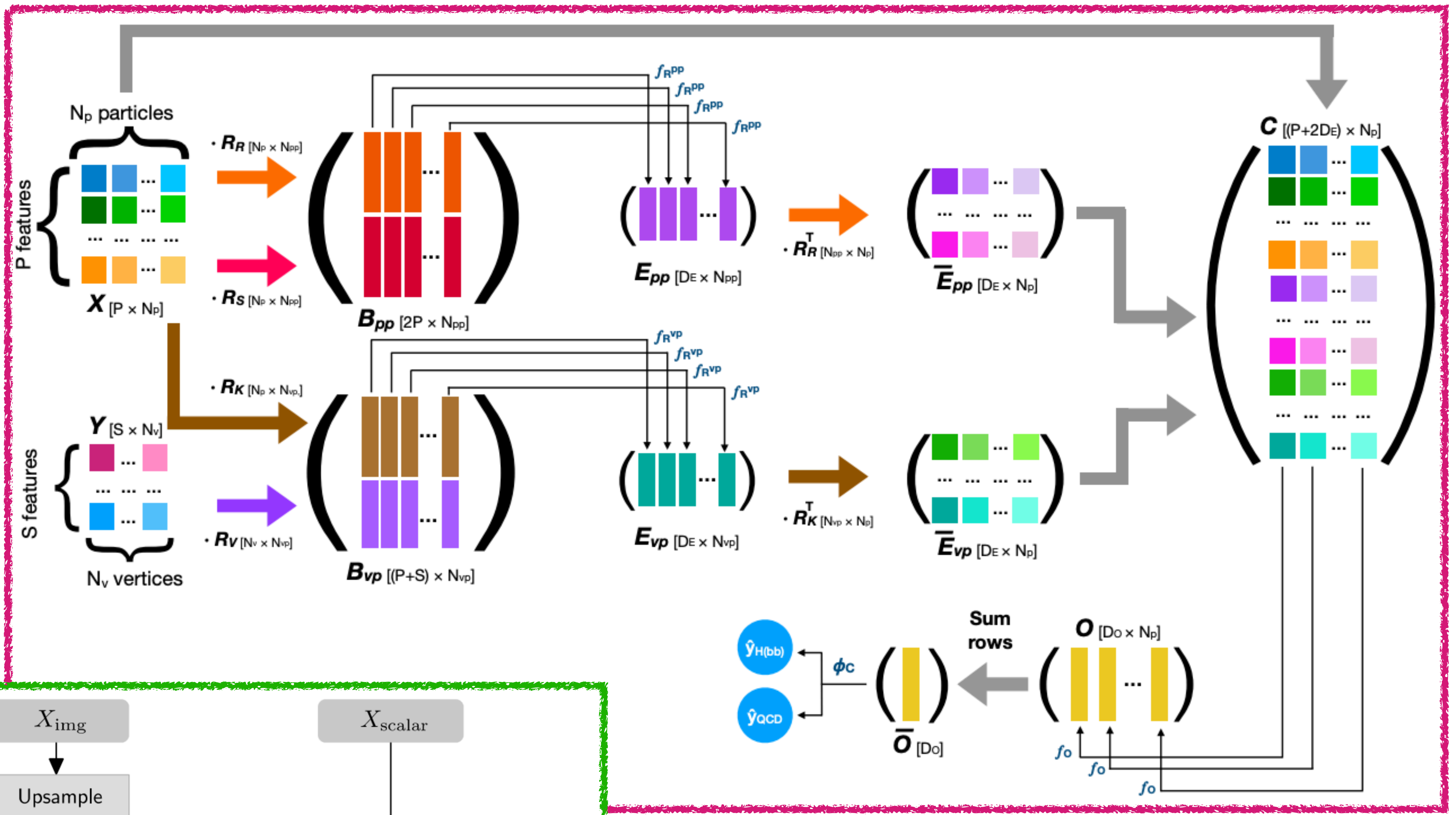
Overview of AI in HEP Readout

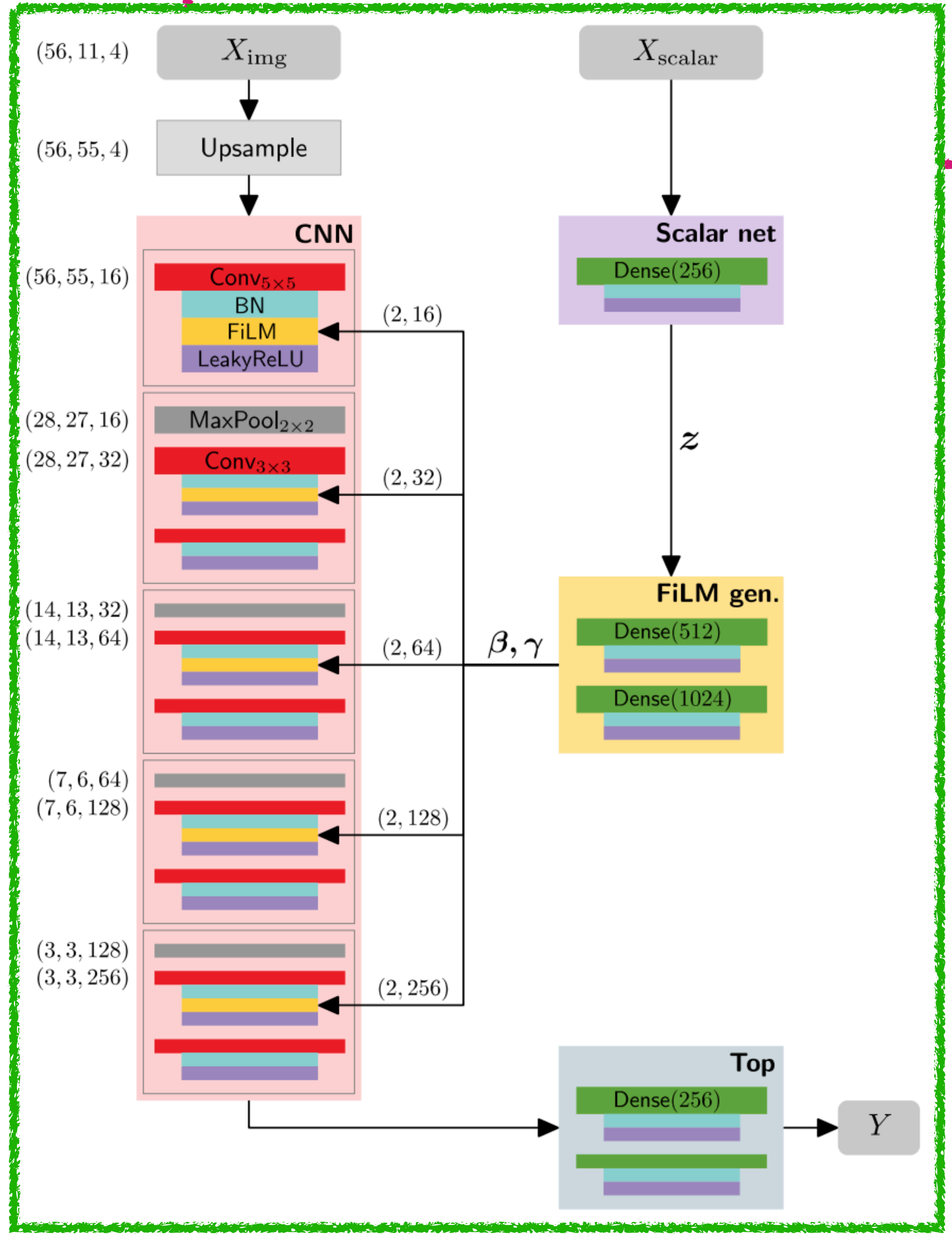
