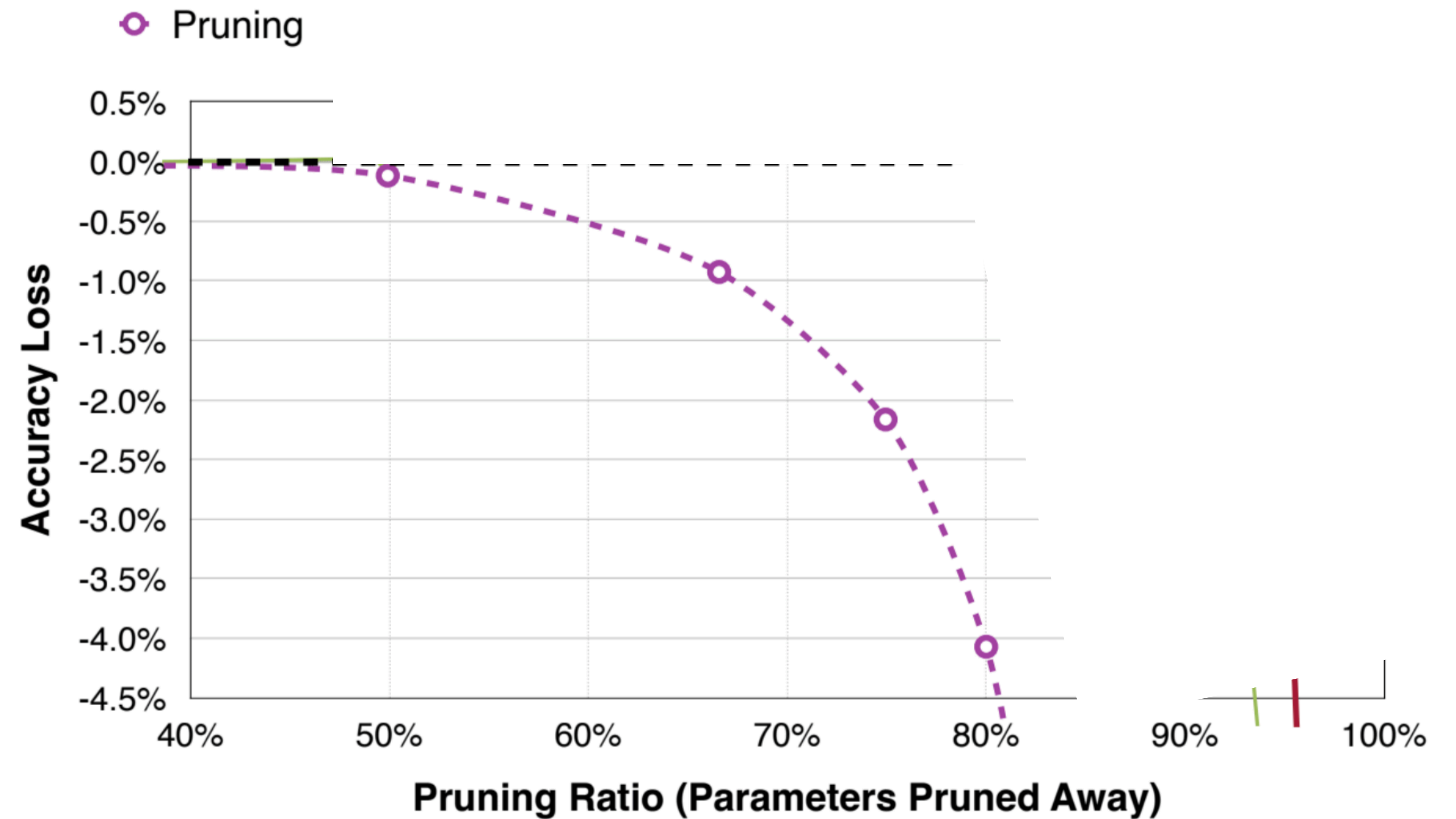
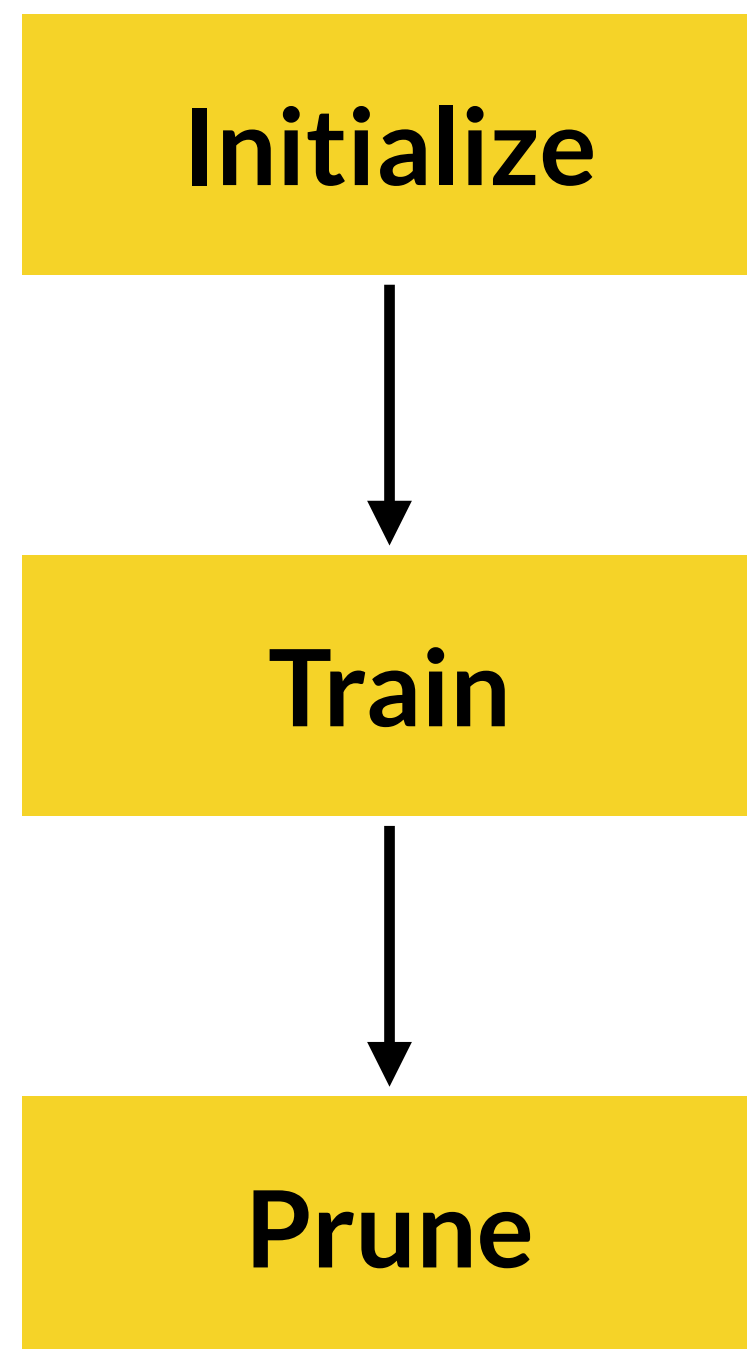
What trend do you expect in the accuracy-loss vs pruning ratio plot beyond 50% pruning ratio?

Assume: the model performance did not degrade after 50% pruning



Accuracy loss in pruning

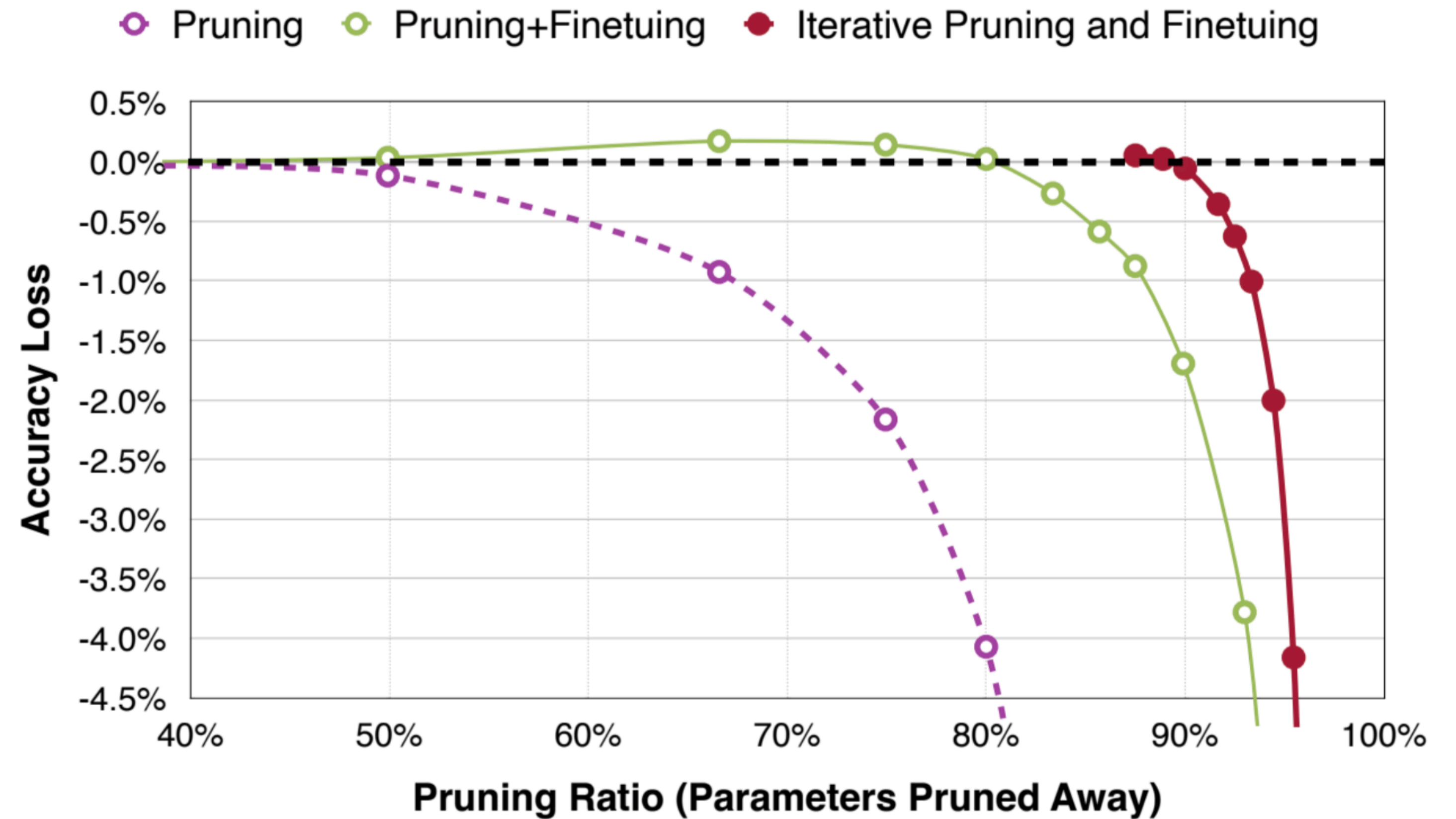
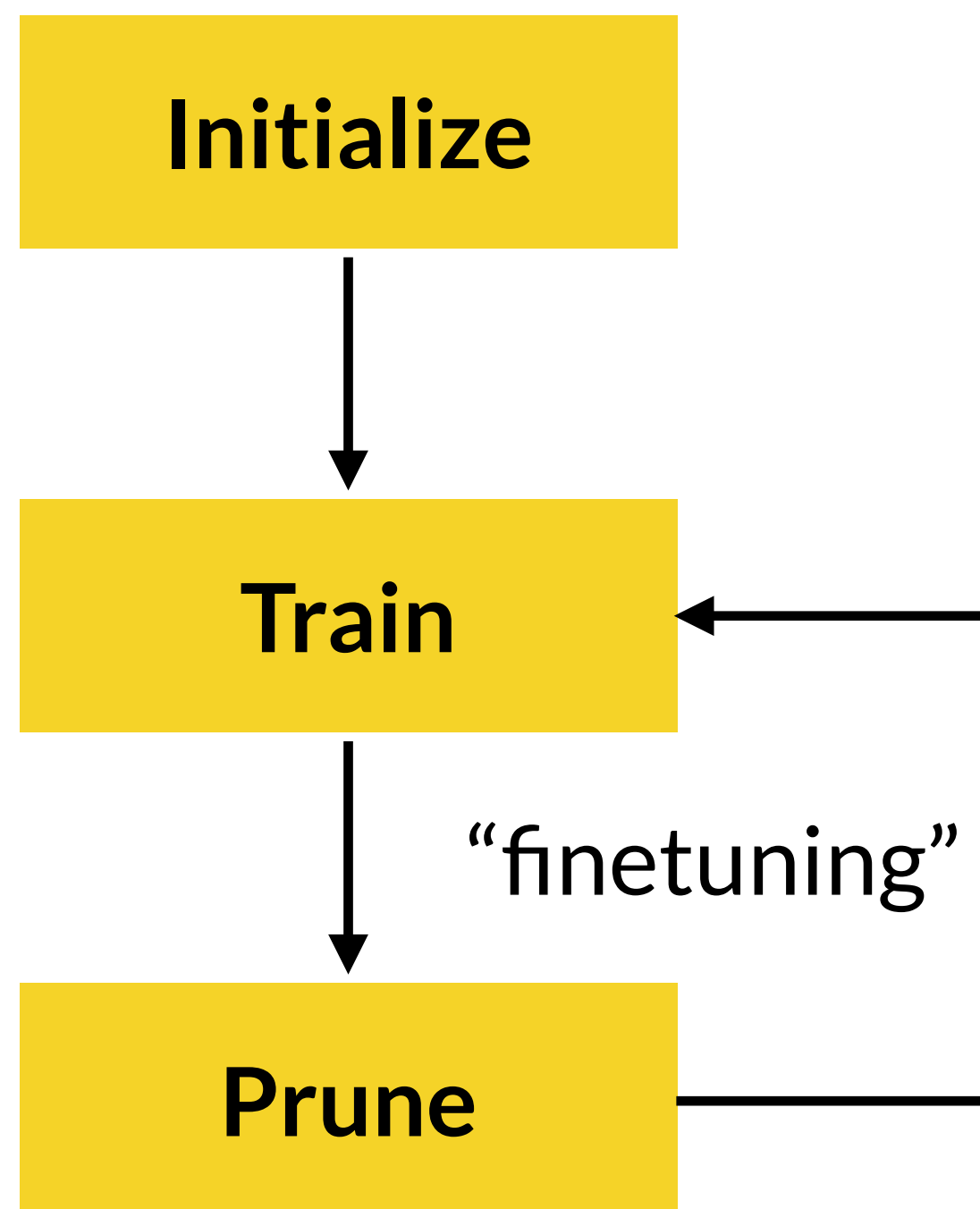
The model performance may decrease after pruning



Learning both Weights and Connections for Efficient Neural Networks {Han et al., NeurIPS 2015}

Accuracy loss in pruning

The model performance may decrease after pruning



Learning both Weights and Connections for Efficient Neural Networks {Han et al., NeurIPS 2015}

Pruning References

Neural Network pruning survey paper:

<https://arxiv.org/abs/2308.06767>

MIT Efficient ML Course:

<https://hanlab.mit.edu/courses/2023-fall-65940>

TensorFlow pruning guide:

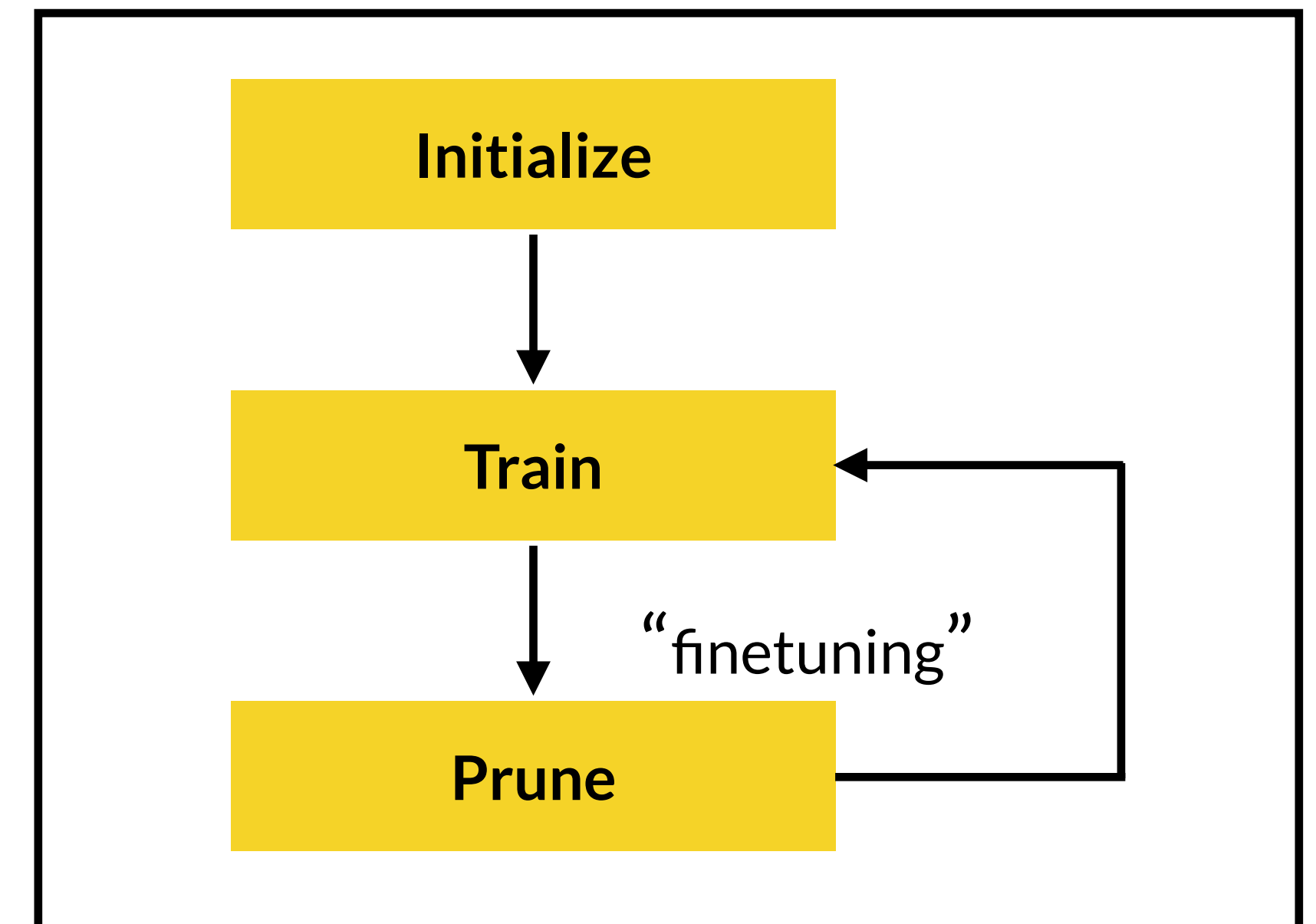
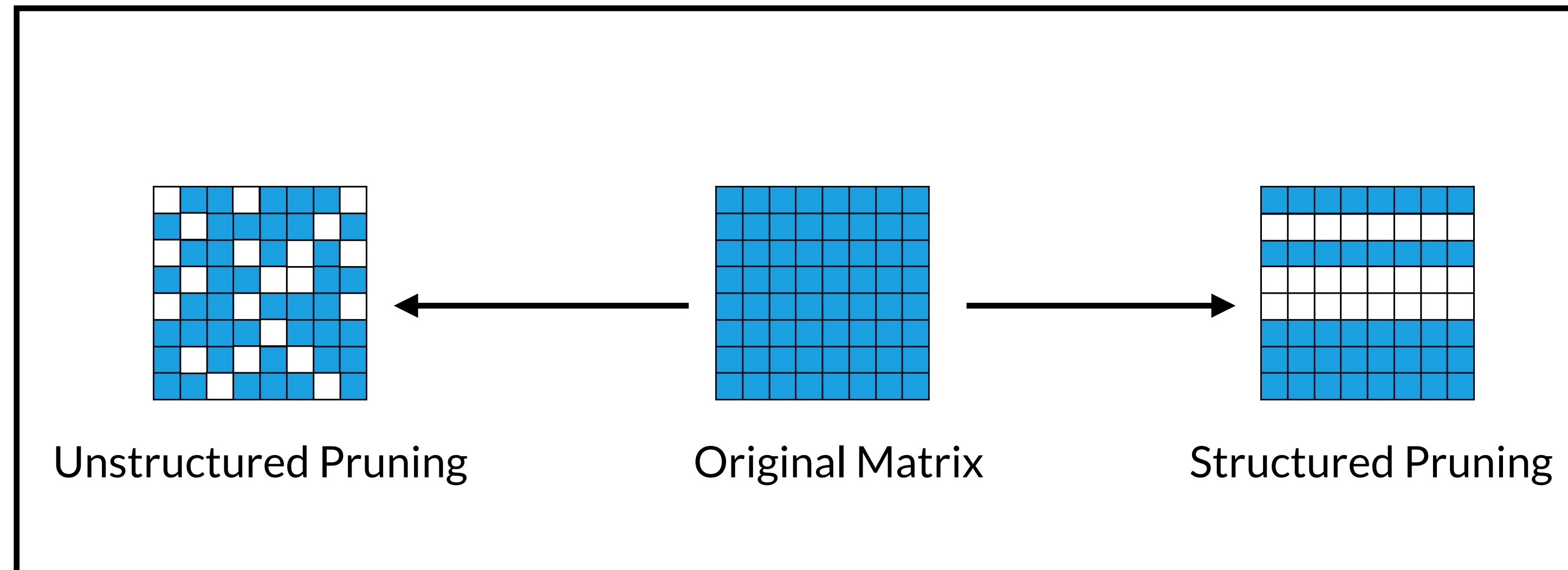
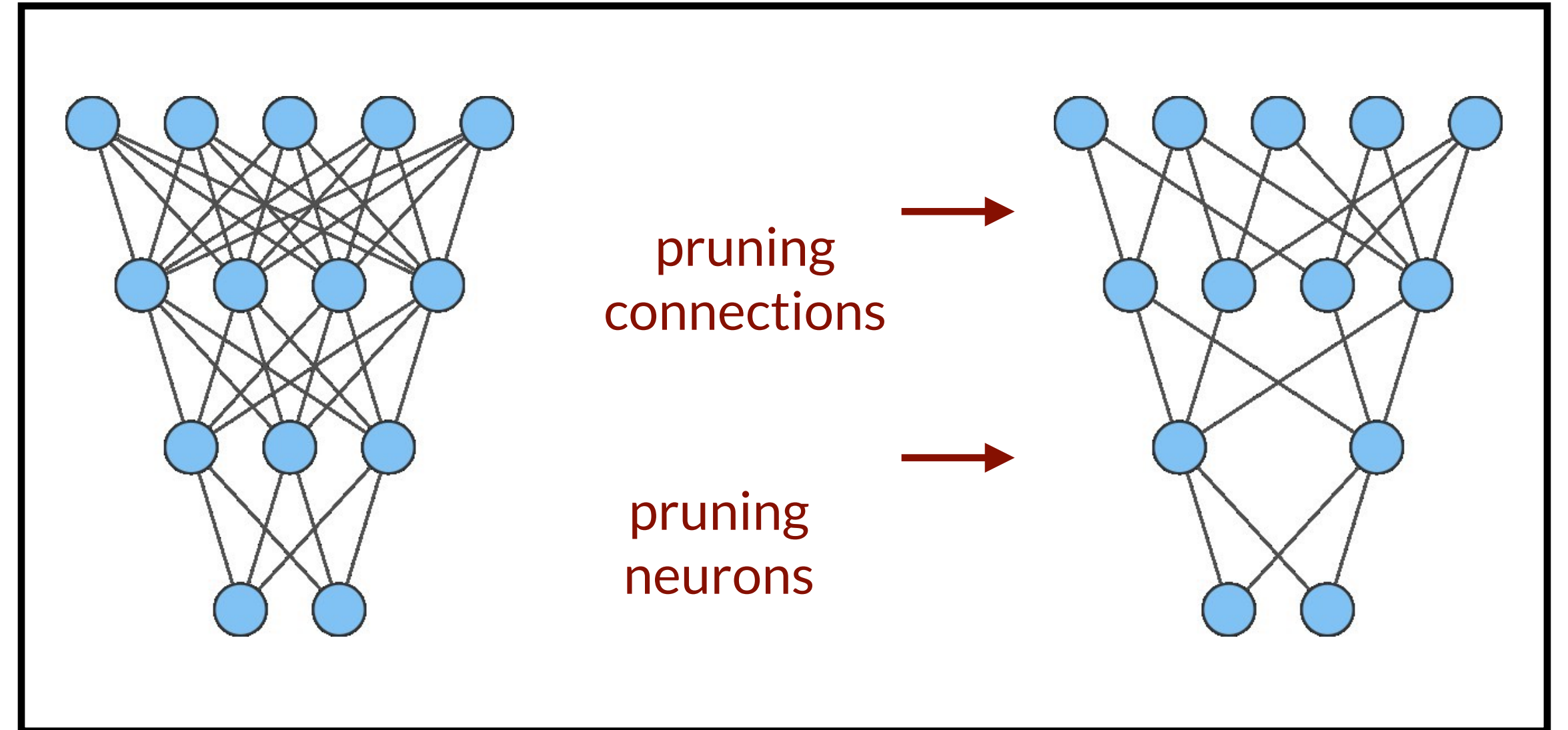
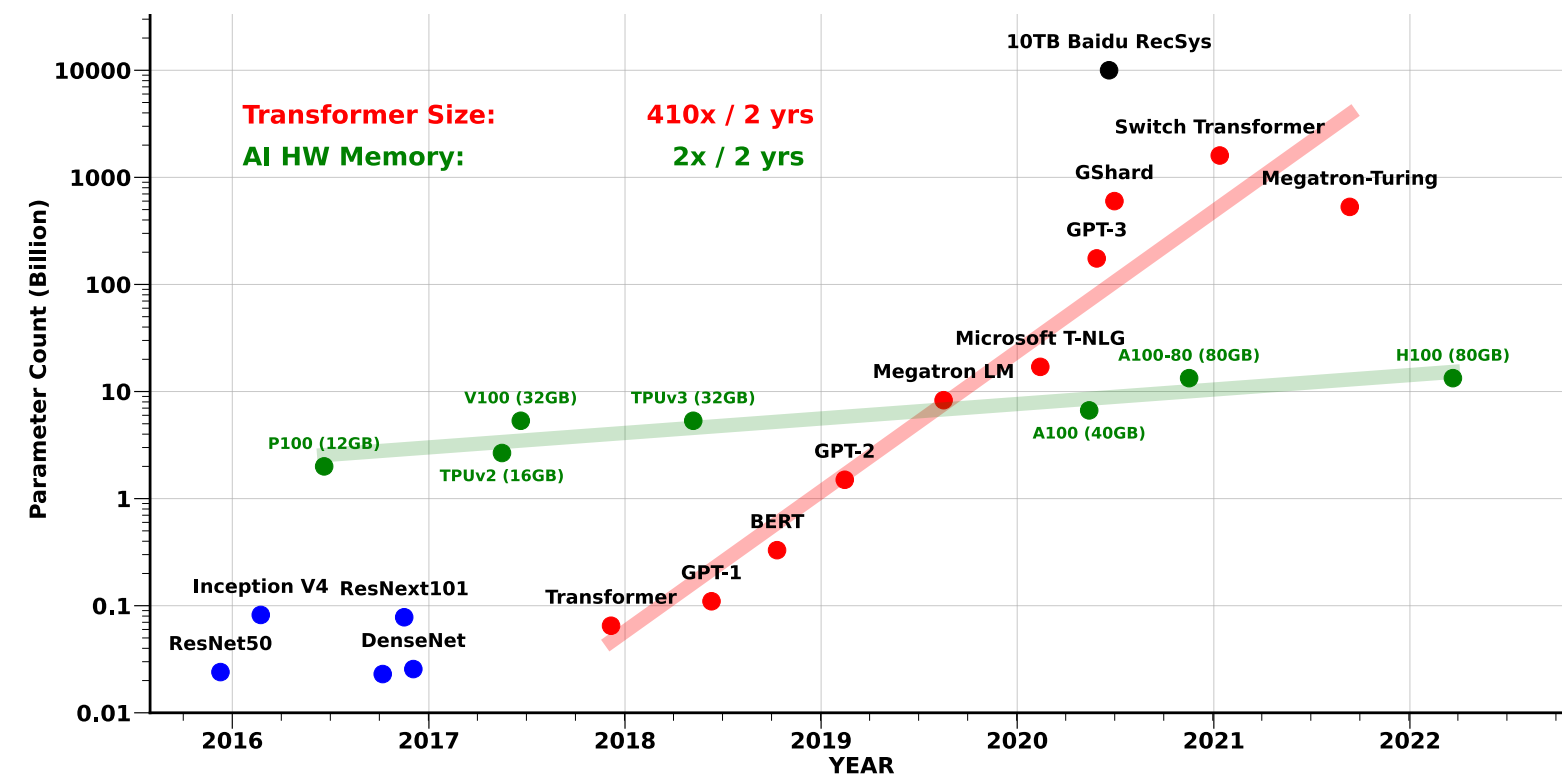
https://www.tensorflow.org/model_optimization/guide/pruning/comprehensive_guide