AI4EIC-Exp

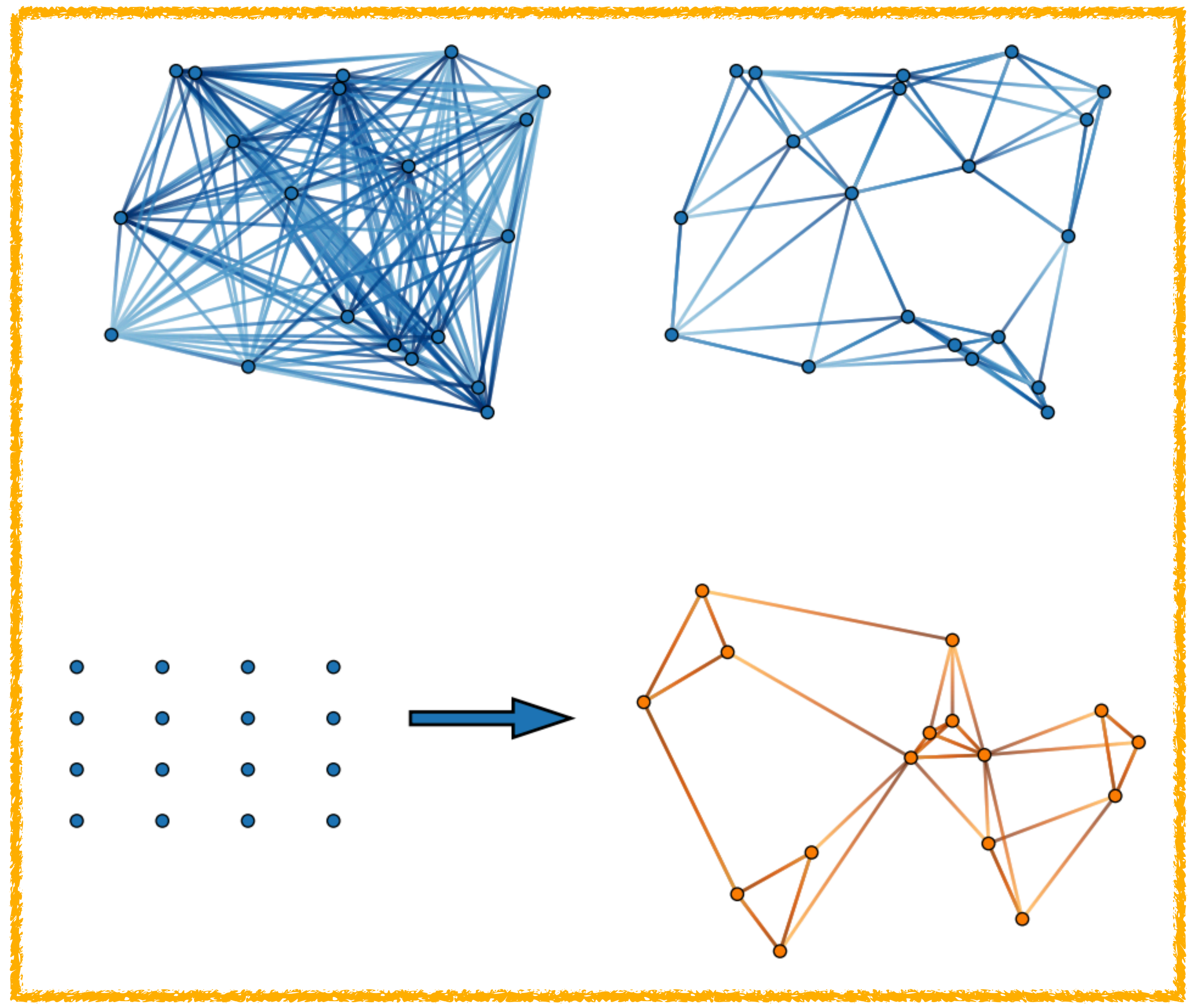
Dylan Rankin [MIT] - September 9th, 2021