PyTorch pruning tutorial:

https://pytorch.org/tutorials/intermediate/pruning_tutorial.html

Key Points

Expensive large models



Let's practice it

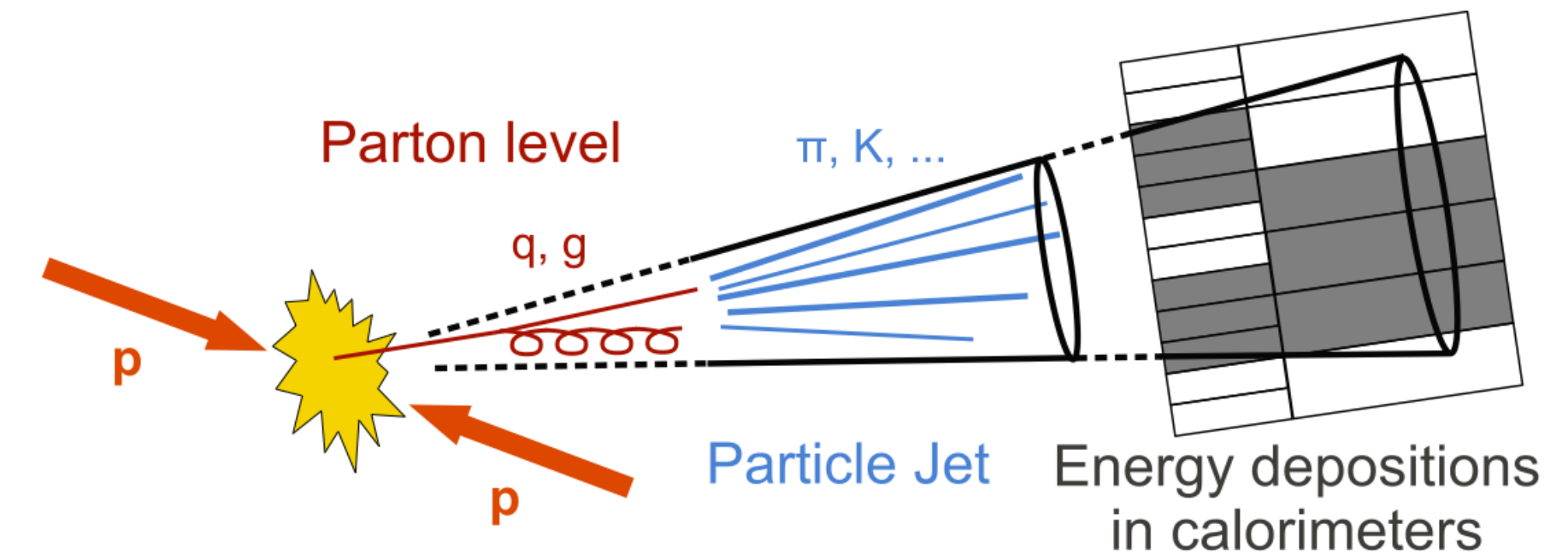
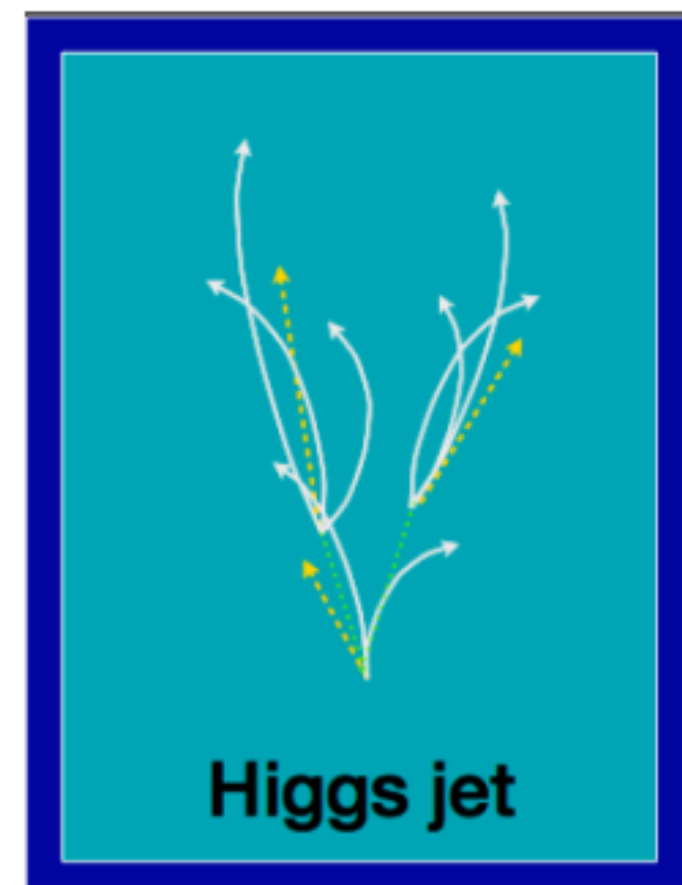
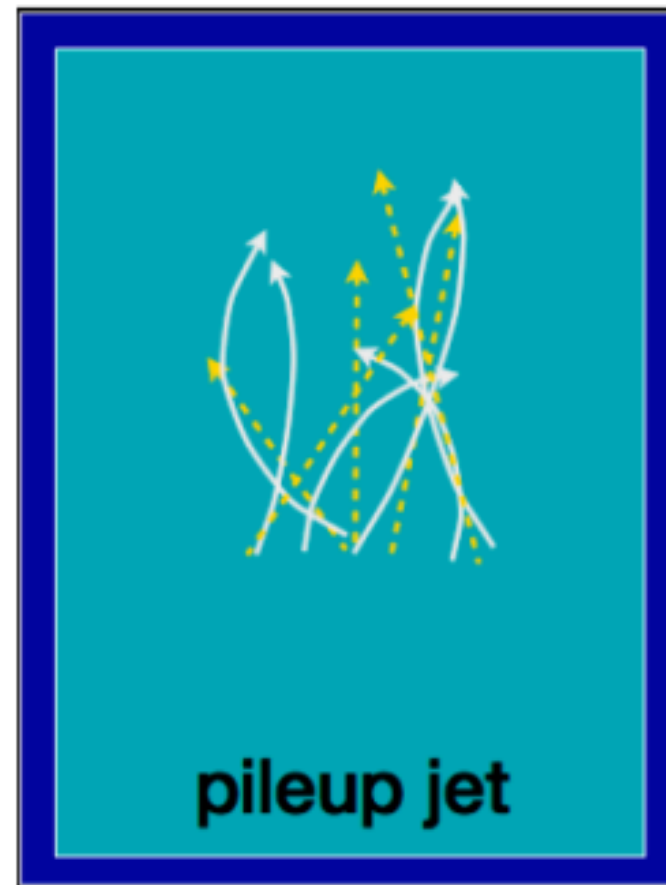
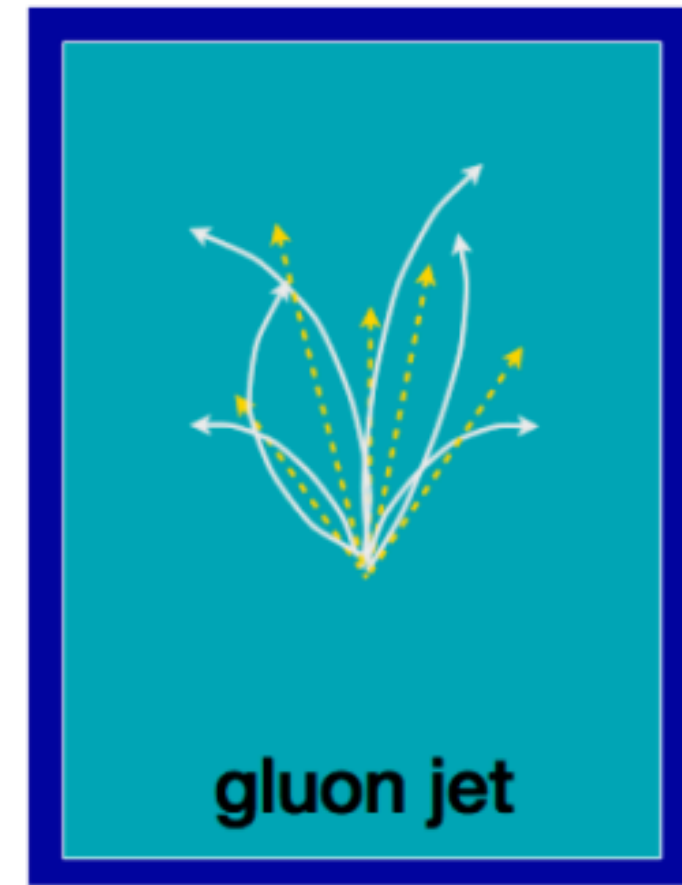
Jupyter Link

<https://jupyter.neresc.gov/>

Tutorial Repo on GitHub

https://github.com/ml4fp/2026-gatech/tree/main/sessions/05_efficient_ml

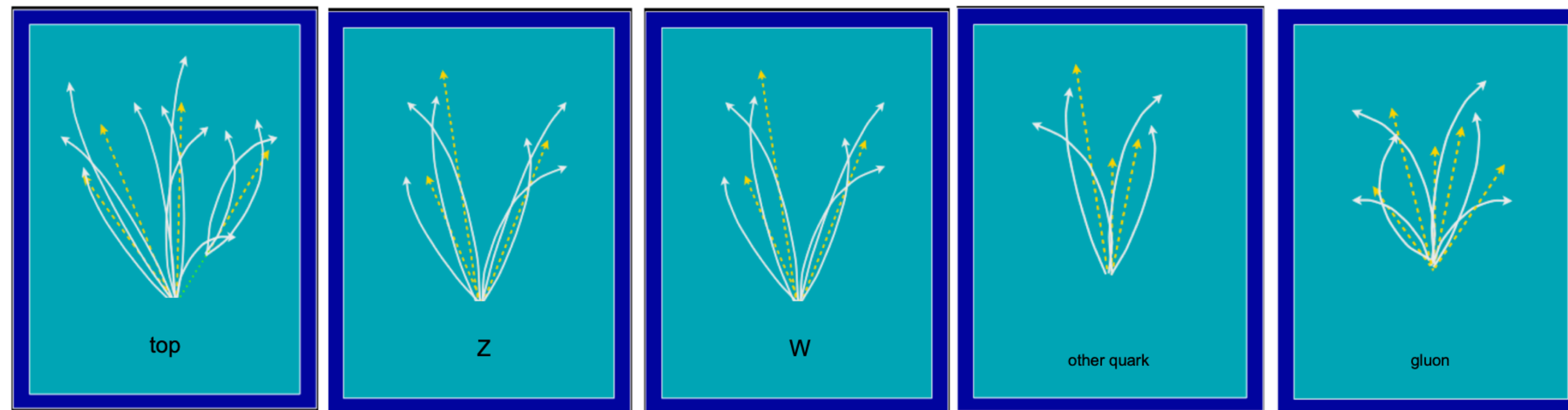
Example: Jet Classification



Jet Classification: 5-Class classifier

Five class classifier

Sample: ~ 1M events with two boosted WW/ZZ/tt/qq/gg anti-kT jets

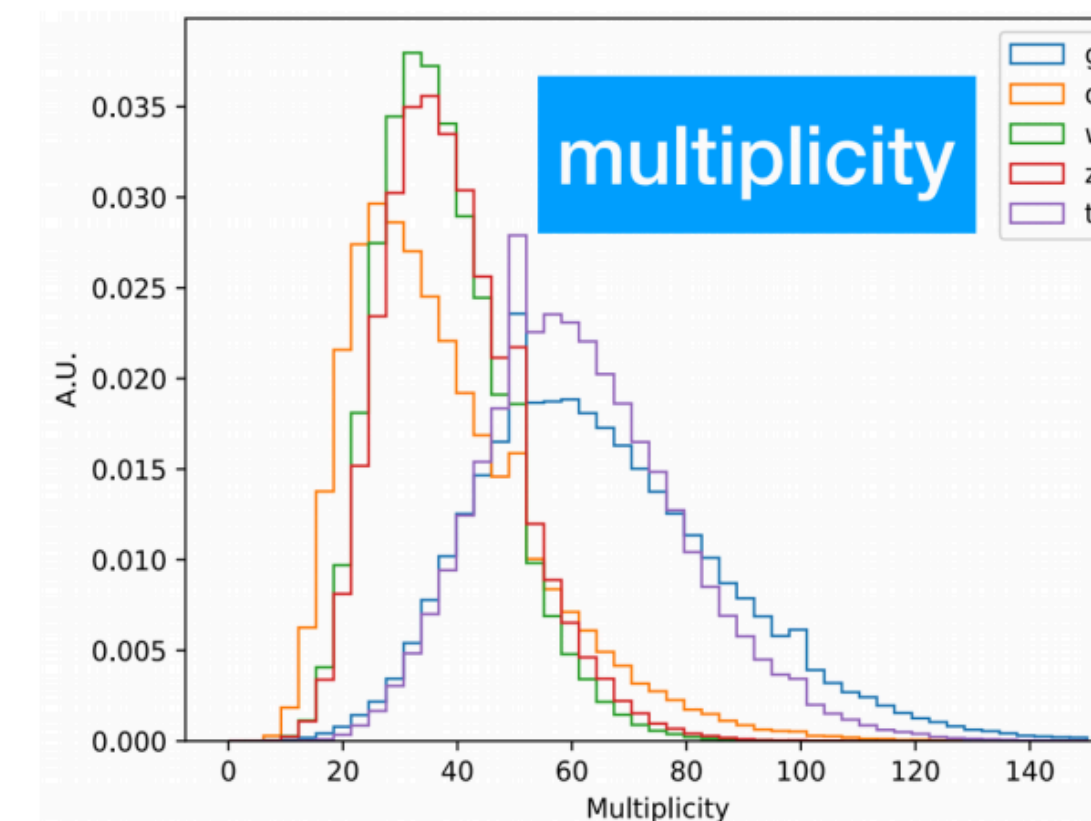
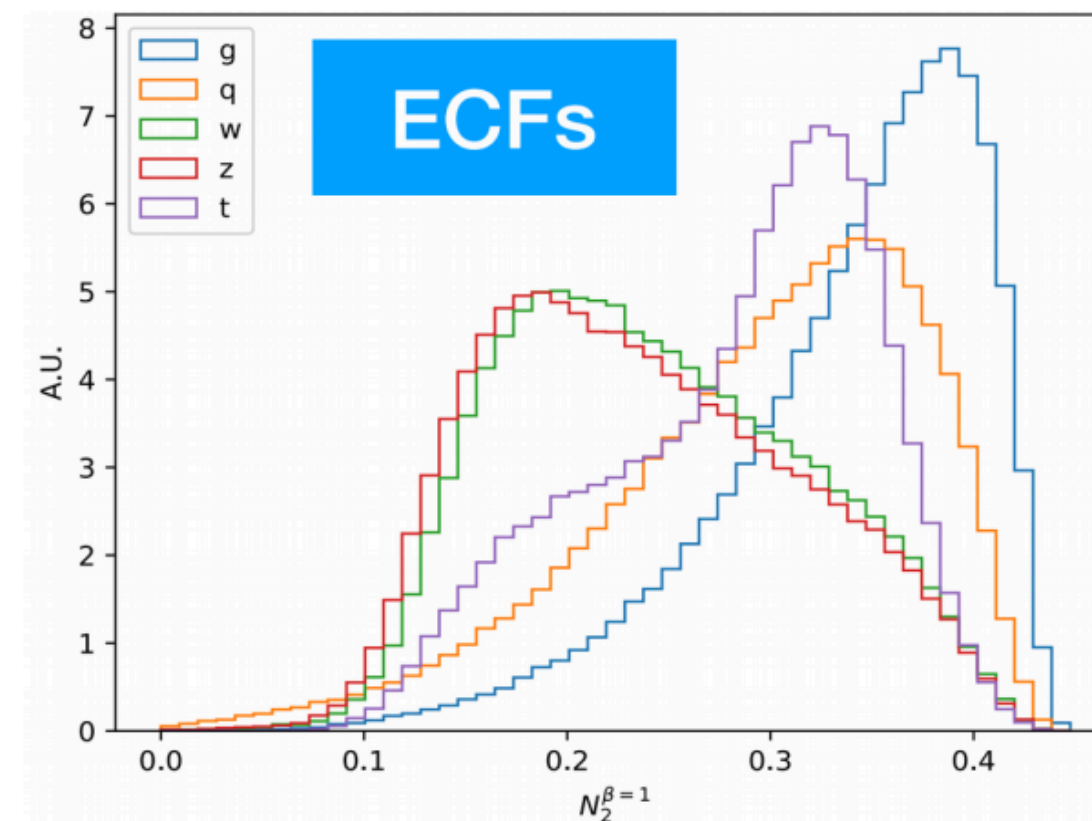
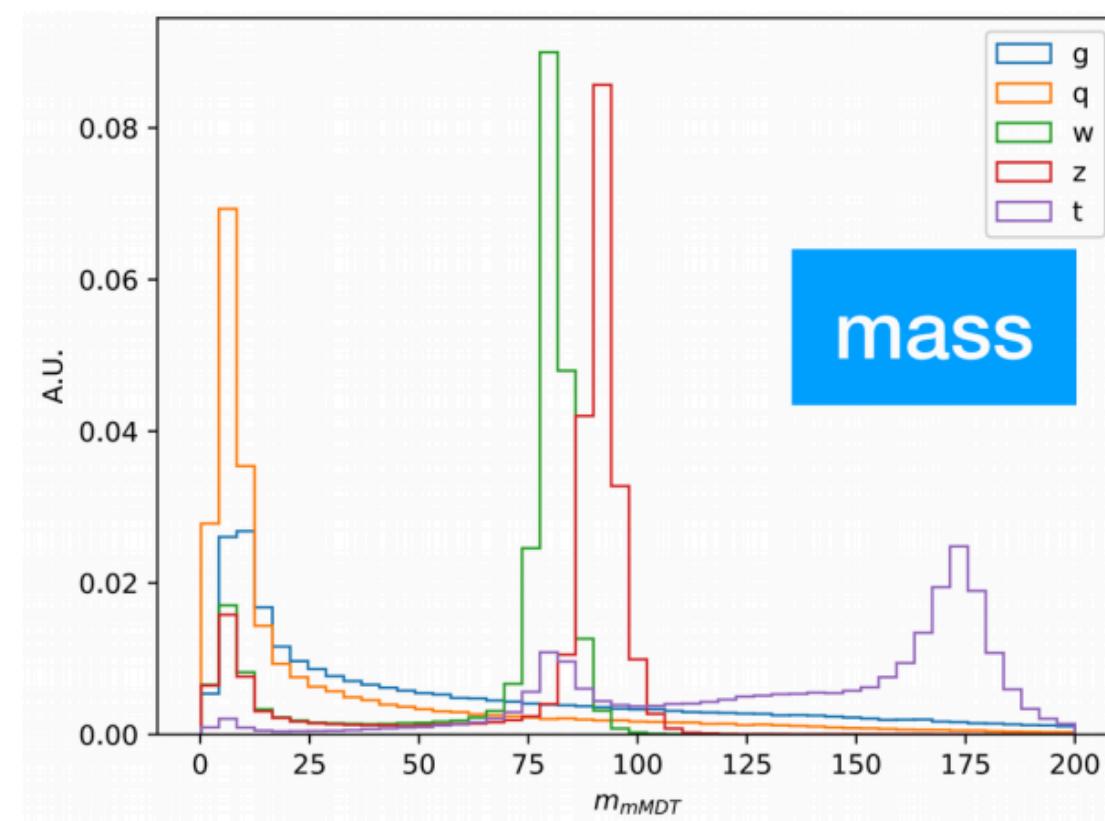