Introduction

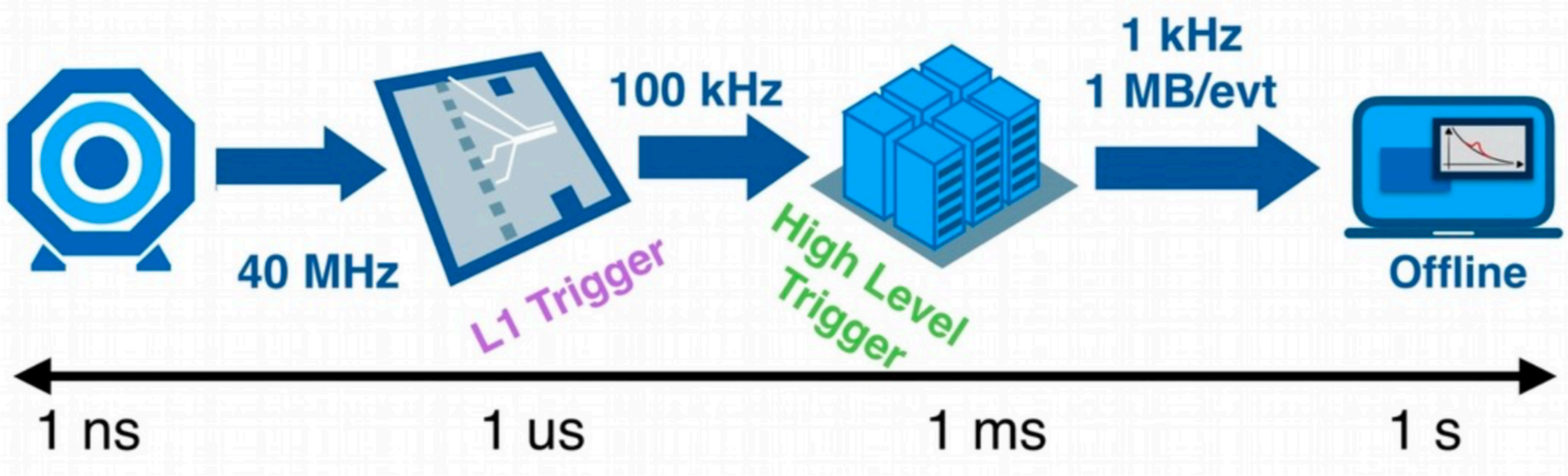
- Machine learning has become a common tool for broad spectrum of HEP problems
  - Particle identification
  - Calibrations/corrections
  - Energy regression
  - Jet/event classification
- Trigger/readout imposes limitations on use of ML
- Recent developments have further opened up the potential for ML solutions in this realm, exciting possibilities





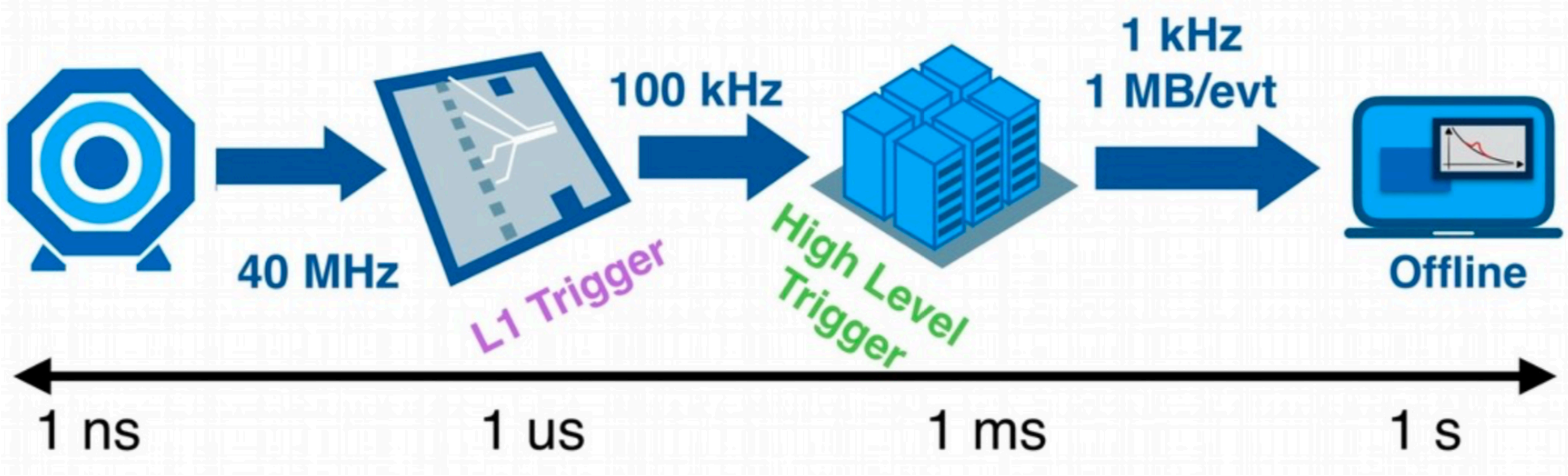


HEP Data Processing / Readout



- **Level-1 Trigger** (hardware: FPGAs) - O(μs) hard latency
- **High Level Trigger** (software: CPUs) - O(100 ms) soft latency
- **Offline** (software: CPUs) - >1 s latencies