$t \rightarrow bW \rightarrow bqq$

$Z \rightarrow qq$

$W \rightarrow qq$

q/g background



Observables

- m_{mMDT}
- $N_2^{\beta=1,2}$
- $M_2^{\beta=1,2}$
- $C_1^{\beta=0,1,2}$
- $C_2^{\beta=1,2}$
- $D_2^{\beta=1,2}$
- $D_2^{(\alpha,\beta)=(1,1),(1,2)}$
- $\sum z \log z$
- Multiplicity

Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

Introduction to Pruning

- What is pruning? How to formulate pruning?
- Determine pruning granularity and criteria
- Network performance after pruning

ML on FPGA

hls4ml and Trigger applications

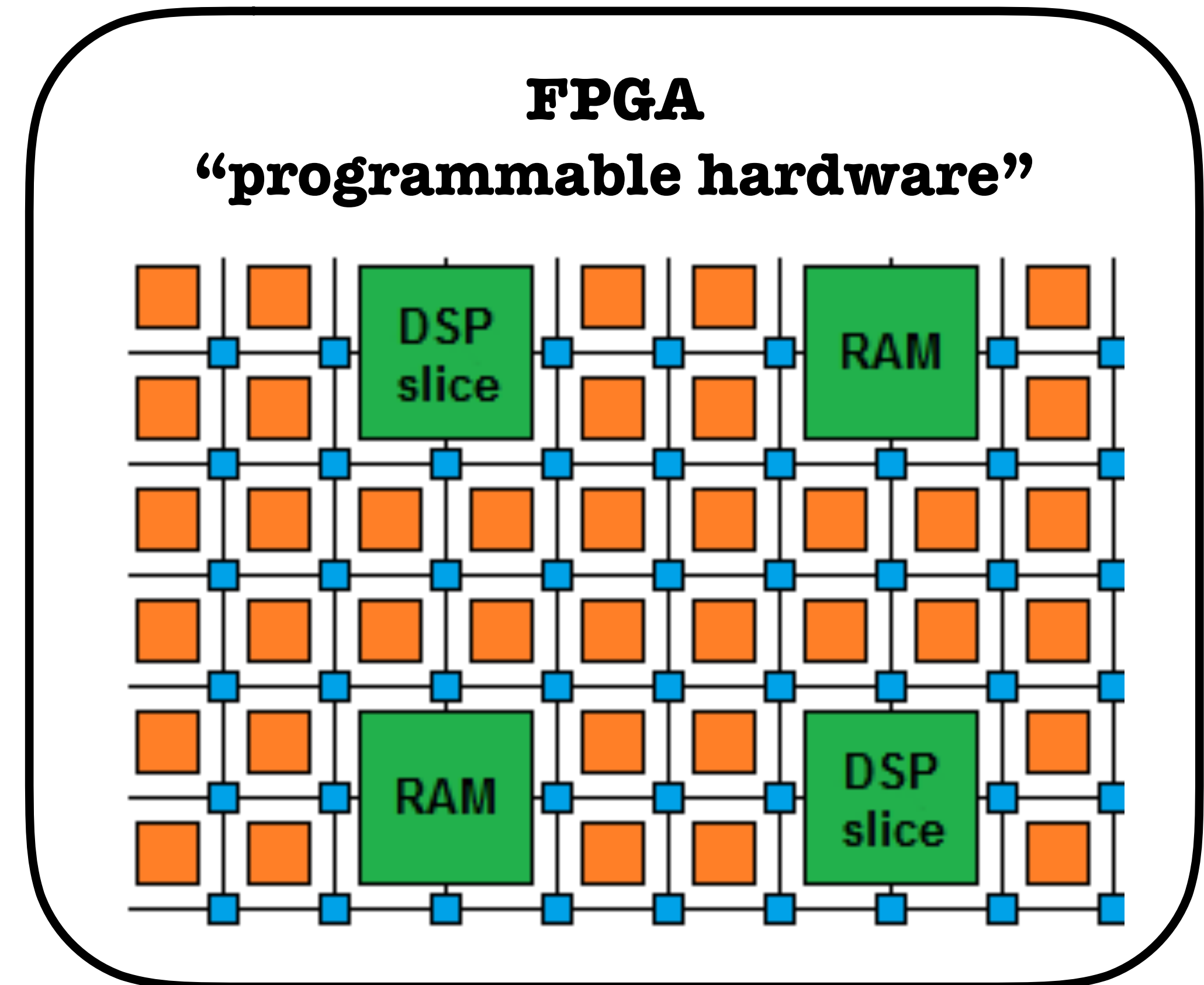
Modern FPGAs

Pros:

- Reprogrammable interconnects between embedded components that perform multiplication (DSPs), apply logical functions (LUTs), or store memory (BRAM)
- High throughput I/O: O(100) optical transceivers running at O(15) Gbps
- Massively parallel
- Low power

Cons:

- Requires domain knowledge to program (using VHDL/Verilog)



Why FPGA at LHC?



High parallelism \uparrow = Low latency \downarrow

- Can work on different data simultaneously (pipelining)! High bandwidth

Power efficient

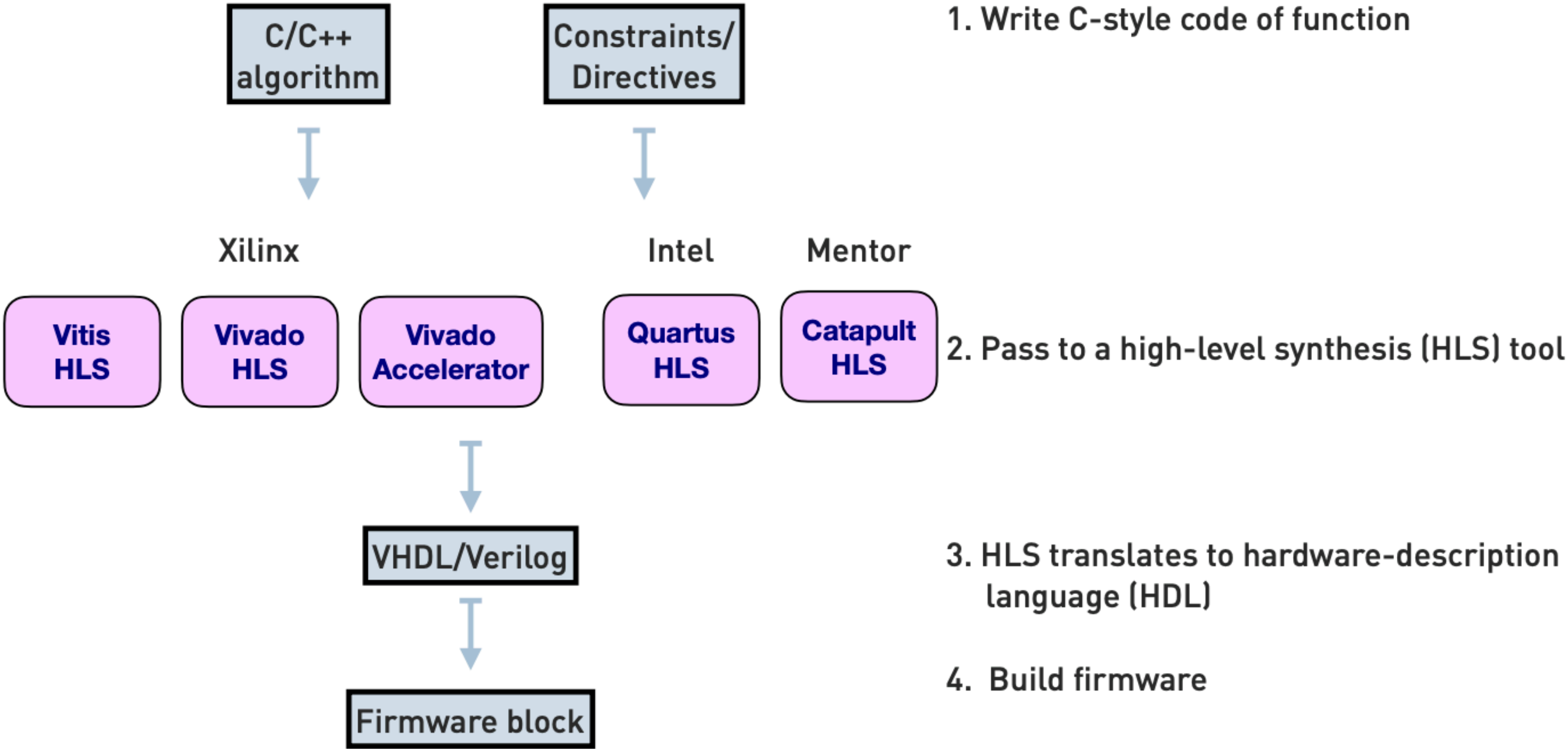
- FPGAs ~x10 more power efficient than GPUs

Latency deterministic

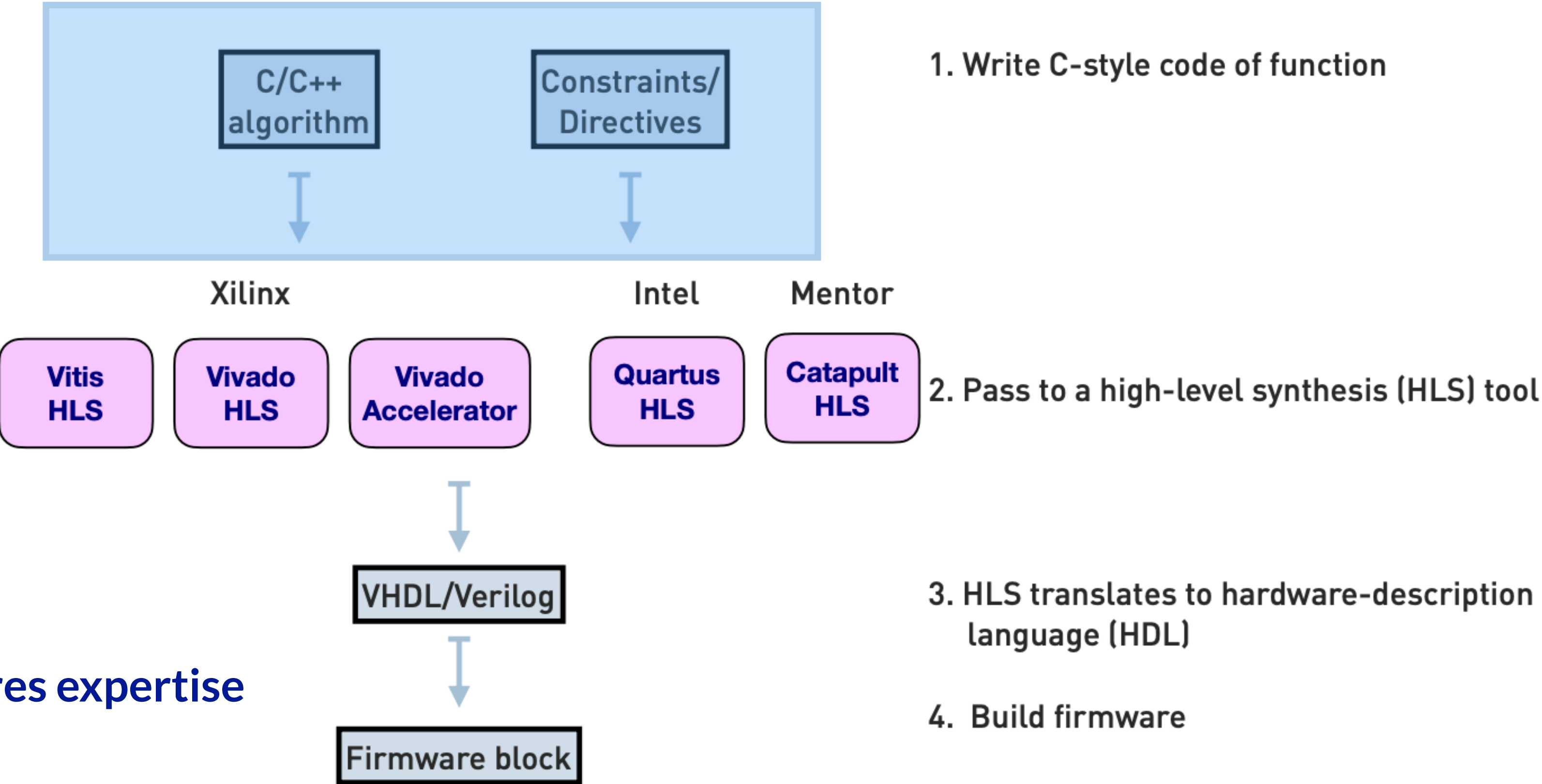
- FPGAs repeatable and predictable latency

Latency is fixed by proton collisions occurring at 40 MHz, cannot tolerate slack

FPGA Programming



FPGA Programming

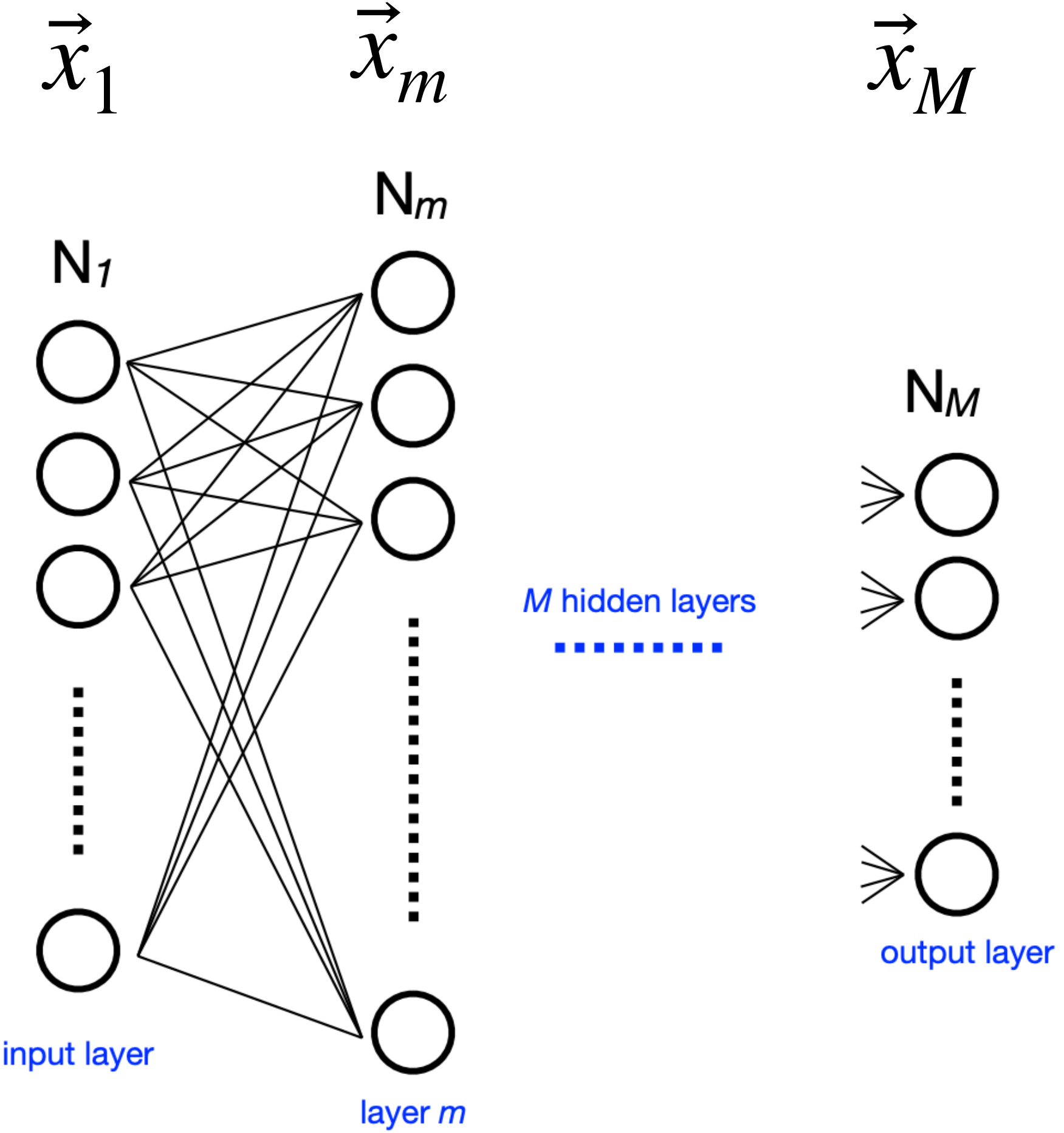


Efficient L1T firmware design requires expertise

- FPGA deployment in busy devices
- $\ll 1\mu\text{s}$ latency target

Not well served by industry tools!

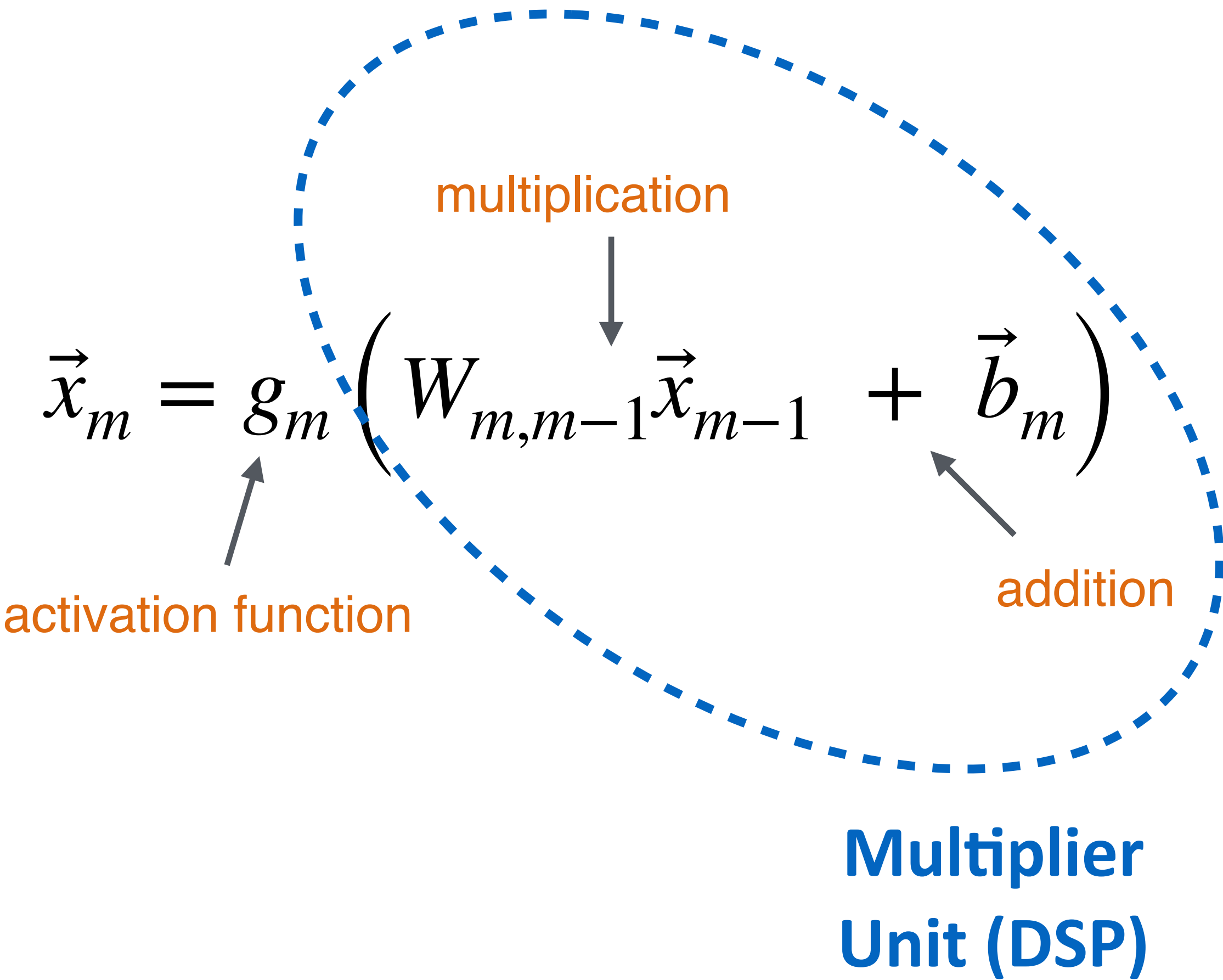
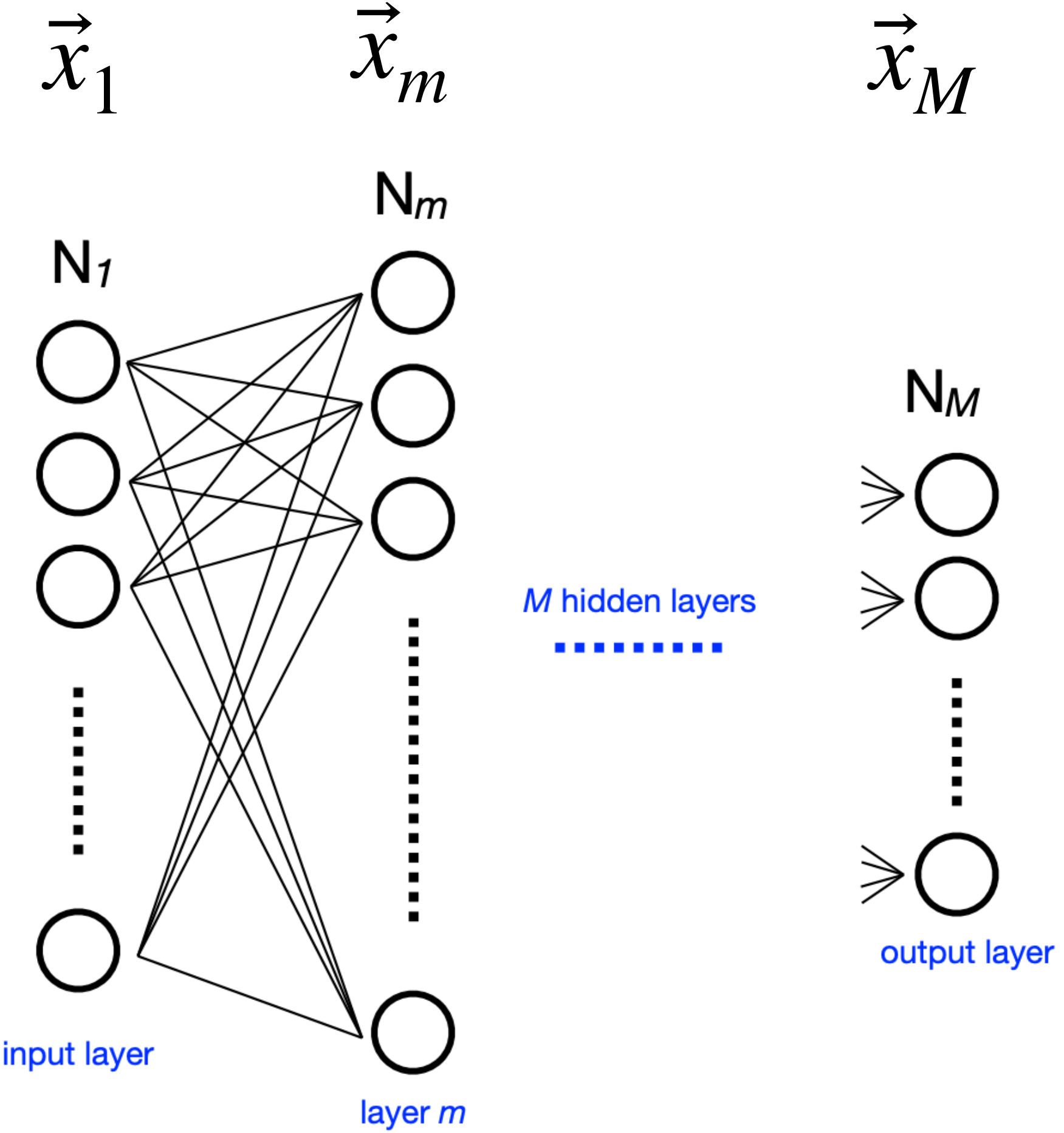
Inference on FPGA



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

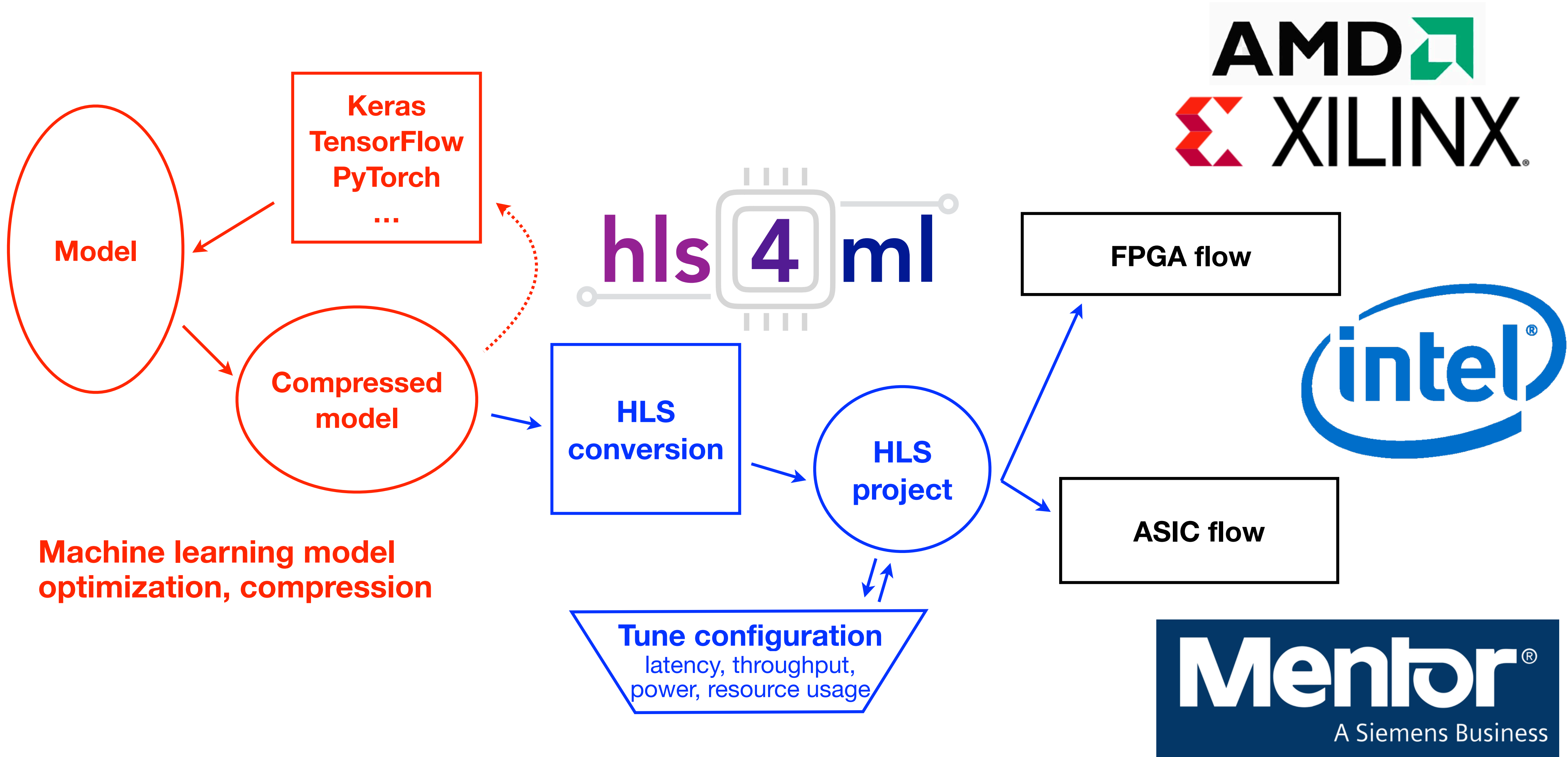
multiplication (pointing to $W_{m,m-1} \vec{x}_{m-1}$)
 activation function (pointing to g_m)
 addition (pointing to $+$)

Inference on FPGA



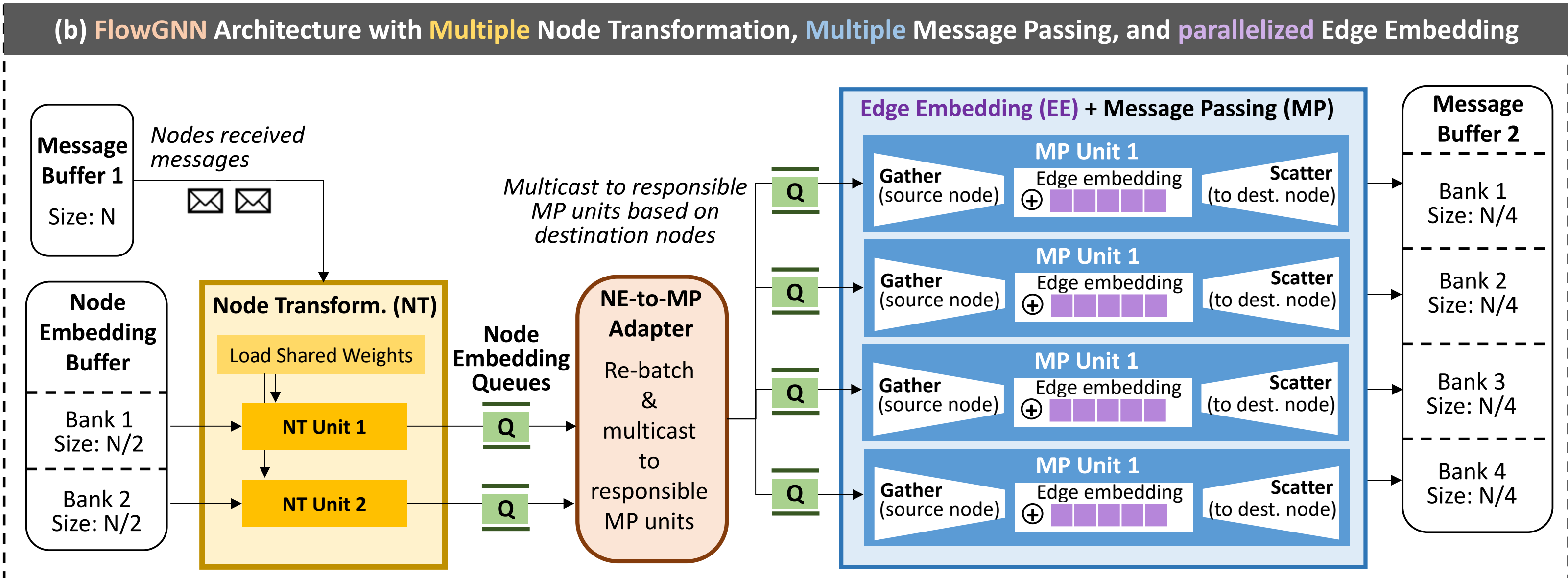
up to ~6k parallel operation (VU9P)

- [hls4ml](#) for scientists or ML experts to translate ML algorithms into RTL firmware



Many tools with different strengths

- FINN (NNs): <https://finn.readthedocs.io/en/latest/>
- Conifer (BDTs): <https://github.com/thesps/conifer>
- fwXMachina (BDTs): <http://fwx.pitt.edu/>
- FlowGNN: <https://github.com/sharc-lab/flowgnn>



Lecture Outline

Motivation behind efficient model design and some highlights

Introduction to Quantization

- Definition
- PTQ vs QAT

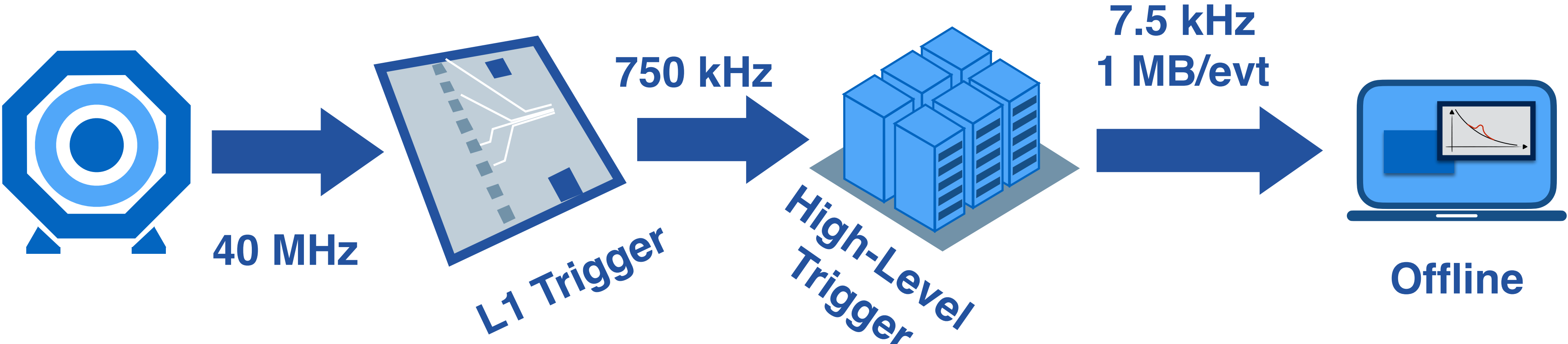
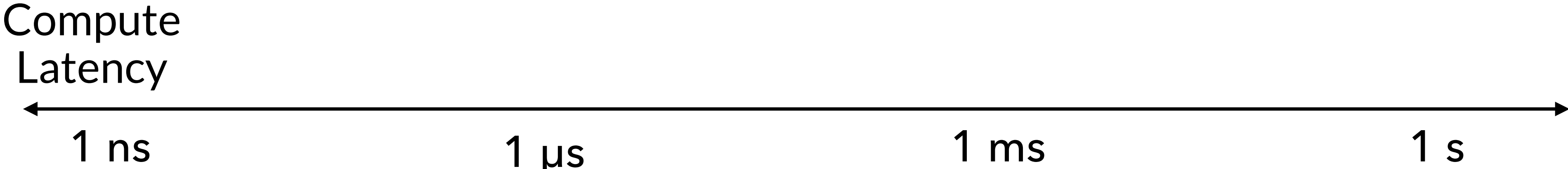
Introduction to Pruning

- What is pruning? How to formulate pruning?
- Determine pruning granularity and criteria
- Network performance after pruning

ML on FPGA

hls4ml and Trigger applications

HL-LHC Data Processing



ASICs

FPGAs

CPUs

GPUs

FPGAs

Other processors:
IPU, TPU ..

CPUs

GPUs

FPGAs

Other processors:
IPU, TPU ..

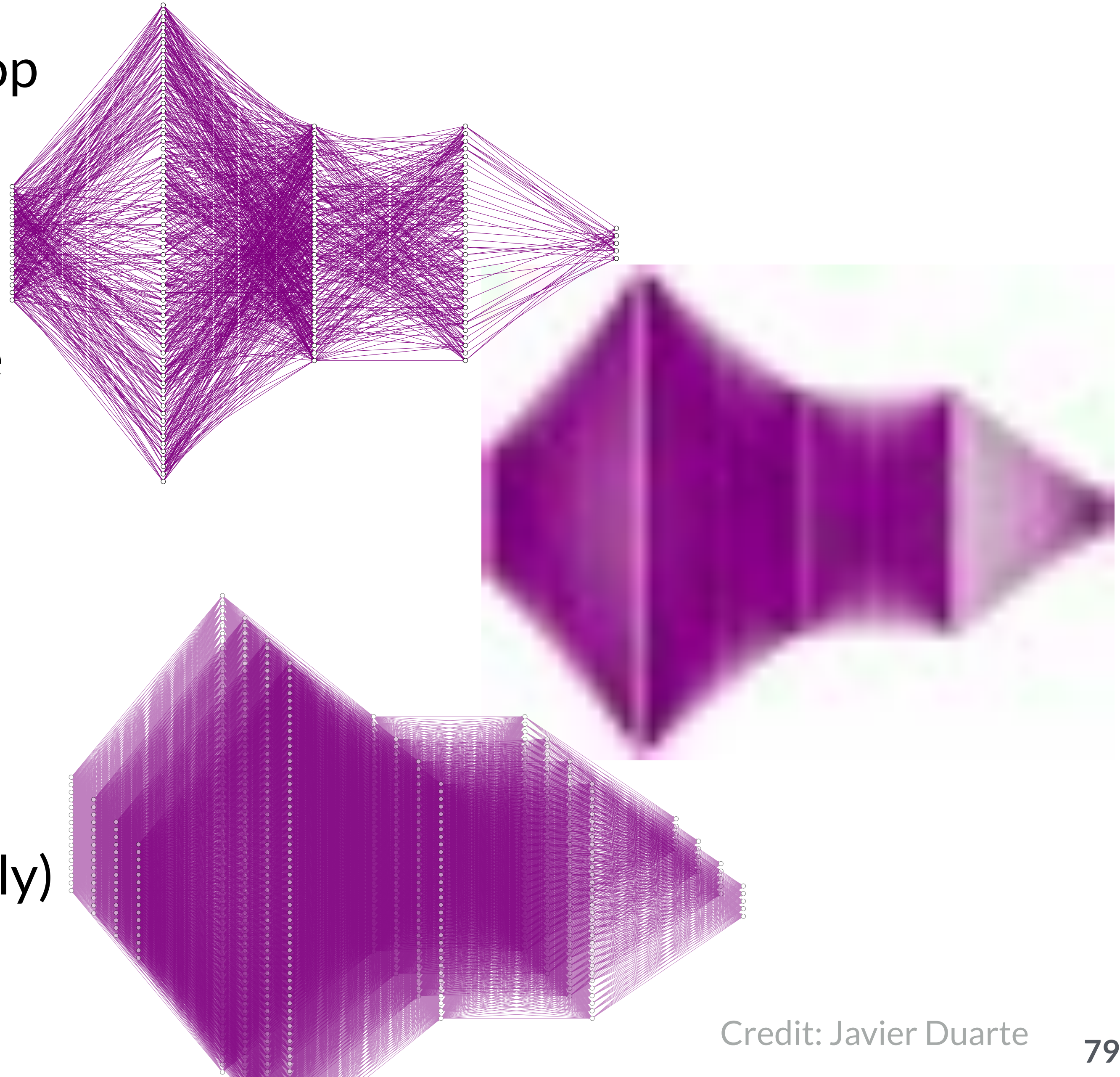
Exabyte-scale datasets

Challenges:

- Each collision produces $O(10^3)$ particles
- The detectors have $O(10^8)$ sensors
- Extreme data rates of $O(100 \text{ TB/s})$

Codesign

- **Codesign:** intrinsic development loop between ML design, training, and implementation
- Pruning
 - Maintain high performance while removing redundant operations
- Quantization
 - Reduce precision from 32-bit floating point to 16-bit, 8-bit, ...
- Parallelization
 - Balance parallelization (how fast) with resources needed (how costly)



Let's Go through the hls4ml Demo



Official hls4ml tutorials:

<https://fastmachinelearning.org/hls4ml-tutorial/README.html>

A software interface for implementing Neural Networks on an FPAG

- Supports many common layer like DNN, CNN, RNN, Graph NN, Transformers, etc

All these ML algorithms could be used for several low-level tasks

Example:

- Jet Energy Calibration
- Missing Transverse Energy reconstruction
- Jet identification
- Anomaly detection

Quantization

Quantization – Reducing the bit precision used for NN arithmetic

Why this is necessary?