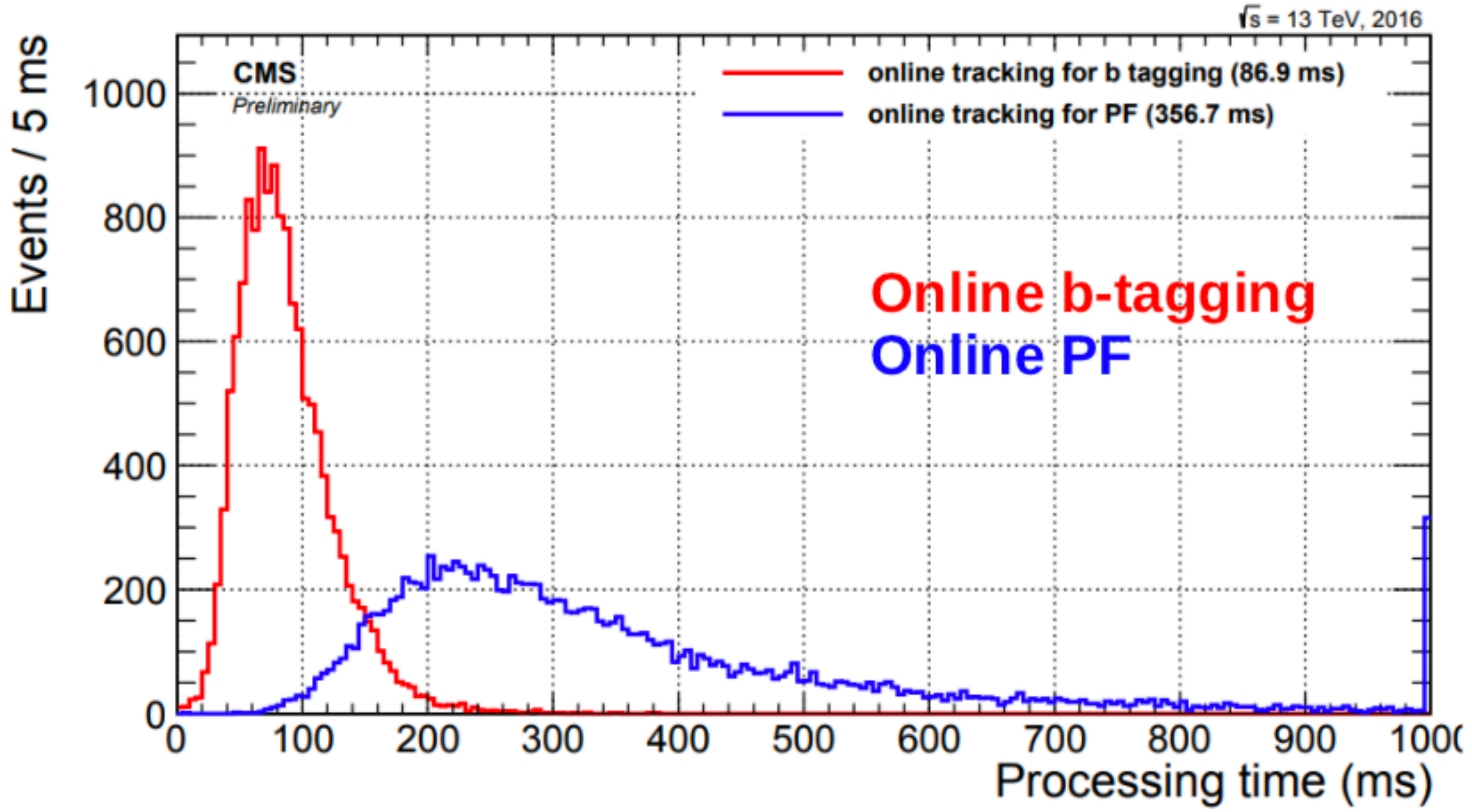
HEP Data Processing / Readout

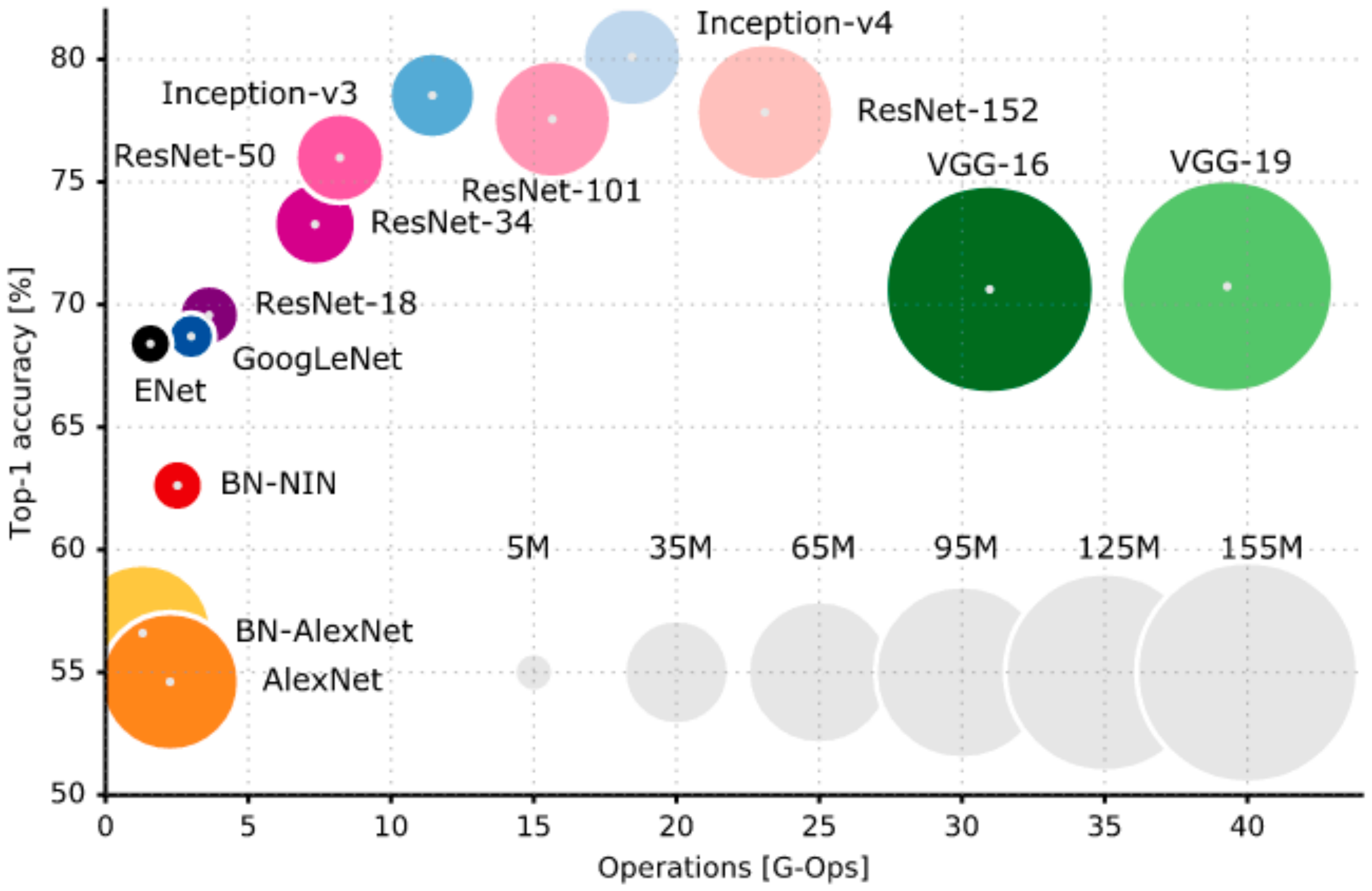


- **Level-1 Trigger** (hardware: FPGAs) - O(μs) hard latency
- **High Level Trigger** (software: CPUs) - O(100 ms) soft latency
- **Offline** (software: CPUs) - >1 s latencies

ML @ HLT

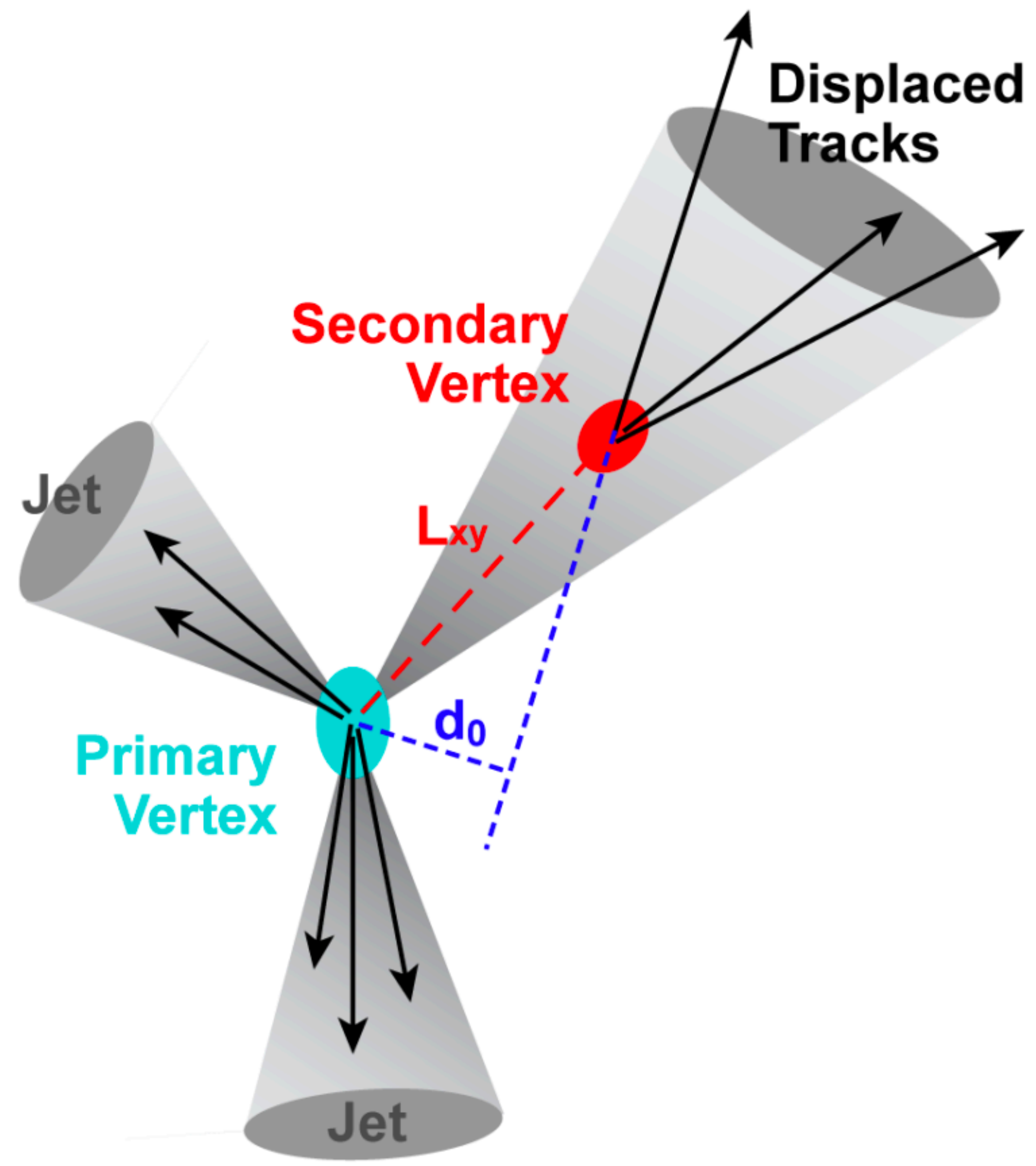
- HLT allows use of CPUs for inference
    - Heterogeneous systems also possible (being investigated)
- Similar to offline inference
- Typically still need to be aware of inference latency
    - Can place ML algorithms later in trigger decisions to reduce average processing time
        - Targeted for specific topologies, not used for full event reconstruction
        - **E.g. b-tagging, taus**
    - Can reduce size/complexity of ML algorithm to lower latency
    - Can utilize hardware accelerators
        - **E.g. GPUs, as-a-service**