- Floating-point operations (32 bit numbers) on an FPGA consumes large resources
- Not necessary to do it for desired performance
- **hls4ml** uses **fixed-point representation** for all computations
 - Operations are integer ops, but we can represent fractional values

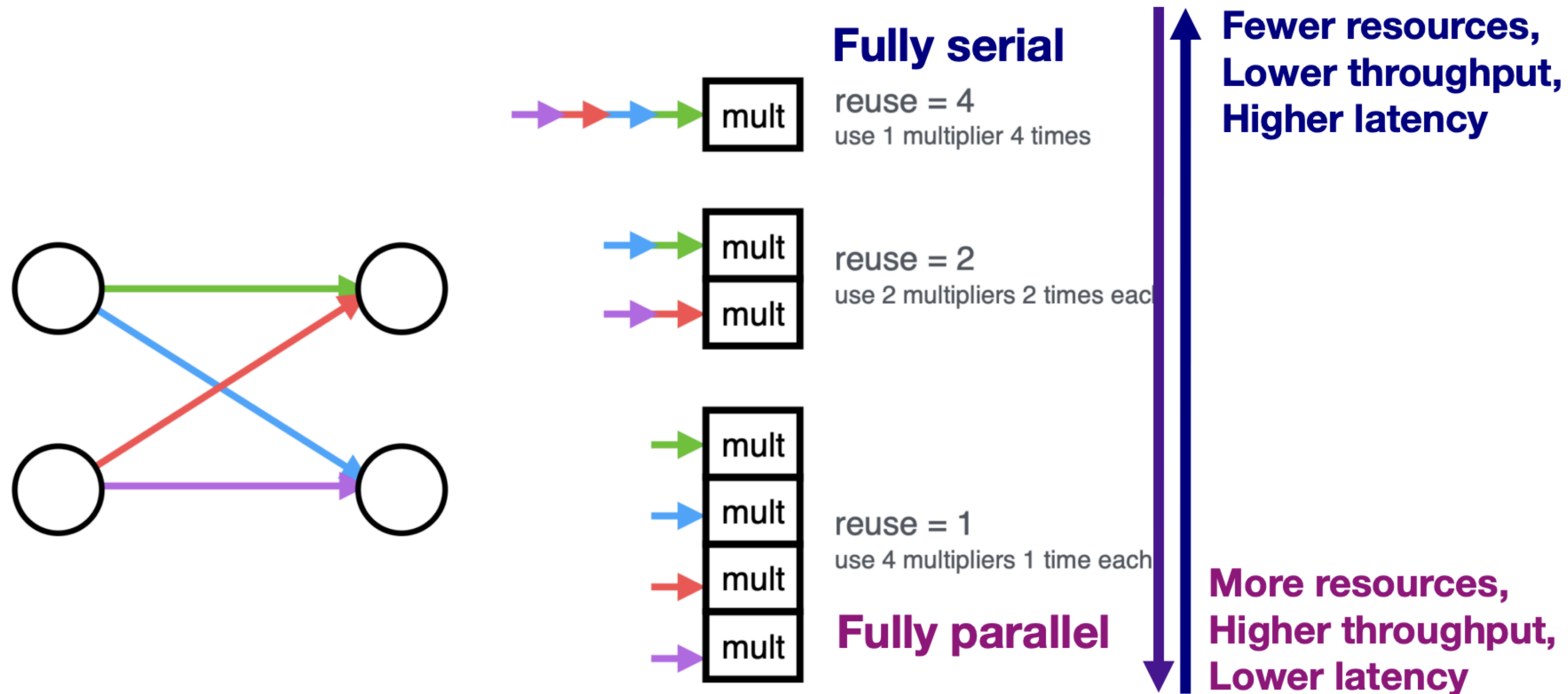
ap_fixed<width bits, integer bits>

0101.1011101010

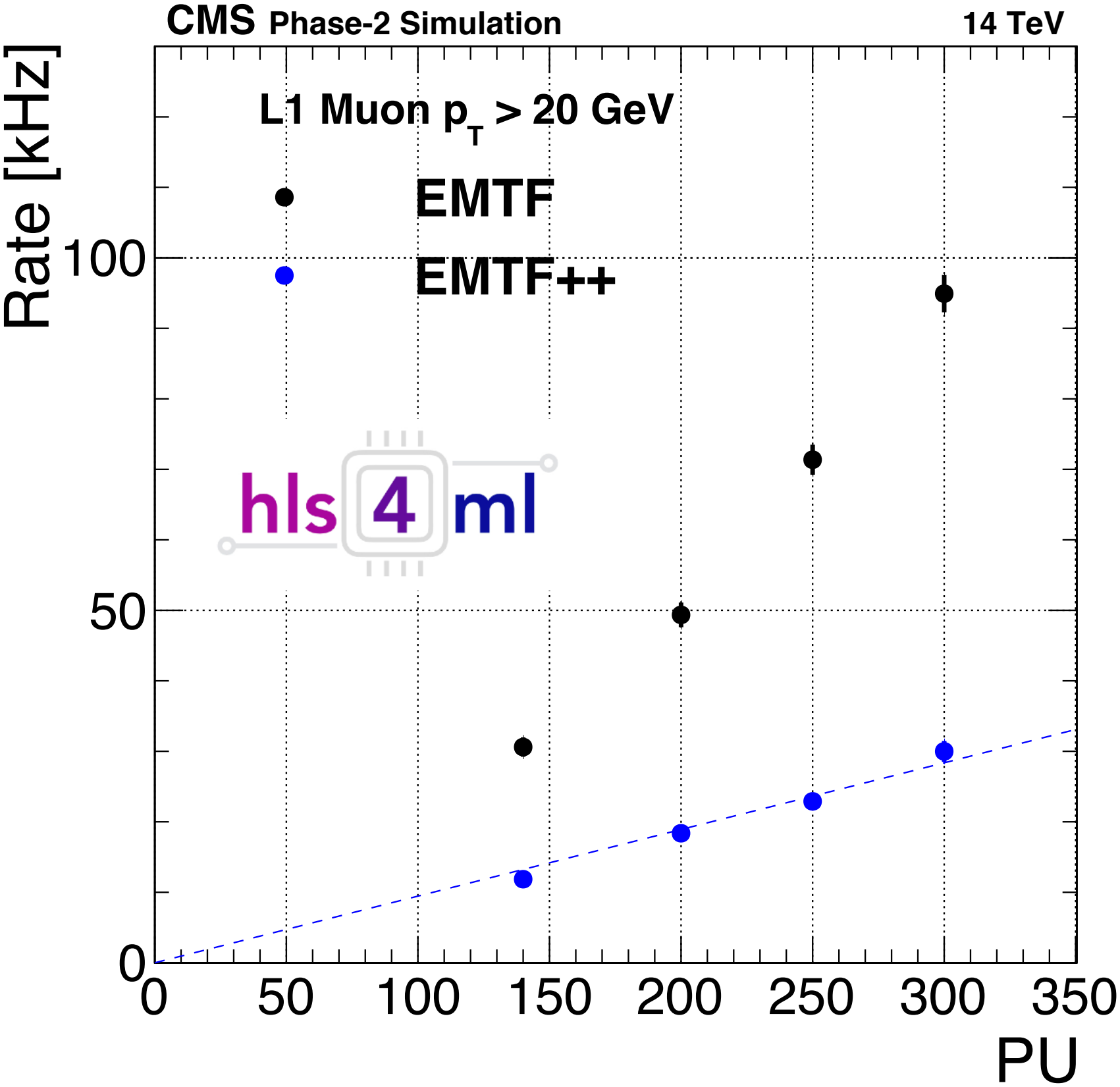
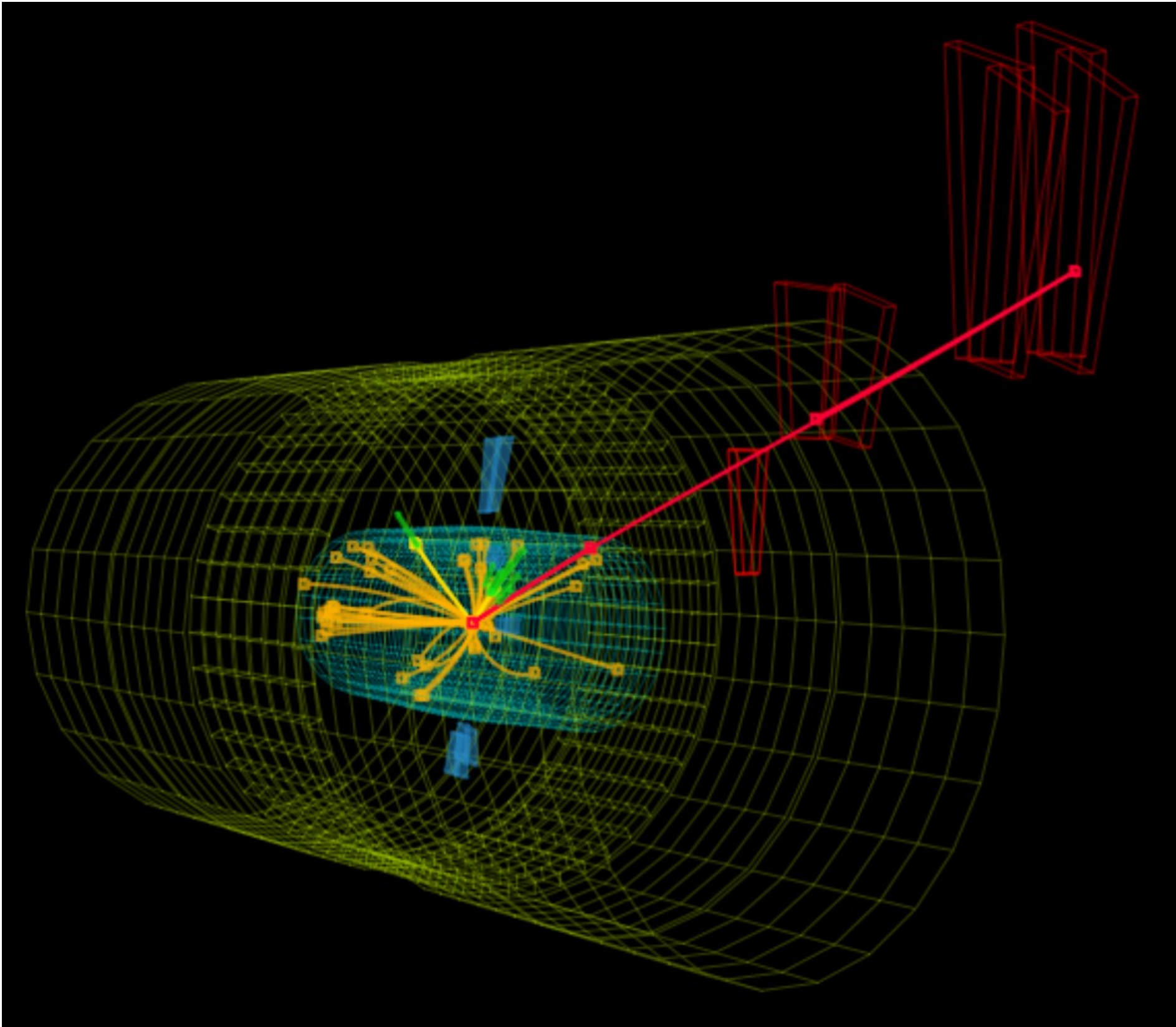


Parallelization

- Trade-off between **latency** and **FPGA resource usage** determined by the **parallelization** of the calculations in each layer
- Configure the “**reuse factor**” = number of times a multiplier is used to do a computation

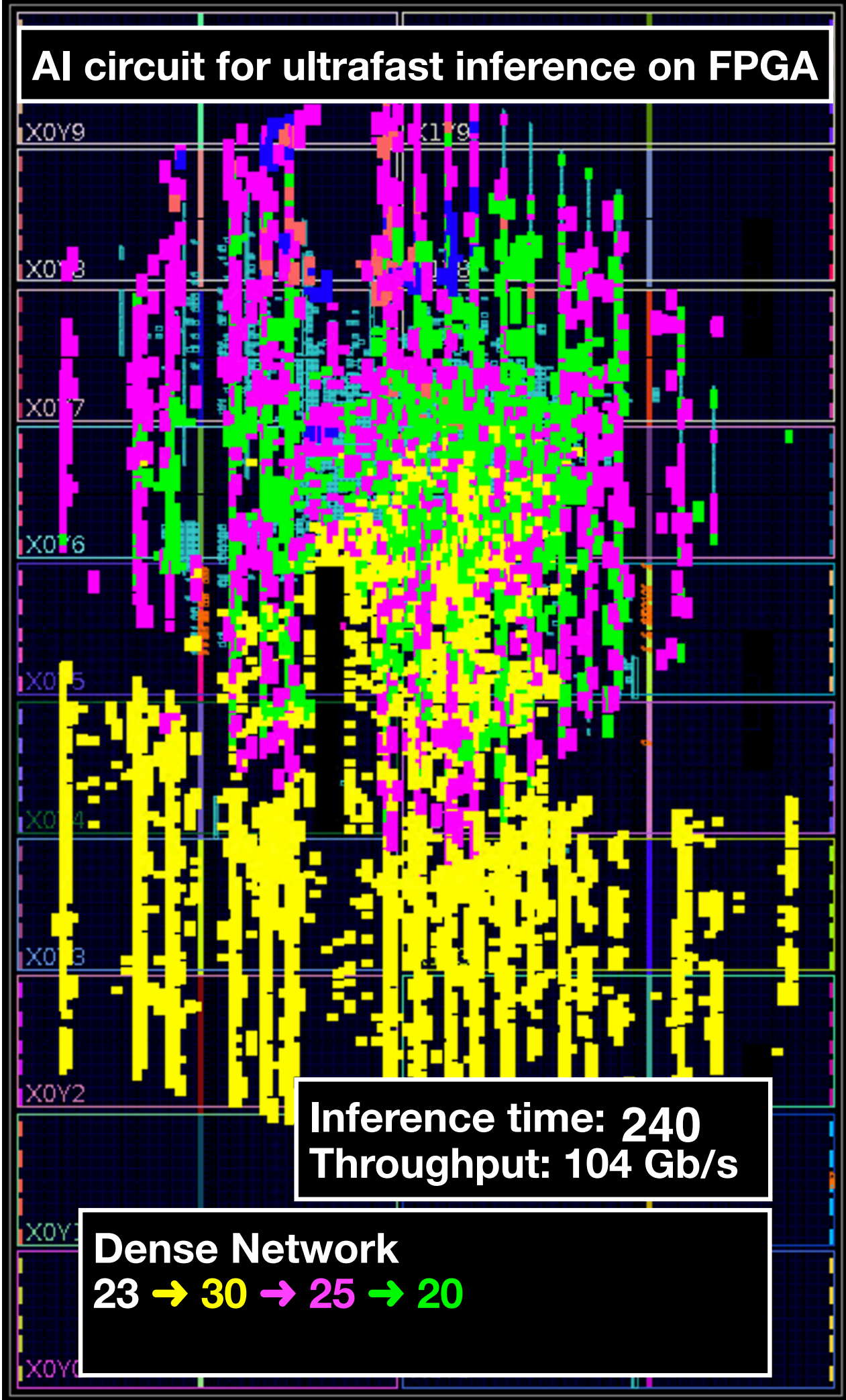


Application: Measure Muon p_T at 40 MHz



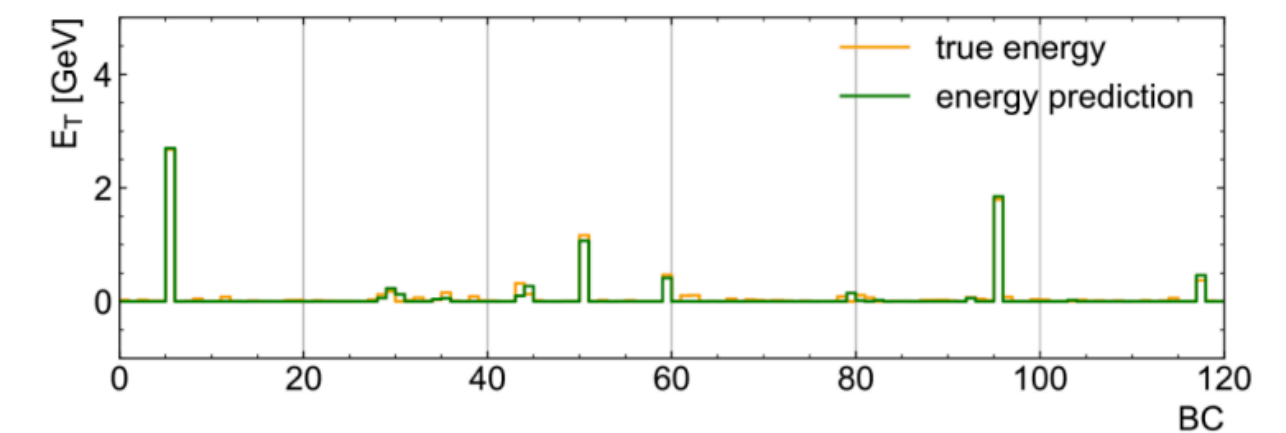
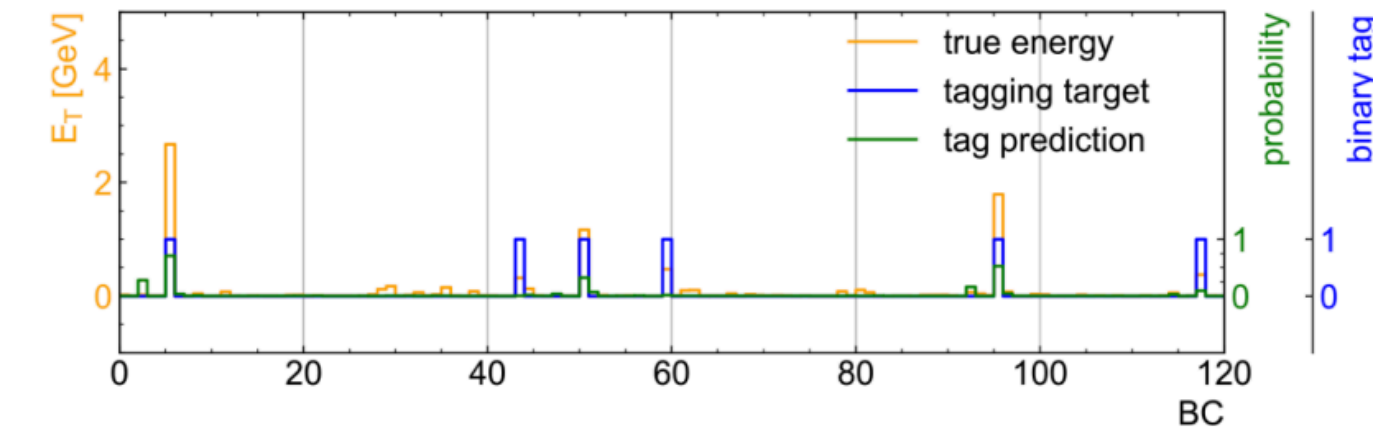
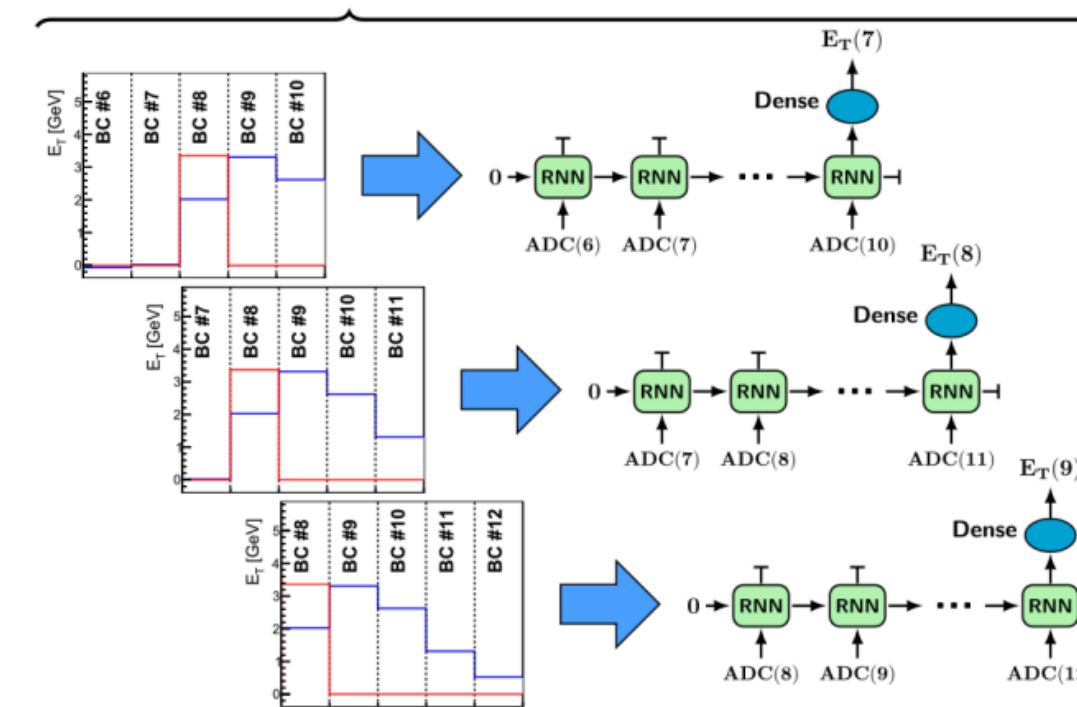
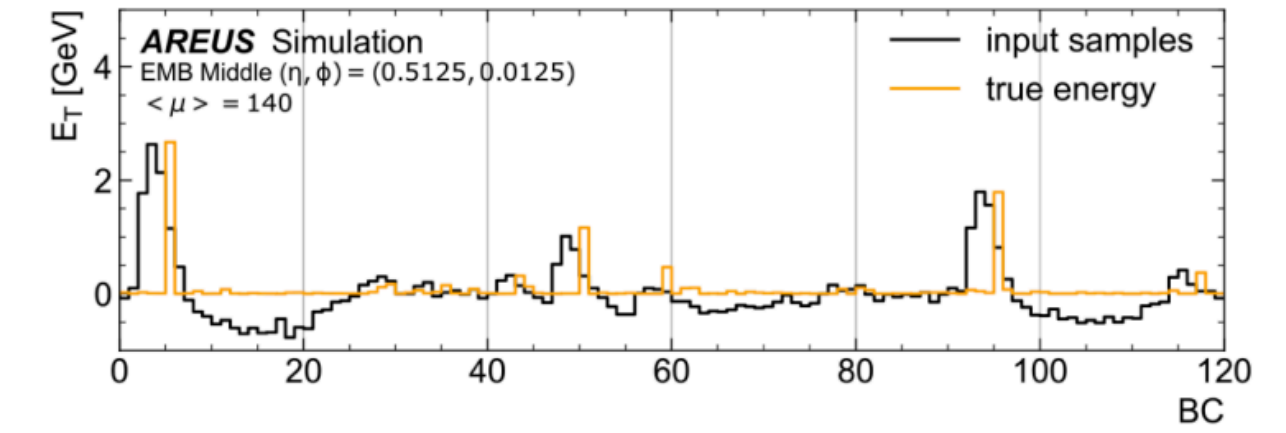
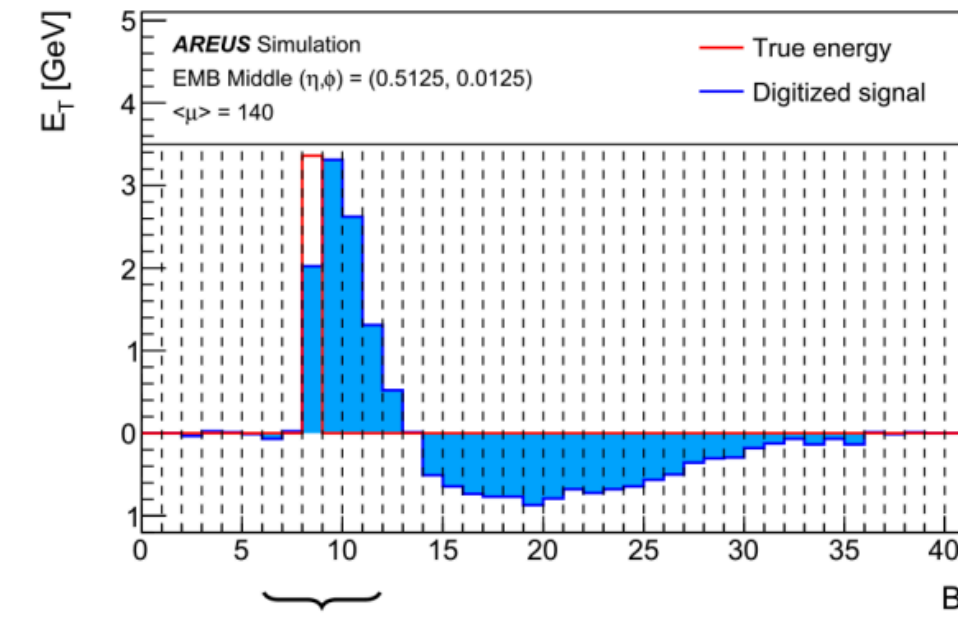
CMS-TDR-021

- NN measures muon momentum
- 3× reduction in the trigger rate for NN!
- Fits within L1 trigger latency (240 ns!) and FPGA resource requirements (less than 30%)