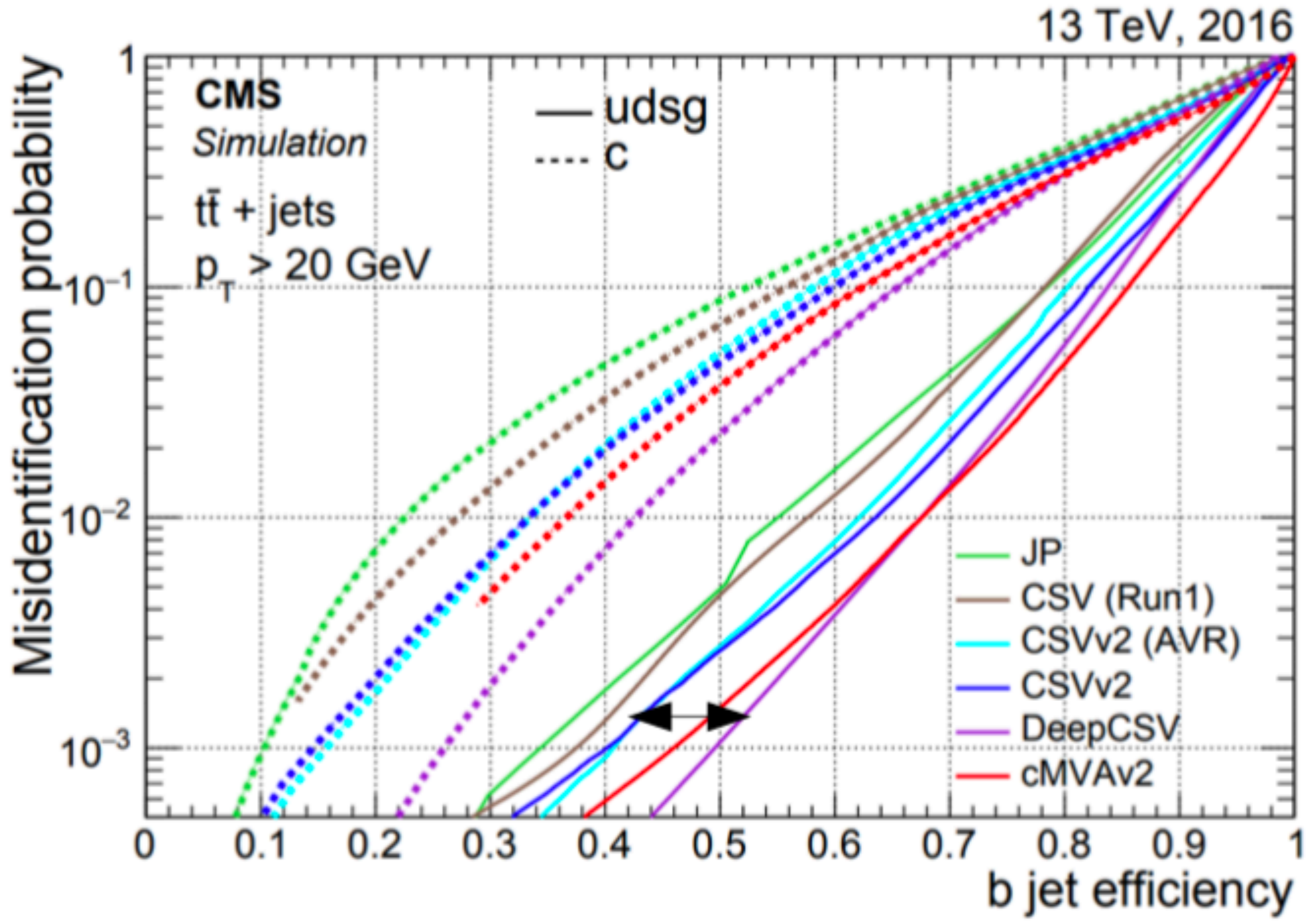


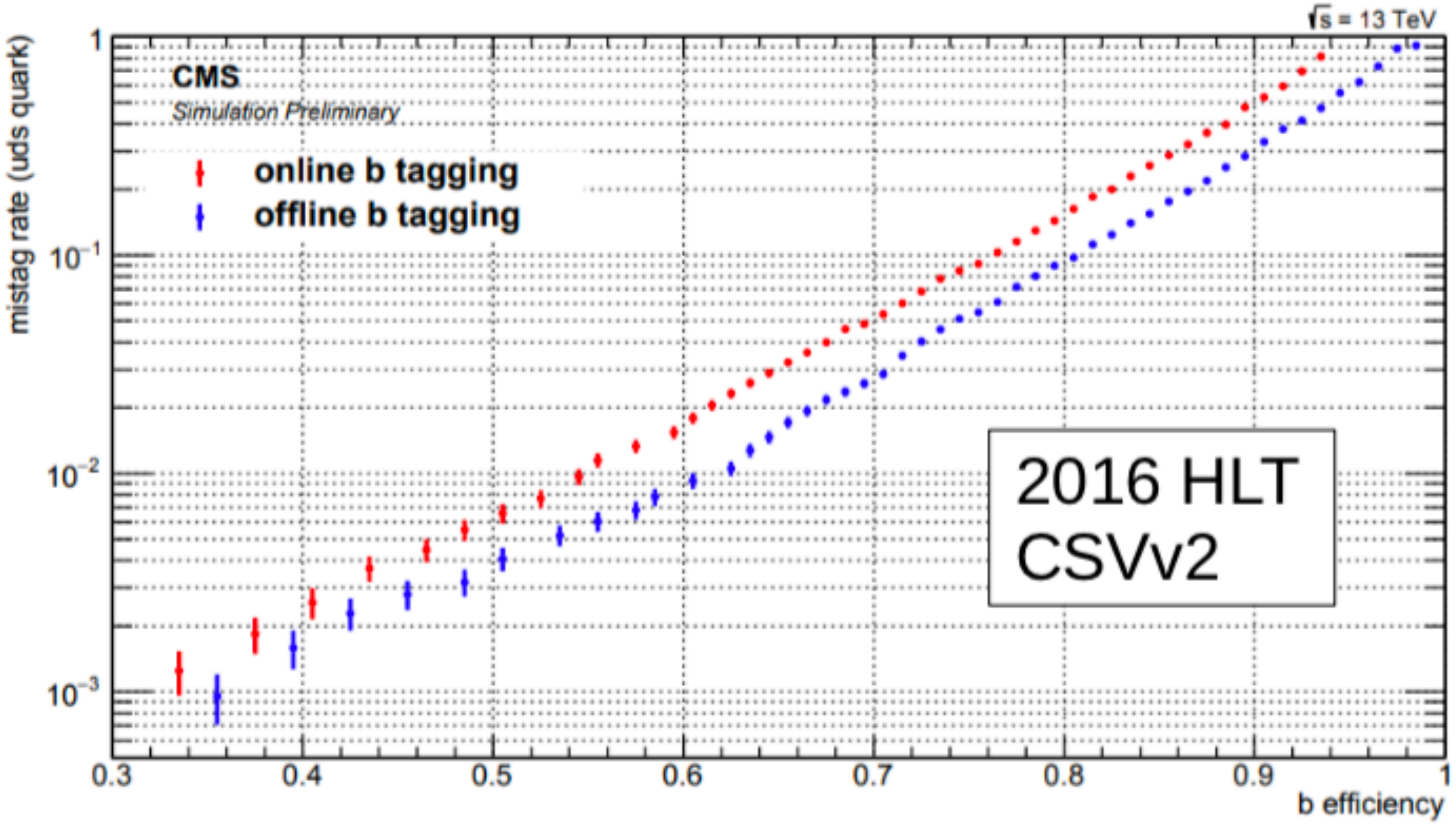


HLT: b-tagging

- Multiple algorithms/architectures (relatively mature usage)
- Ex. CMS:
  - **CSVv2**: BDT
  - **DeepCSV**: DNN, ~50k parameters
- Good online performance
  - Run on small fraction of events
  - Minimal performance degradation w.r.t. offline