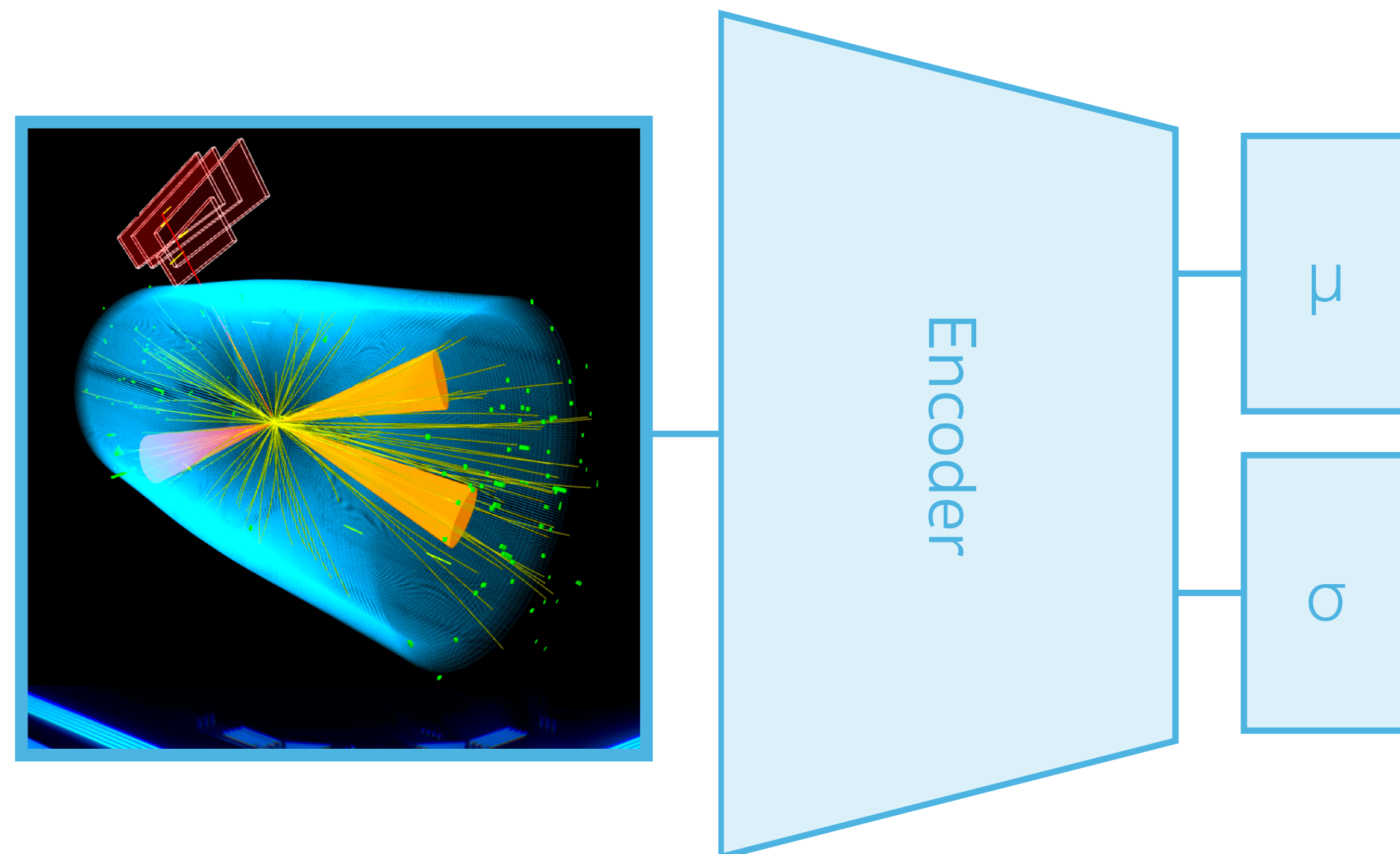
Convolutional and Recurrent Neural Networks

- real-time energy reconstruction of ATLAS LAr Calorimeter for Phase 2
- Up to around 600 calorimeter channels processed by on device
- 200 ns latency of predictions
- Implemented on Intel FPGAs (previous examples are all AMD)
 - Team contributed majorly to RNN and Intel implementations of hls4ml



Application: Anomaly Detection

- **Challenge:** if new physics has an unexpected signature that doesn't align with existing triggers, precious BSM events may be discarded at trigger level
- Can we use unsupervised algorithms to detect non-SM-like anomalies?
 - **Autoencoders (AEs):** compress input to a smaller dimensional latent space then decompress and calculate difference
 - **Variational autoencoders (VAEs):** model the latent space as a probability distribution; possible to detect anomalies purely with latent space variables



Key observation: Can build an anomaly score from the latent space of VAE directly! No need to run decoder!

$$R_z = \sum_i \frac{\mu_i^2}{\sigma_i^2}$$

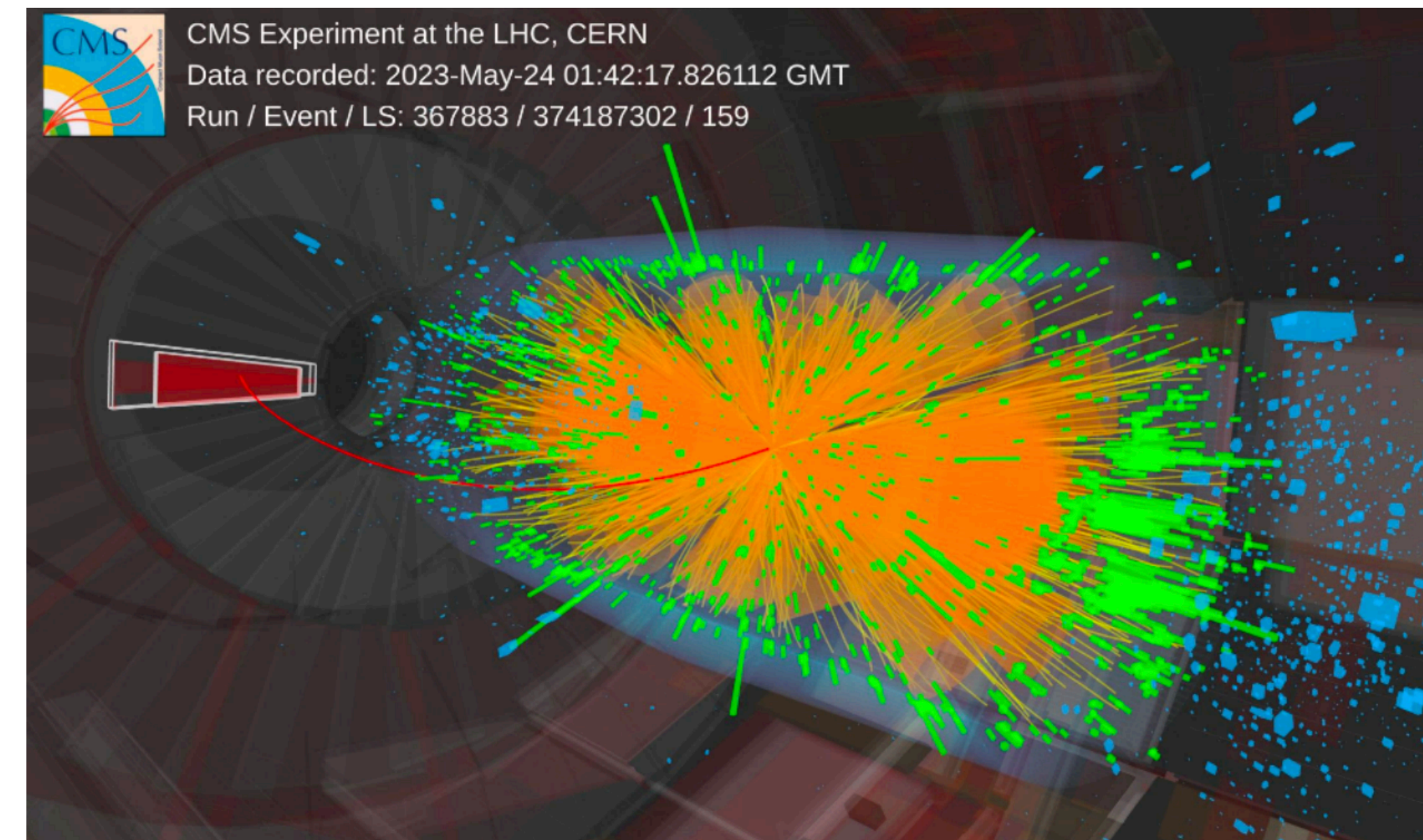
Application: CMS Anomaly Trigger

CMS has implemented a similar idea: **AXOL1TL**

- L1 Hardware implemented VAE-based AD trigger (based on <https://arxiv.org/abs/2108.03986>)
- Trained on 2018 zerobias data, ran in 2023 Global Trigger Test Crate
- CMS is also developing CICADA, a calorimeter only AD trigger

AXOL1TL

Event display of the
highest anomaly score

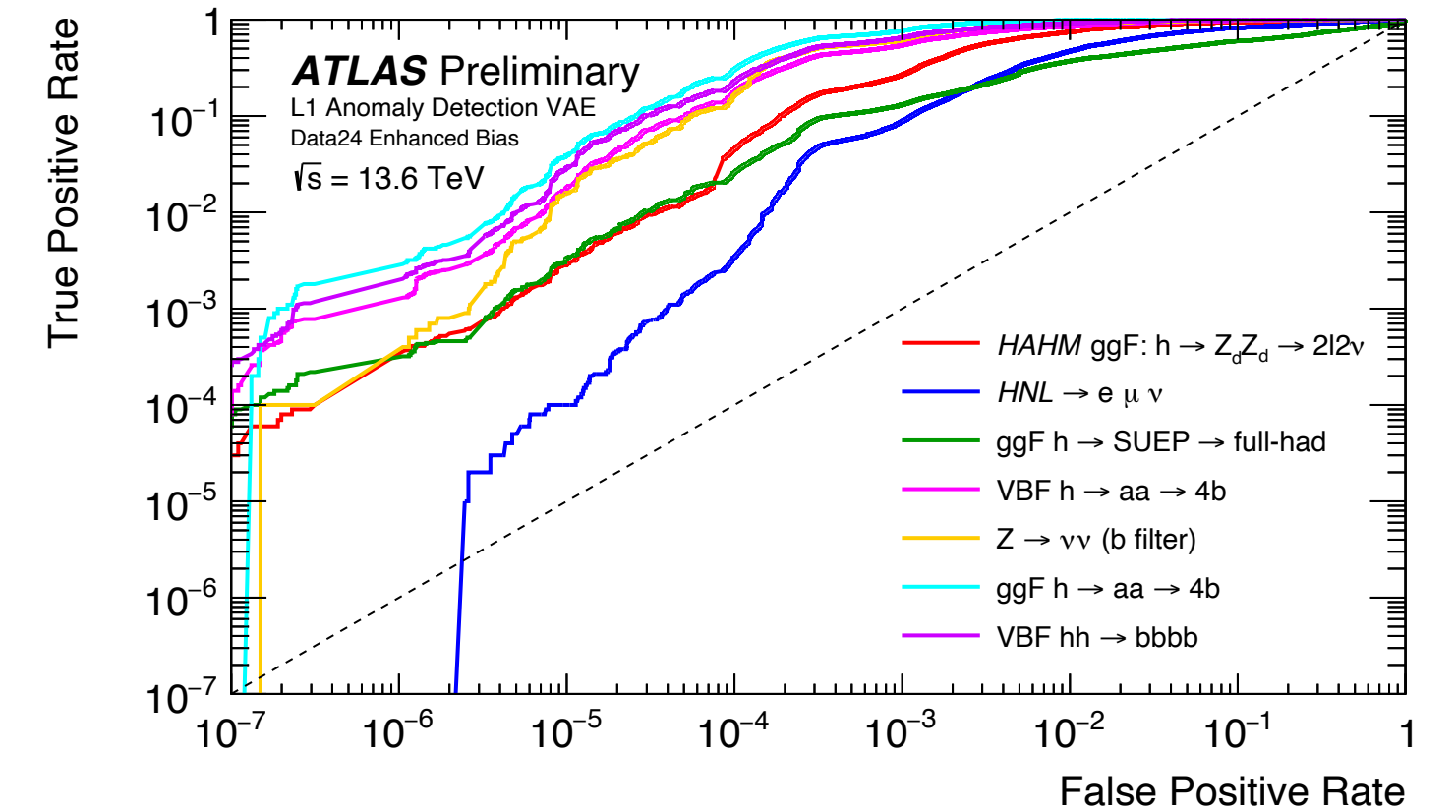


[CMS-DP-2023-079](#)

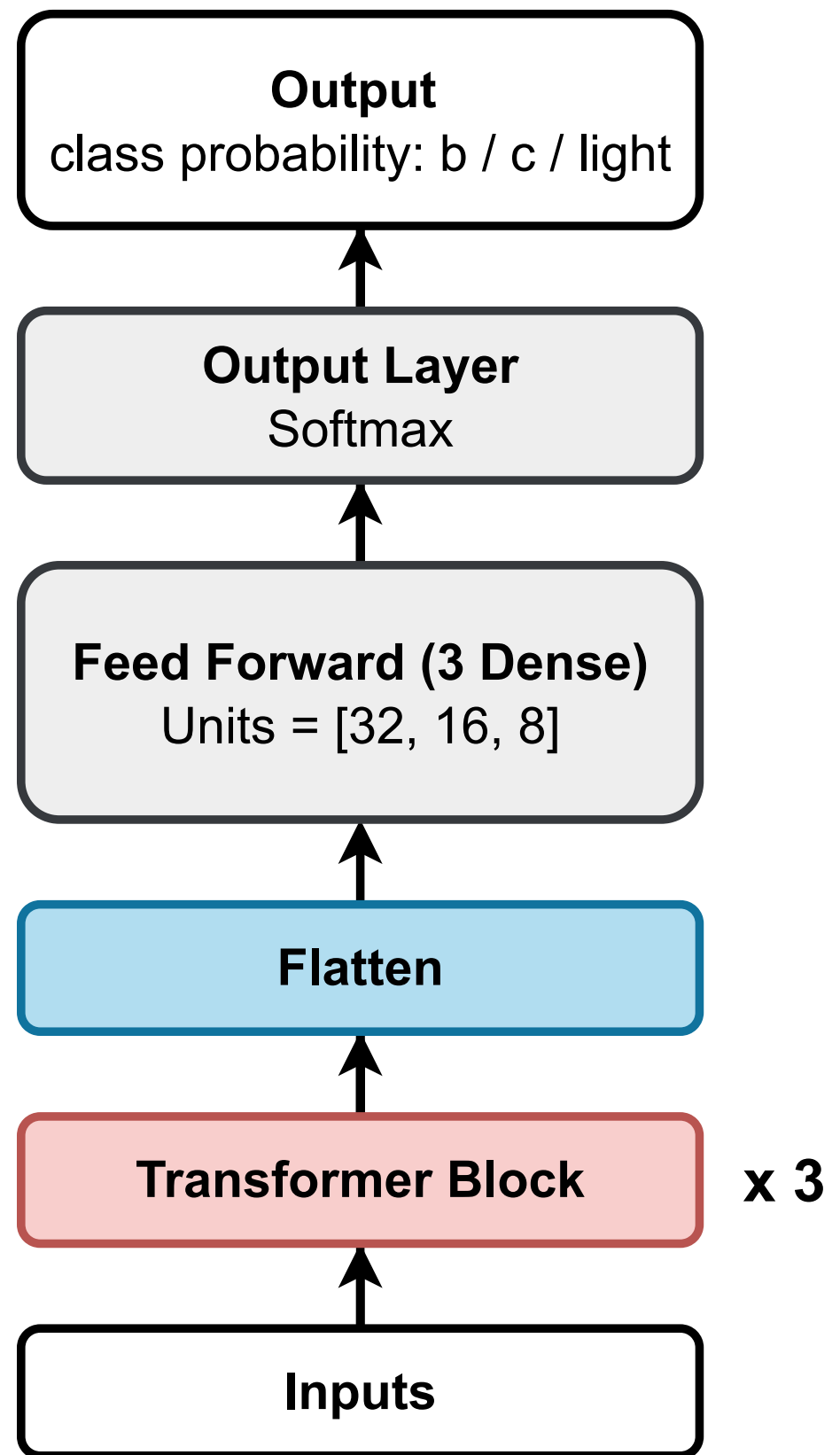
Application: ATLAS Anomaly Trigger

ATLAS has also implemented a similar idea: **GELATO**

- ATLAS is now employing a two-stage VAE-based pipeline to trigger anomalous events!
- Model events based on four-vectors of leading objects (**HLT: AE**)
- **L1: VAE + generative adversarial network (GAN)** to refine latent space; only encoder implemented on FPGA



Reuse and clk	Interval (cycle)	Latency (cycles)	Latency(time)
R1 (6.577 ns)	49	269	2.077 us
R2 (6.215 ns)	65	449	3.467 us
R4 (4.723 ns)	100	768	5.853 us



Ultra Fast Transformers on FPGAs for Particle Physics Experiments

Zhixing Jiang
University of Washington
Zhixij@uw.edu

Dennis Yin University of Washington lostecho@uw.edu
Elham E Khoda University of Washington ekhoda@uw.edu
Vladimir Loncar* Massachusetts Institute of Technology vloncar@mit.edu

Ekaterina Govorkova Massachusetts Institute of Technology katyag@mit.edu
Eric Moreno Massachusetts Institute of Technology Emoreno@mit.edu

Philip Harris Massachusetts Institute of Technology pcharris@mit.edu
Scott Hauck University of Washington hauck@uw.edu
Shih-Chieh Hsu University of Washington schsu@uw.edu

arXiv:2402.01047v1 [cs.LG] 1 Feb 2024

Abstract

This work introduces a highly efficient implementation of the transformer architecture on a Field-Programmable Gate Array (FPGA) by using the hls4ml tool. Given the demonstrated effectiveness of transformer models in addressing a wide range of problems, their application in experimental triggers within particle physics becomes a subject of significant interest. In this work, we have implemented critical components of a transformer model, such as multi-head attention and softmax layers. To evaluate the effectiveness of our implementation, we have focused on a particle physics jet flavor tagging problem, employing a public dataset. We recorded latency under 2 μ s on the Xilinx UltraScale+ FPGA, which is compatible with hardware trigger requirements at the CERN Large Hadron Collider experiments.

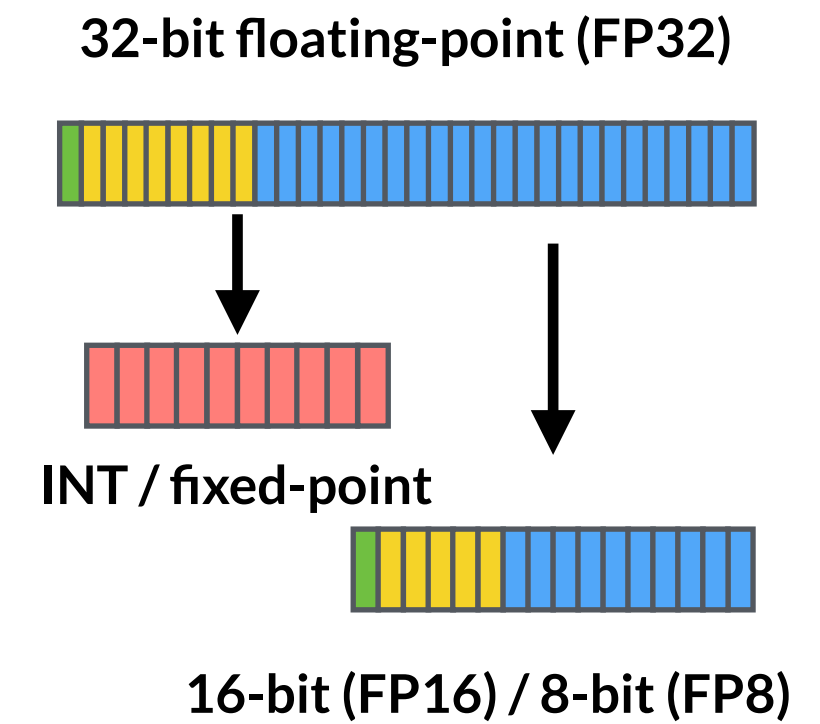
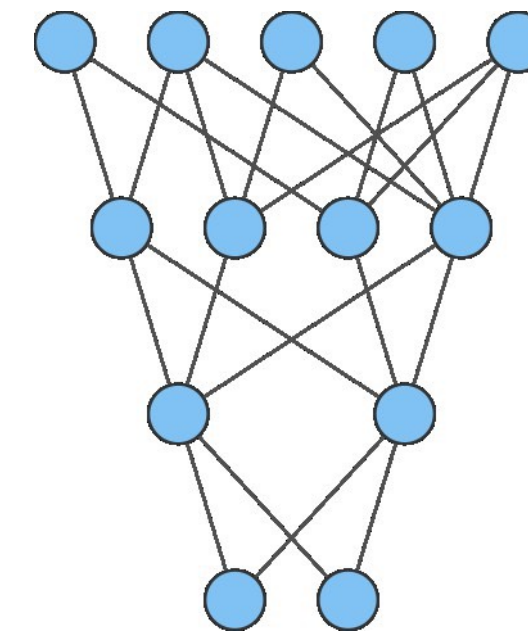
1 Introduction

Accelerated Machine Learning (ML) inference is necessary to run the algorithms in the online event selection systems of the particle physics experiments. Due to the extremely high particle collision

Observed Inference Latency ~ 2-6 μ s

Summary

- Efficient model design is becoming more important as the models are getting bigger
- Several tricks to design efficient ML inference
 - Some popular method: Pruning and Quantization
 - Need to optimize pruning and quantization strategy for satisfactory results
 - *Another effective technique: Knowledge Distillation*
- ML-based algorithms are getting popular for experimental trigger applications
- Efficient ML techniques are crucial for real-time inference

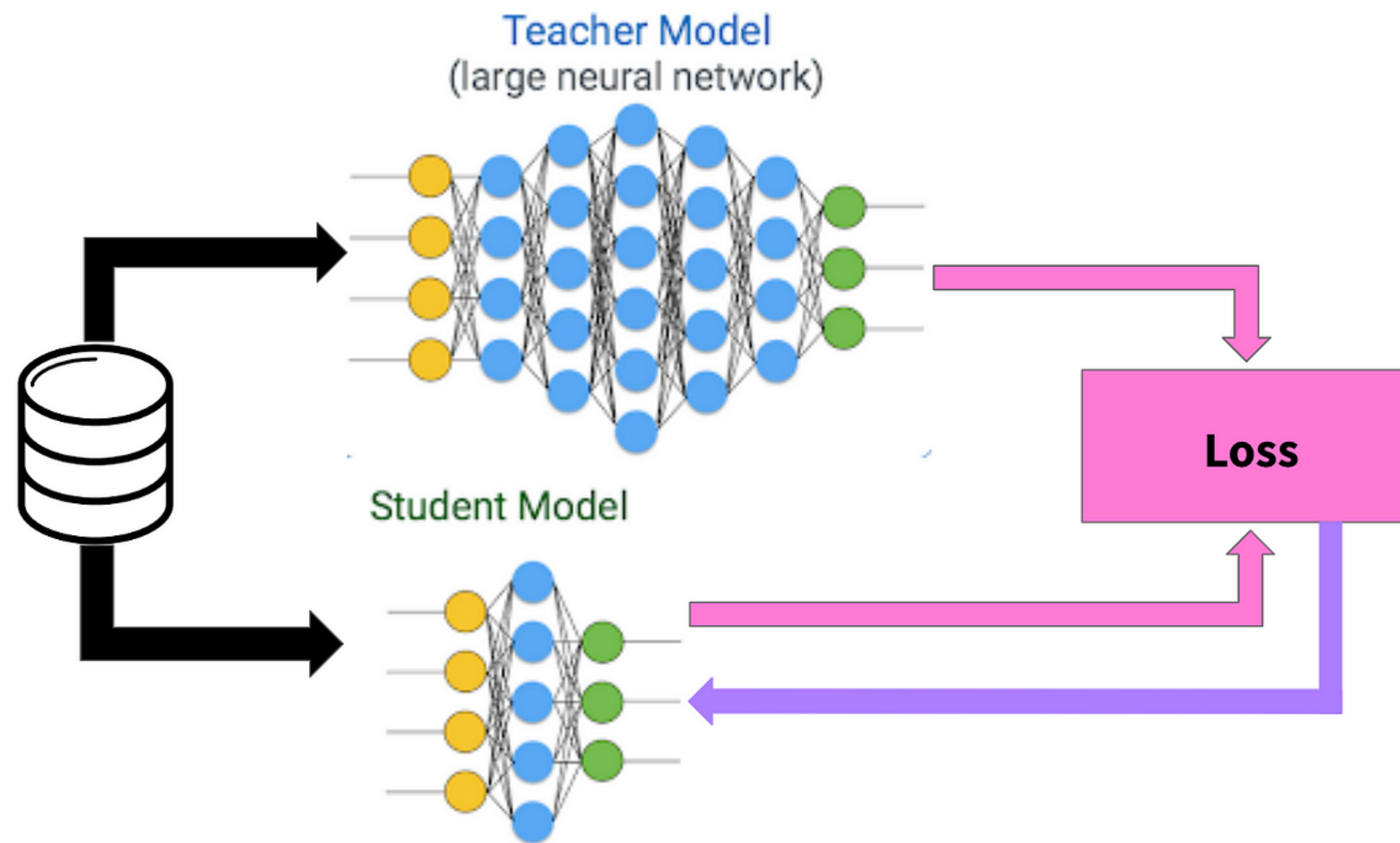


End of the Lesson

Thank you for your attention and participation

Knowledge Distillation

One Example: particle physics application



Efficient and Robust Jet Tagging at the LHC with Knowledge Distillation

Ryan Liu
University of California, Berkeley
Berkeley, CA 94720

Abhijith Gandrakota
Fermi National Accelerator Laboratory
Batavia, IL 60510

Jennifer Ngadiuba
Fermi National Accelerator Laboratory
Batavia, IL 60510

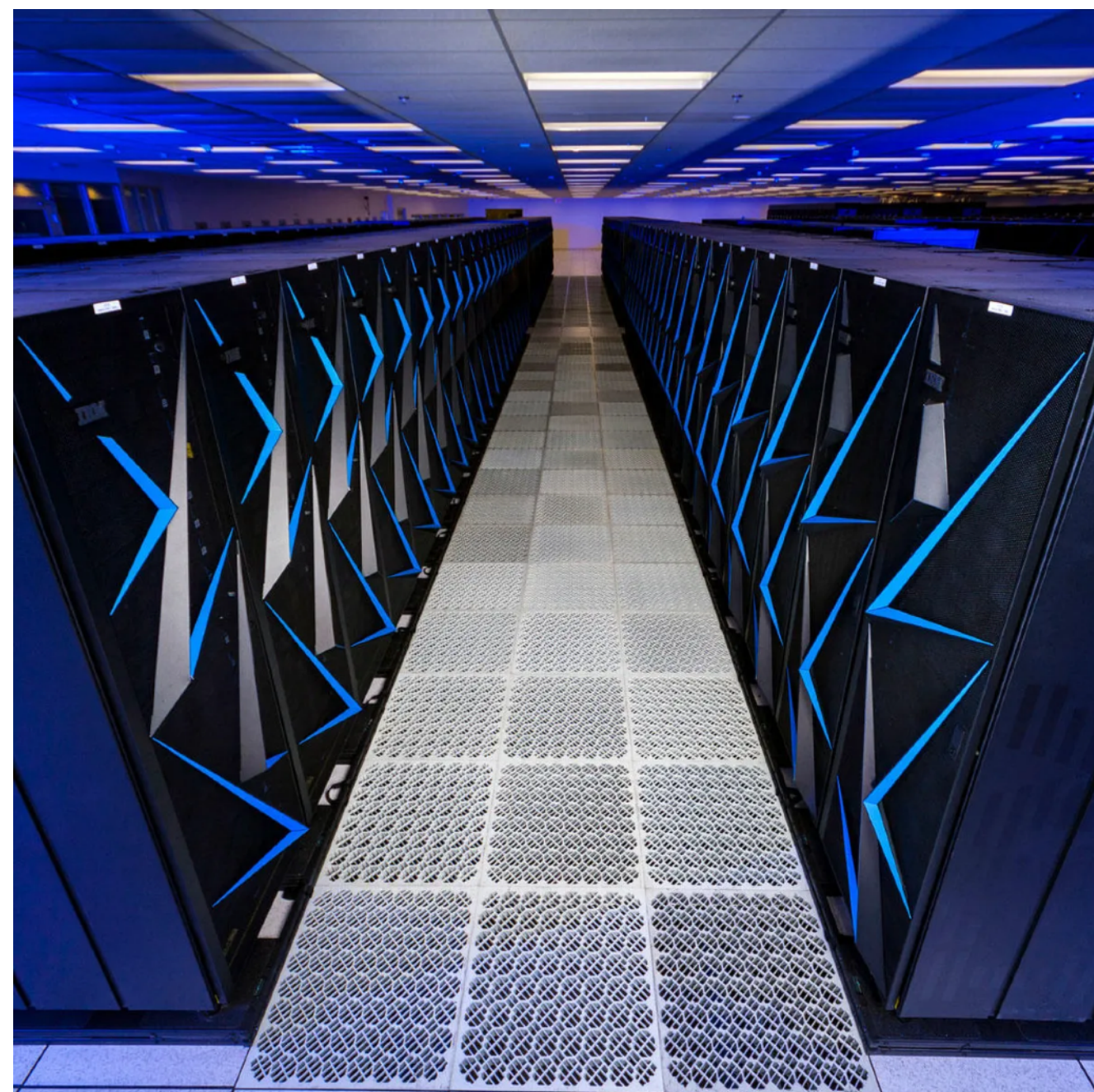
Maria Spiropulu
California Institute of Technology
Pasadena, CA 91125

Jean-Roch Vlimant
California Institute of Technology
Pasadena, CA 91125

Thank You!

Extra Slides

GPT-3 Training and Inference



GPT-3 Training

- 285,000 CPUs
- 10,000 GPUs
- 400 Gbits/sec network
- Several weeks

Inference: Asking Question to ChatGPT



You

How many GPUs are you using currently?



ChatGPT

As an AI language model, I don't have access to real-time information about my current computational resources or the infrastructure being used to host me. My availability and performance are managed by OpenAI's servers, which may utilize GPUs or other computational resources based on demand and system configuration. Therefore, I cannot provide information on the number of GPUs currently being used.

GPT-3: 10x larger than max memory in a single Nvidia A100 GPU

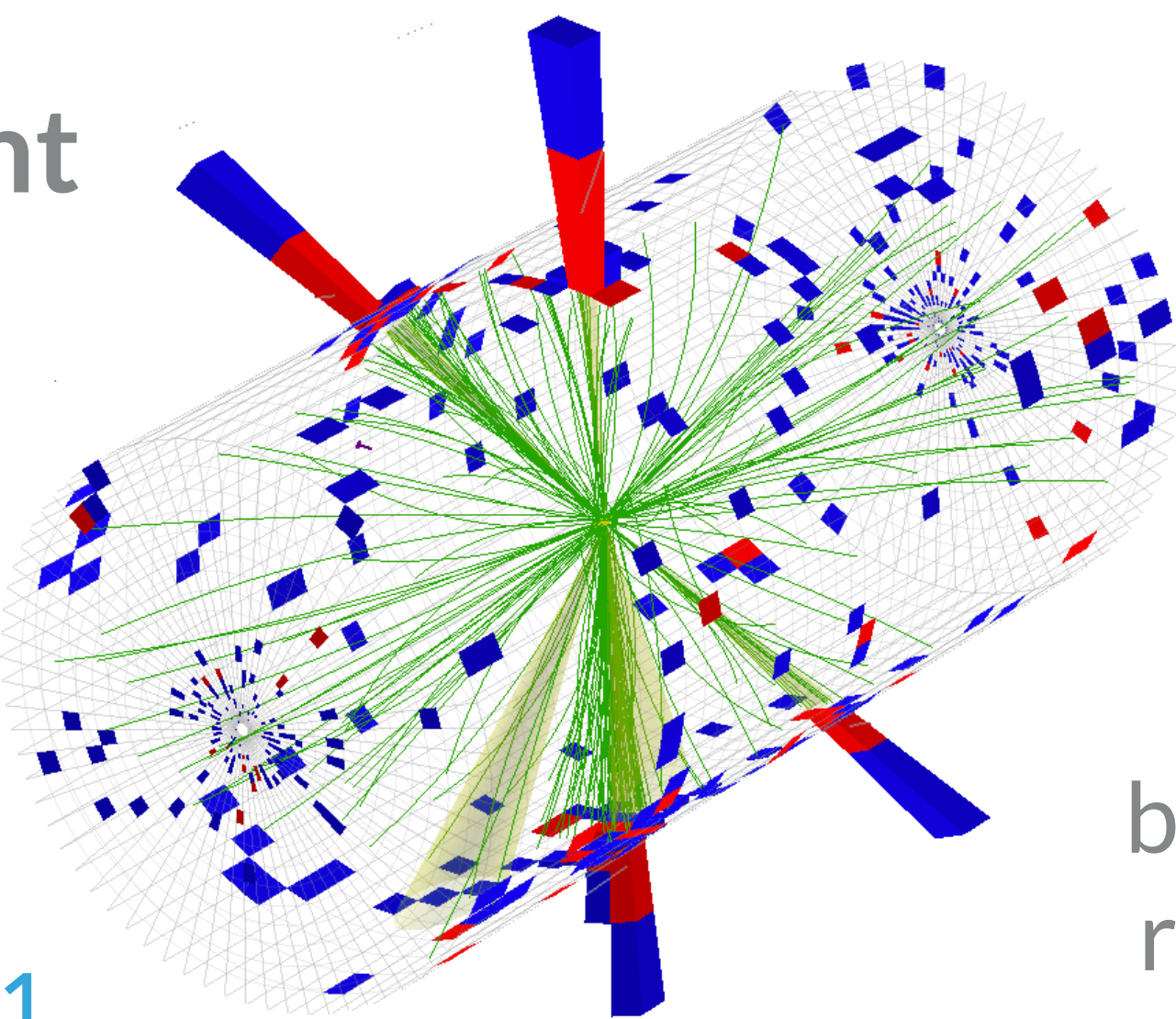
Millions of queries daily

1 GWh each day \approx 33,000 U.S. households

Simplified HL-LHC Trigger

- Single/double/triple muons/electrons
- Photons
- Taus
- Hadronic
- Missing transverse energy
- “Cross” triggers (not shown)

4-jet event

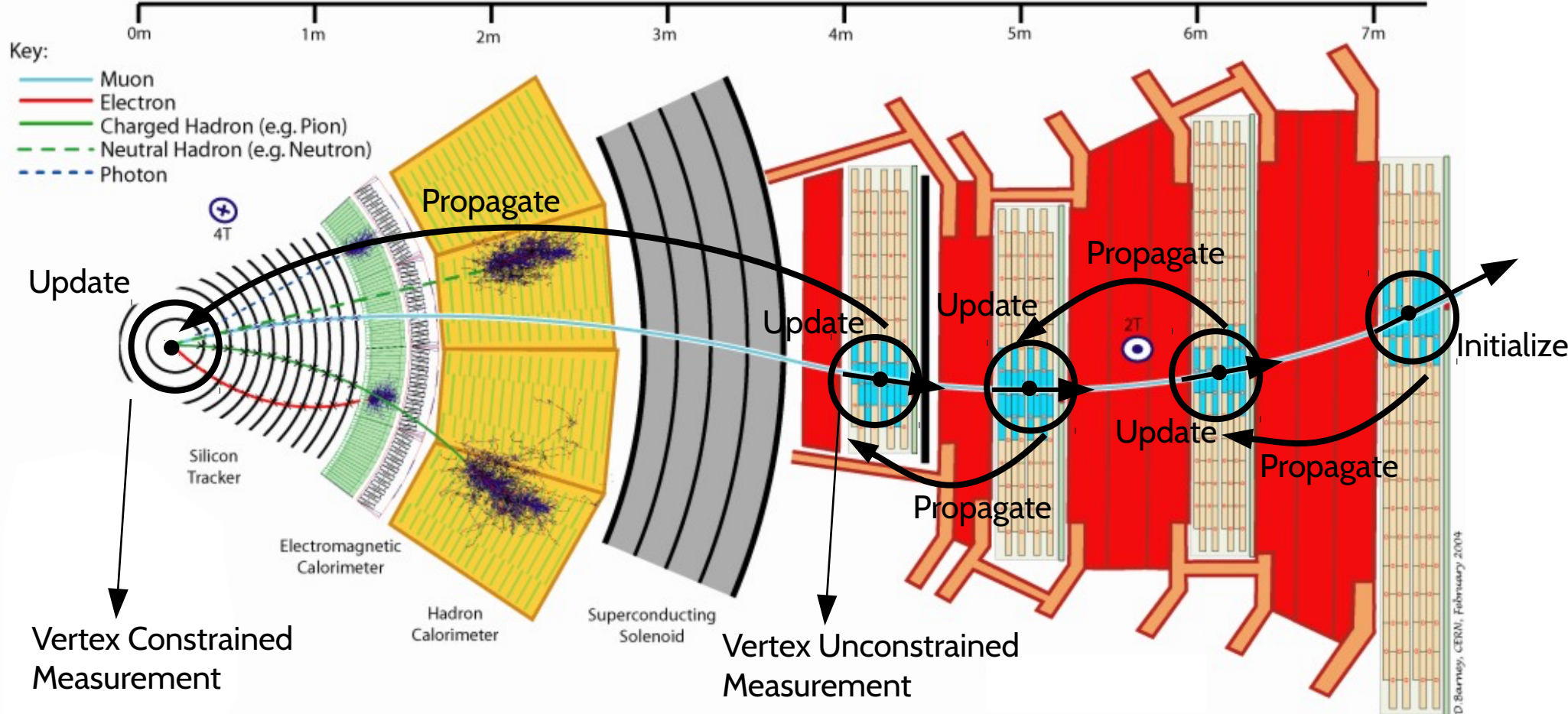
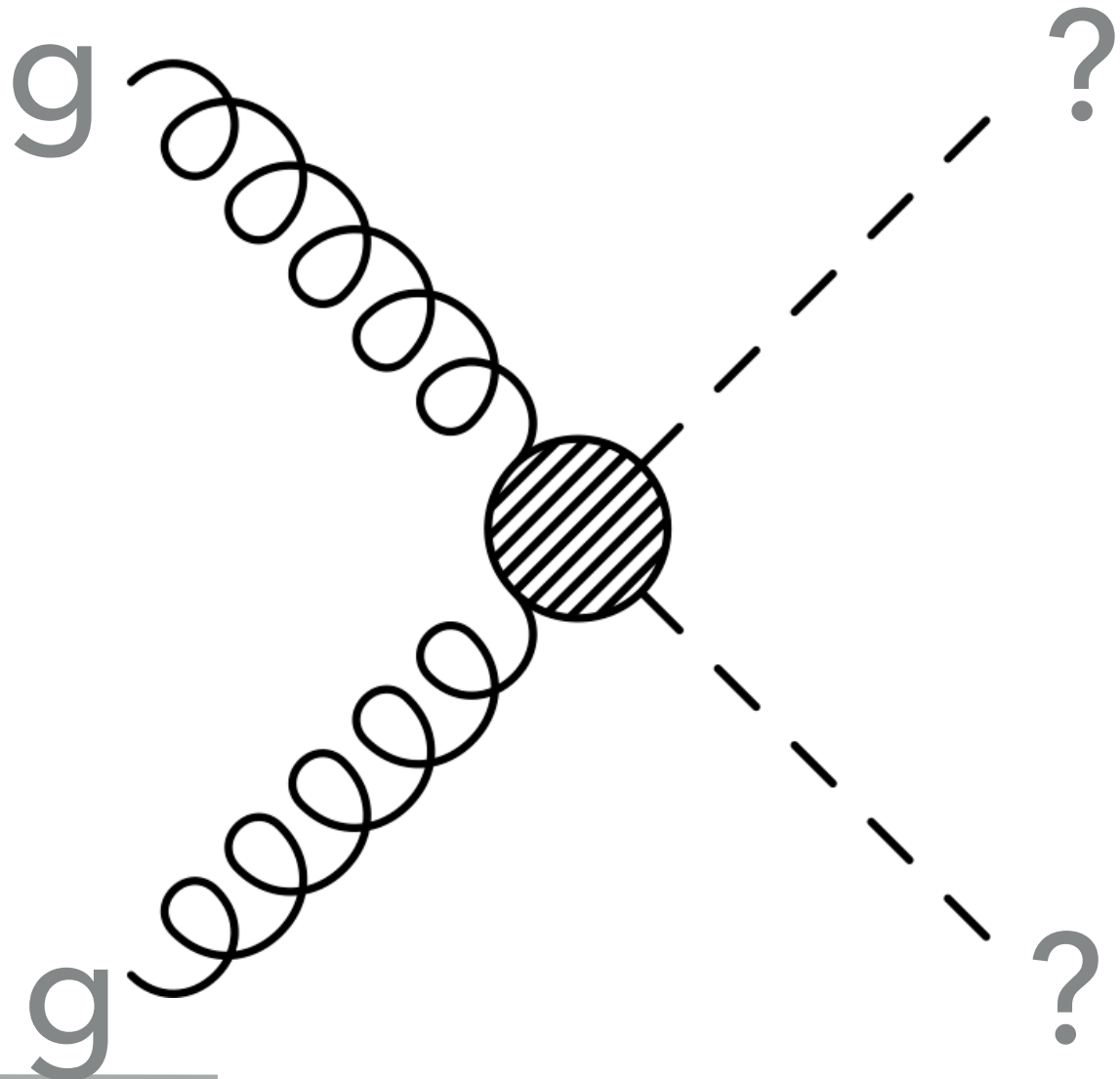
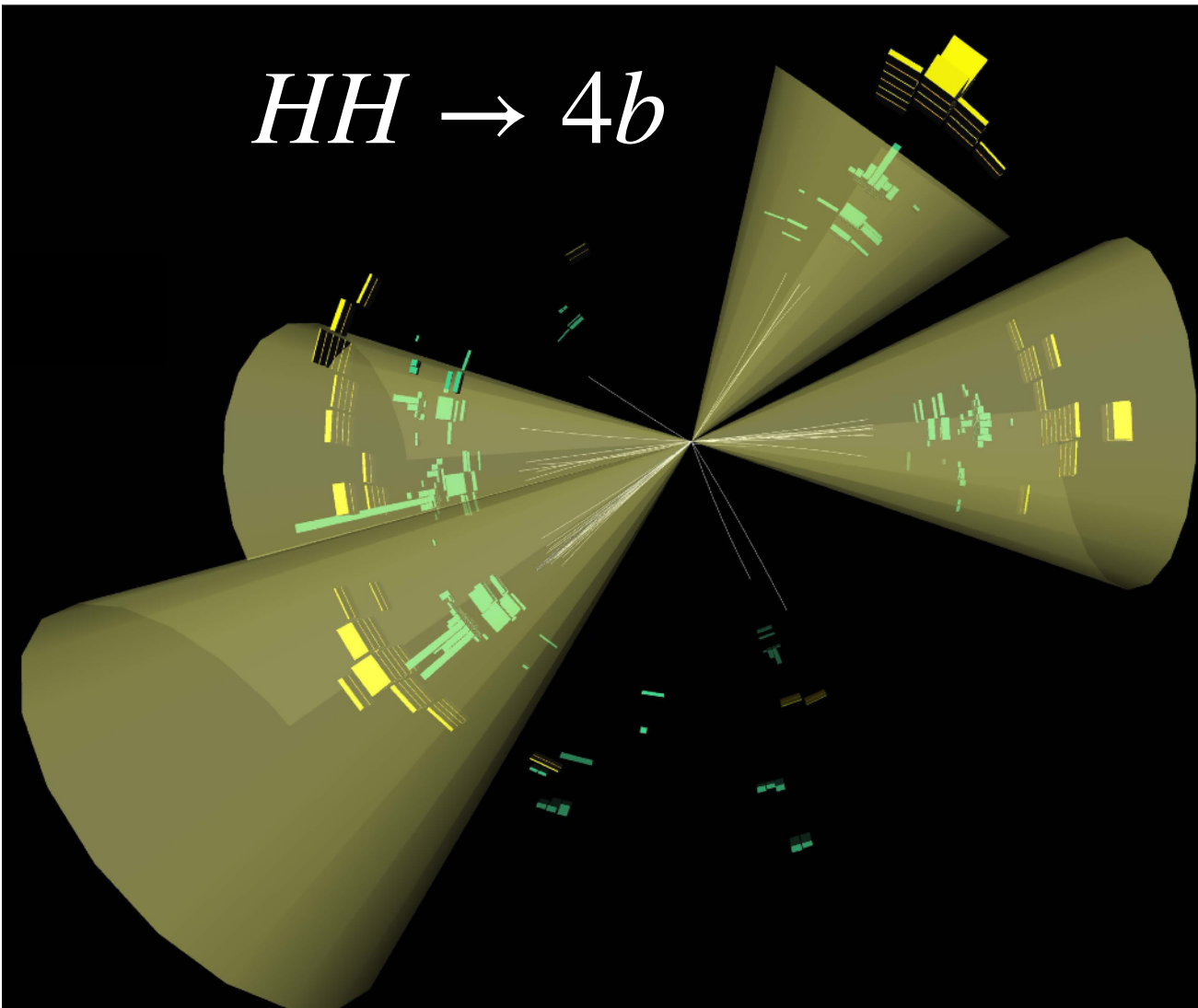
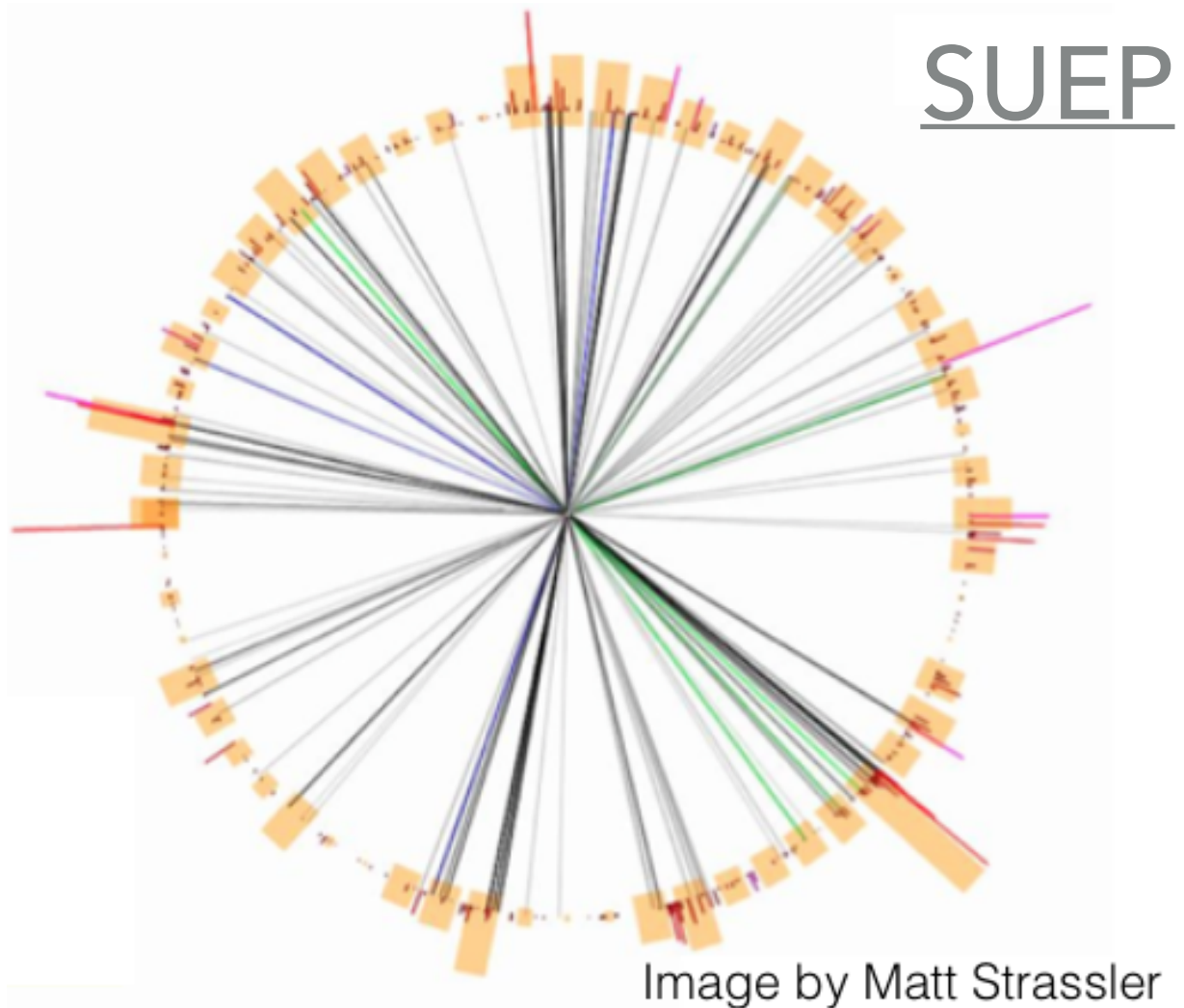


Thresholds set by backgrounds, limited resolution @ L1, and rate budget

Trigger	Threshold [GeV]
1 μ	22
2 μ	15, 7
3 μ	5, 3, 3
1 e	36
2 e	25, 12
1 γ	36
2 γ	22, 12
1 τ	150
2 τ	90, 90
1 jet	180
2 jet	112, 112
H_T	450
4 jet + H_T	75, 55, 40, 40, 400
p_T^{miss}	200

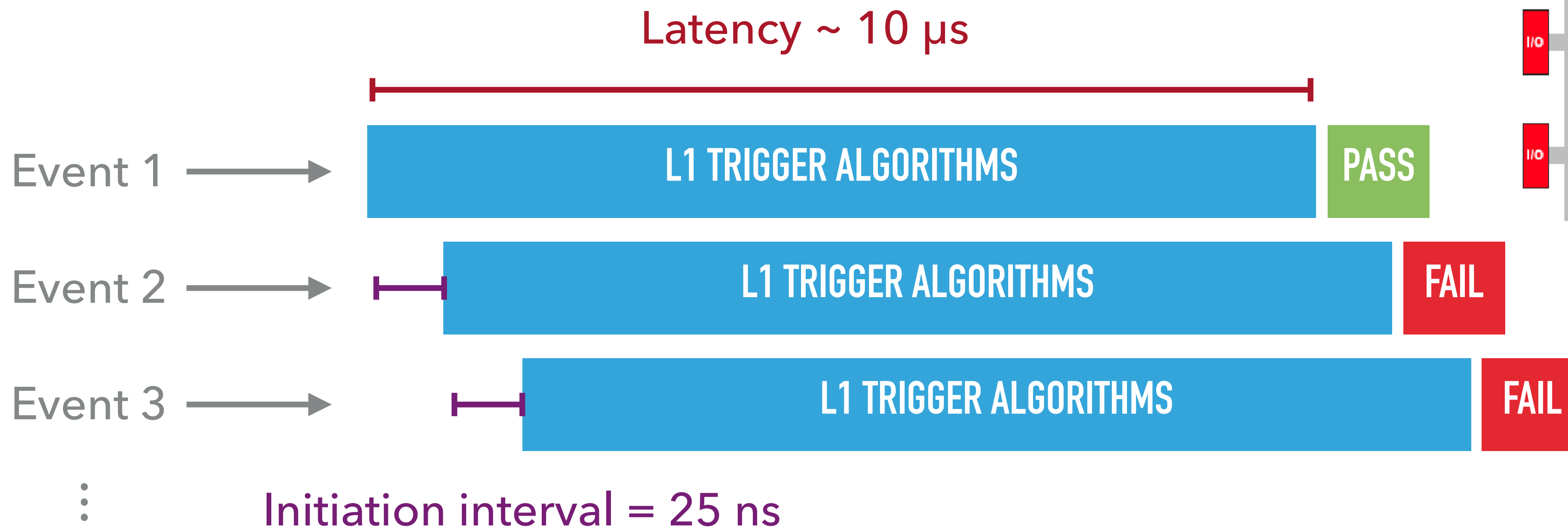
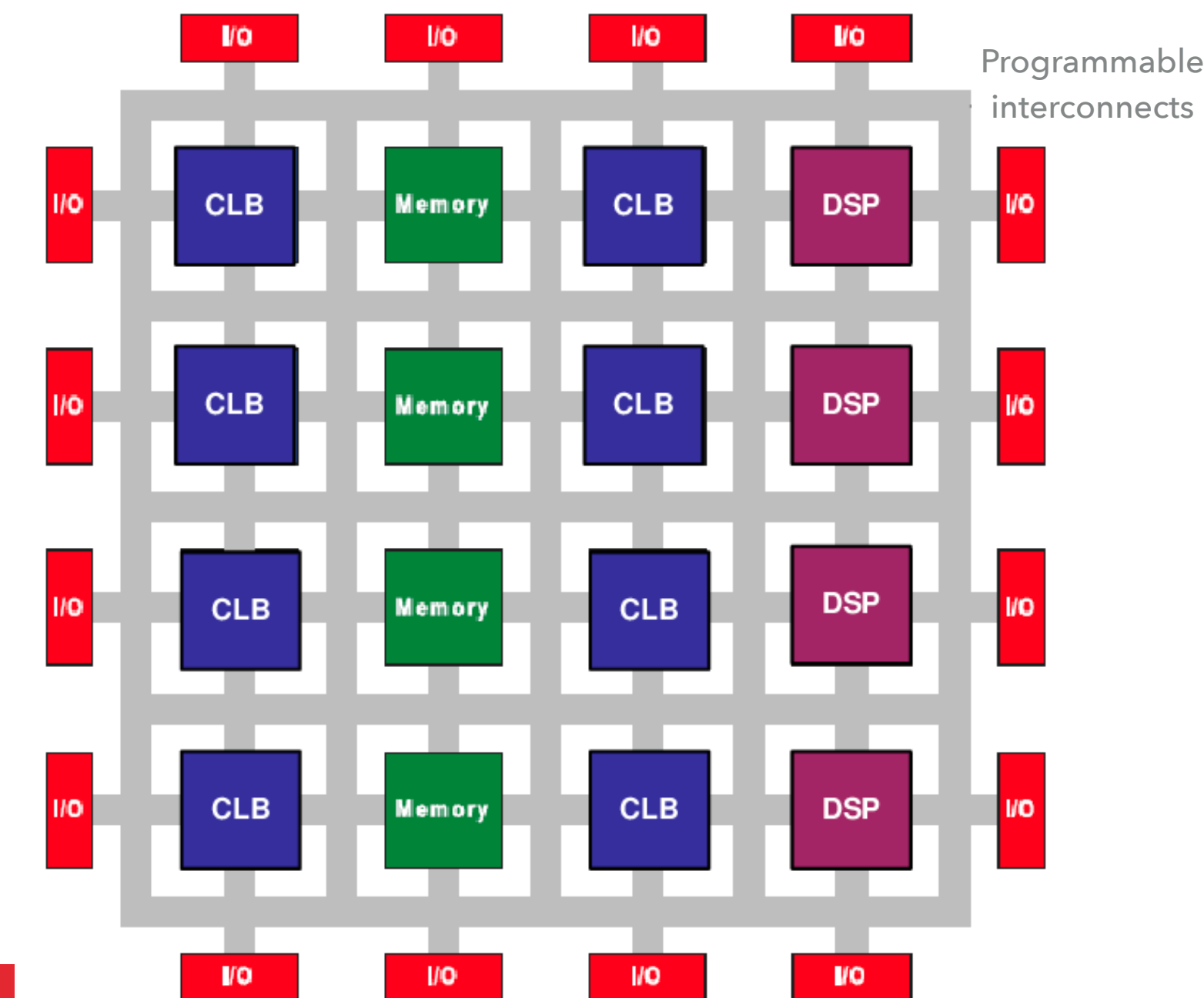
What could be missing?

- How can we trigger on more complex low-energy hadronic signatures? Long-lived/displaced particles?
- What if we don't know exactly what to look for?
- What if our signatures require complex multivariate algorithms (e.g. b tagging)?
- How can we improve on our traditional (often slow) reconstruction algorithms?



What makes this Hard?

- Reconstruct all events and reject 98% of them in $\sim 10 \mu\text{s}$
 - Algorithms have to be $< 1 \mu\text{s}$ and process new events every $(25 \text{ ns}) \times N_{tmux}$
- Latency necessitates all **FPGA** design
 - Algorithms have to fit on < 1 FPGA
- How can we satisfy these constraints?



Magnitude-based Pruning

Row-wise pruning using L_n -norm

We can define L_n norm as: $\|W^{(S)}\| = \left(\sum_{i \in S} |w_i|^n \right)^{\frac{1}{n}}$

W^S = a structural set of parameters

□ Pruned
 ■ Preserved

2	-4
7	-1

Original Weight

Row-wise
 L_n -norm

$(2 ^2 + -4 ^2)^{1/n}$ $20^{1/p}$
$(7 ^p + -1 ^p)^{1/n}$ $= 50^{1/p}$

Importance

20 ^{1/n}
50 ^{1/n}

-	-
-	-

Pruned Weight

Neural Network Pruning: Formulation

Neural network model $f(x; W)$ Remove least important connections such that

Pruning $\arg \min_{W_p} L(x; W_p) \quad \|W_p\|_0 < N$

Calculate

subject to sparsity ratio \rightarrow least significant / important connections /

$$\text{Sparsity} = \frac{\# \text{ of non-zero weights}}{\text{Total \# of weights}}$$

\odot

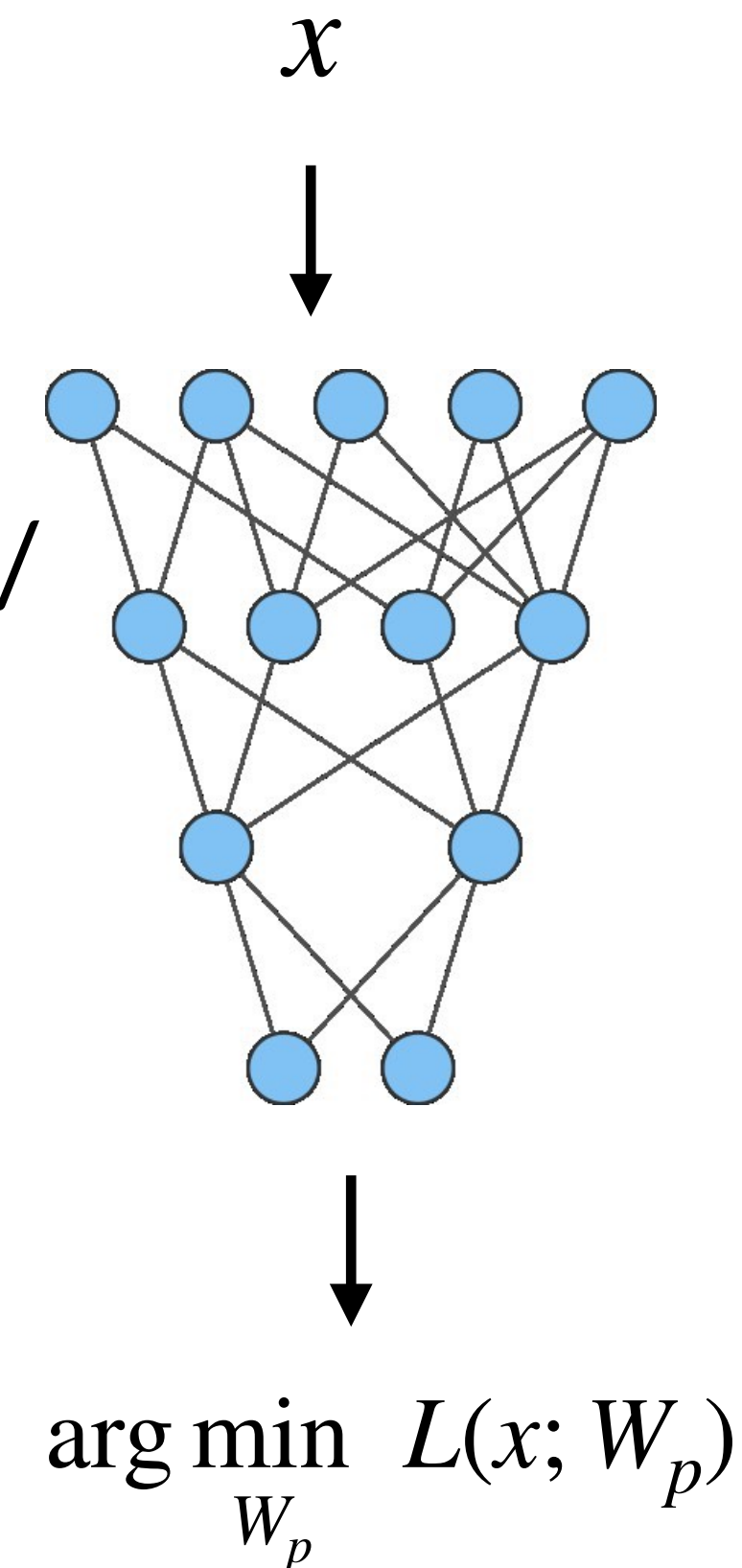
Element-wise
product operator

Pruned model

$$f(x; M \odot W'_p)$$

$$M \in \{0,1\}^{|W|}$$

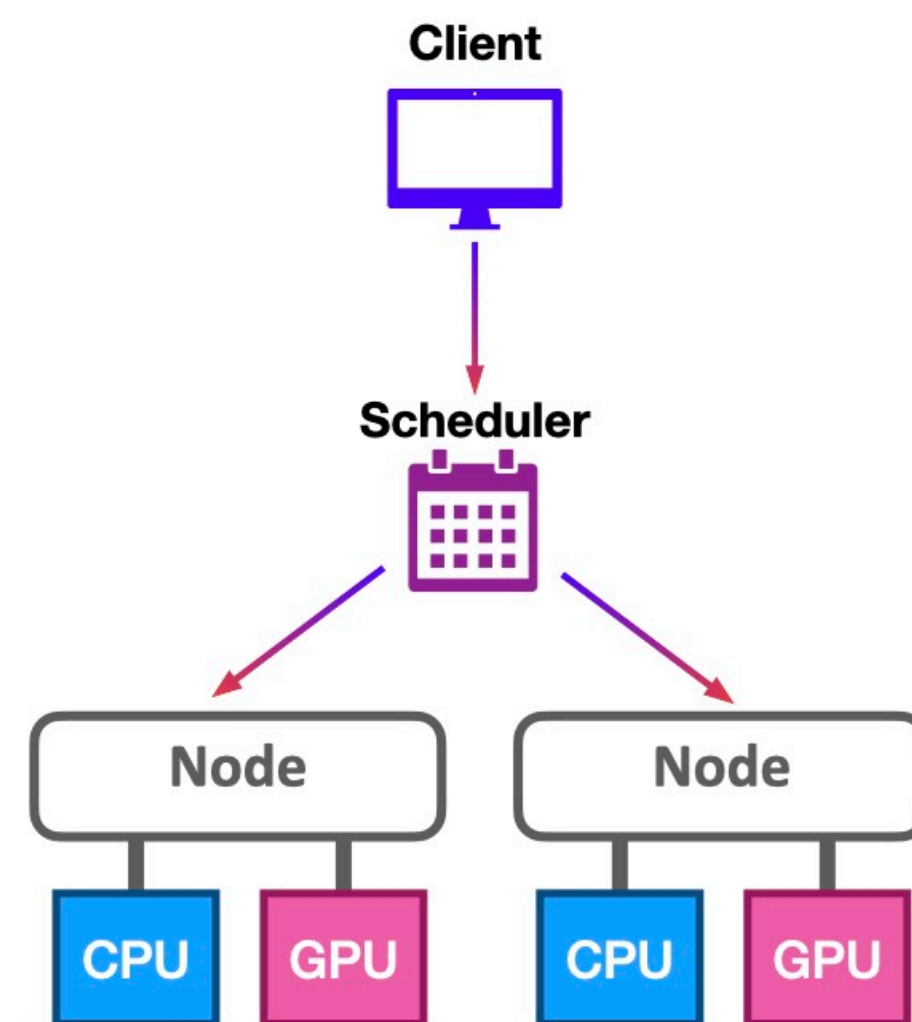
M is binary mask
tensor that fixes
certain parameters
to 0



- Support for different type of hardware is becoming important

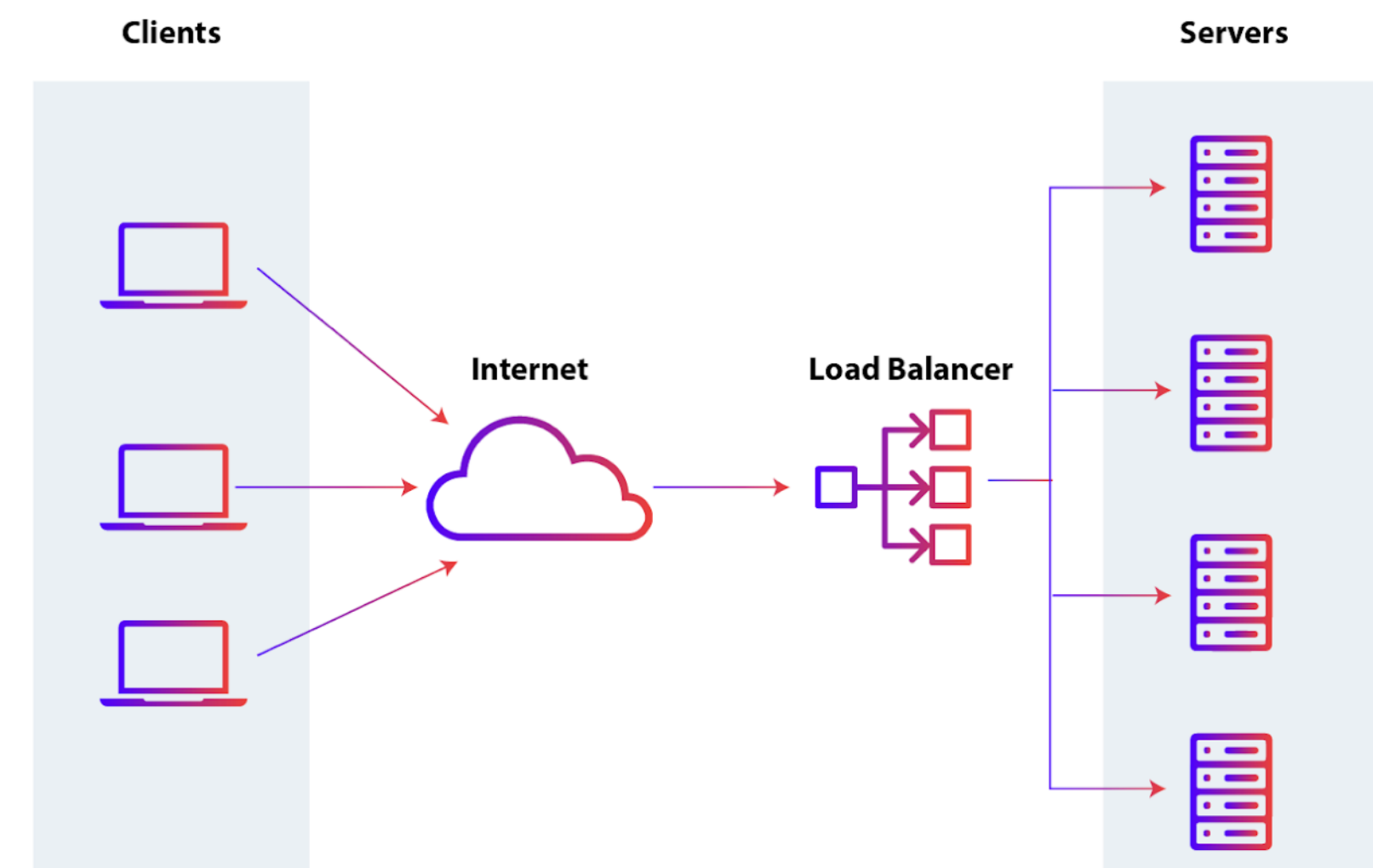
Direct Connection

CPUs and GPUs are connected



As a Service

No need to have a local GPU



- Simple support for mixed hardware
- Scalable
- Throughput optimization for multiple-core

SoftMax Optimization

General approach:

each SoftMax output S_i requires the calculation of the exponent of the difference between z_j and z_i , summed over all elements, and then inverted

$$S_i = \left(\sum_{j=1}^k (e^{(z_j - z_i)}) \right)^{-1}$$

Modified approach:

Stage 1:

Element-wise exponent computation

Stage 2:

Sum of all the exponents

-> Inverted using inverse look-up table

-> stored in register

Stage 3:

Element-wise multiplication