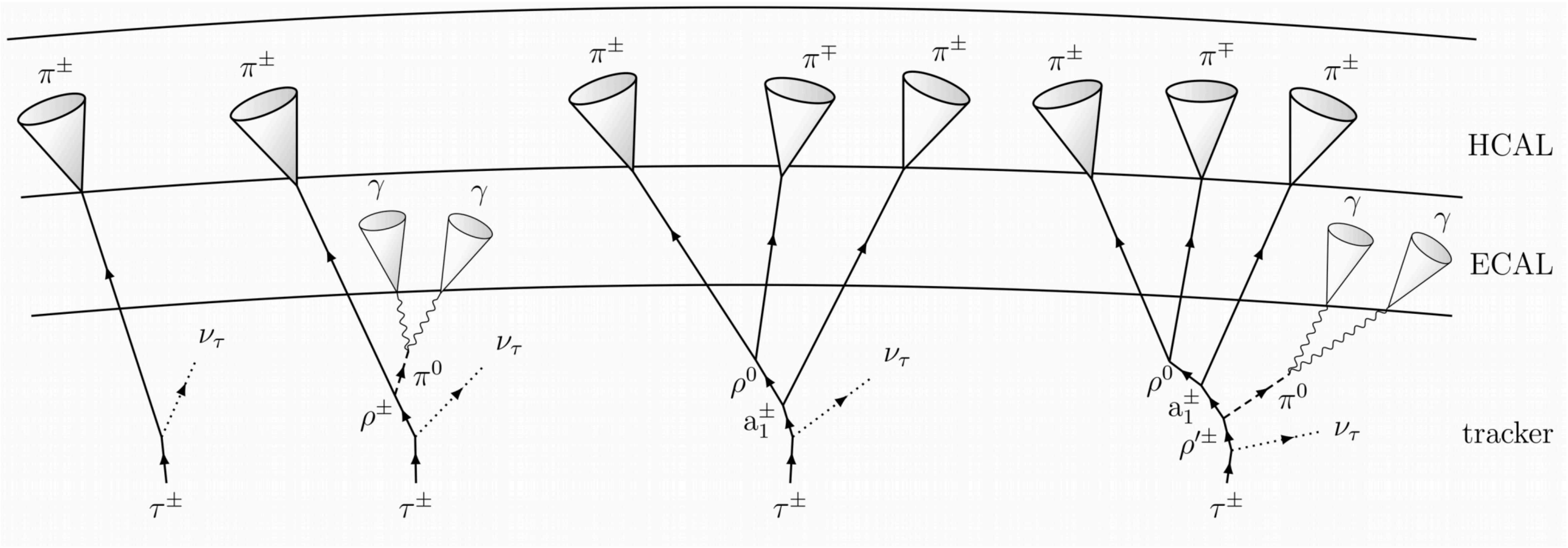


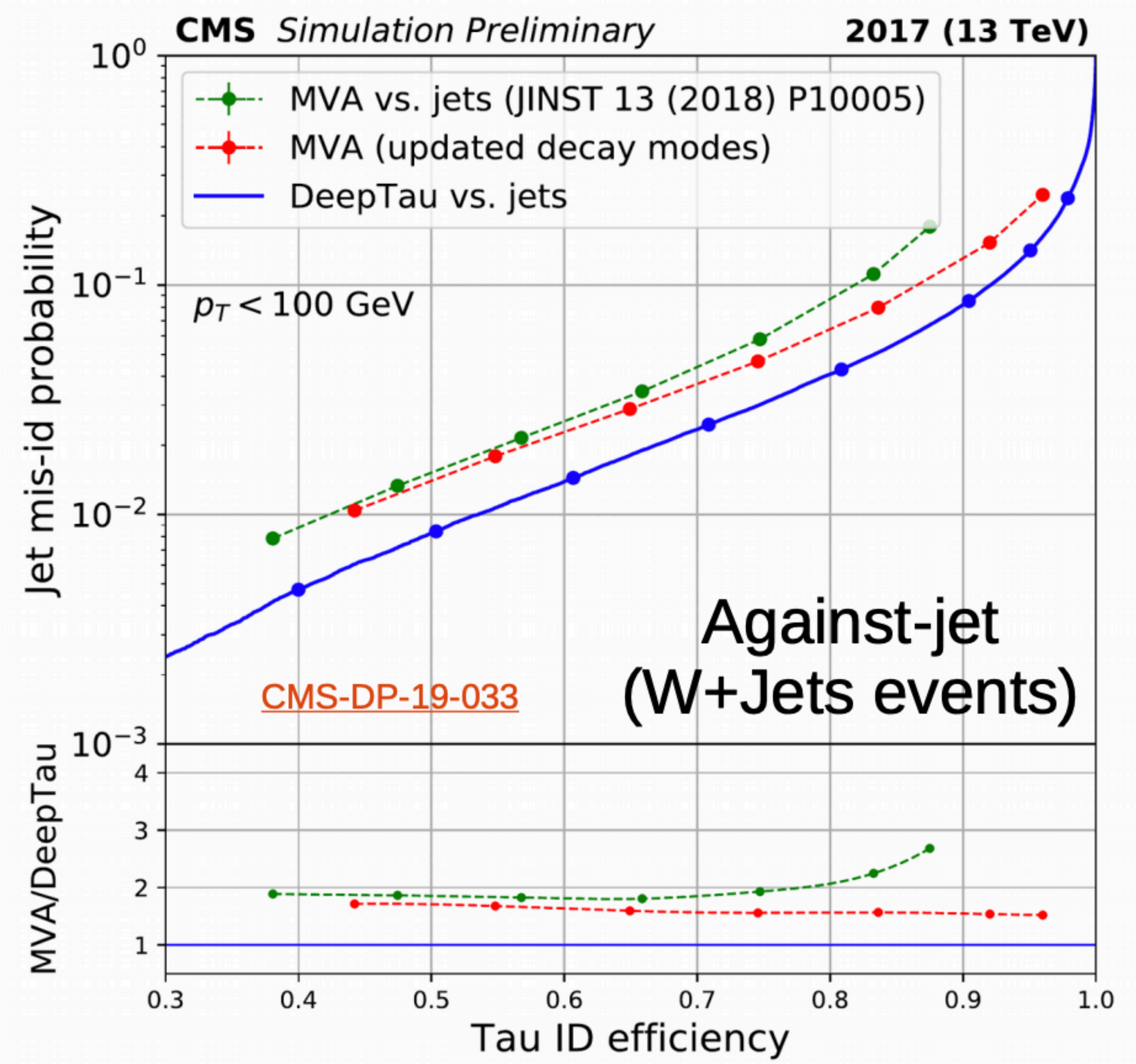


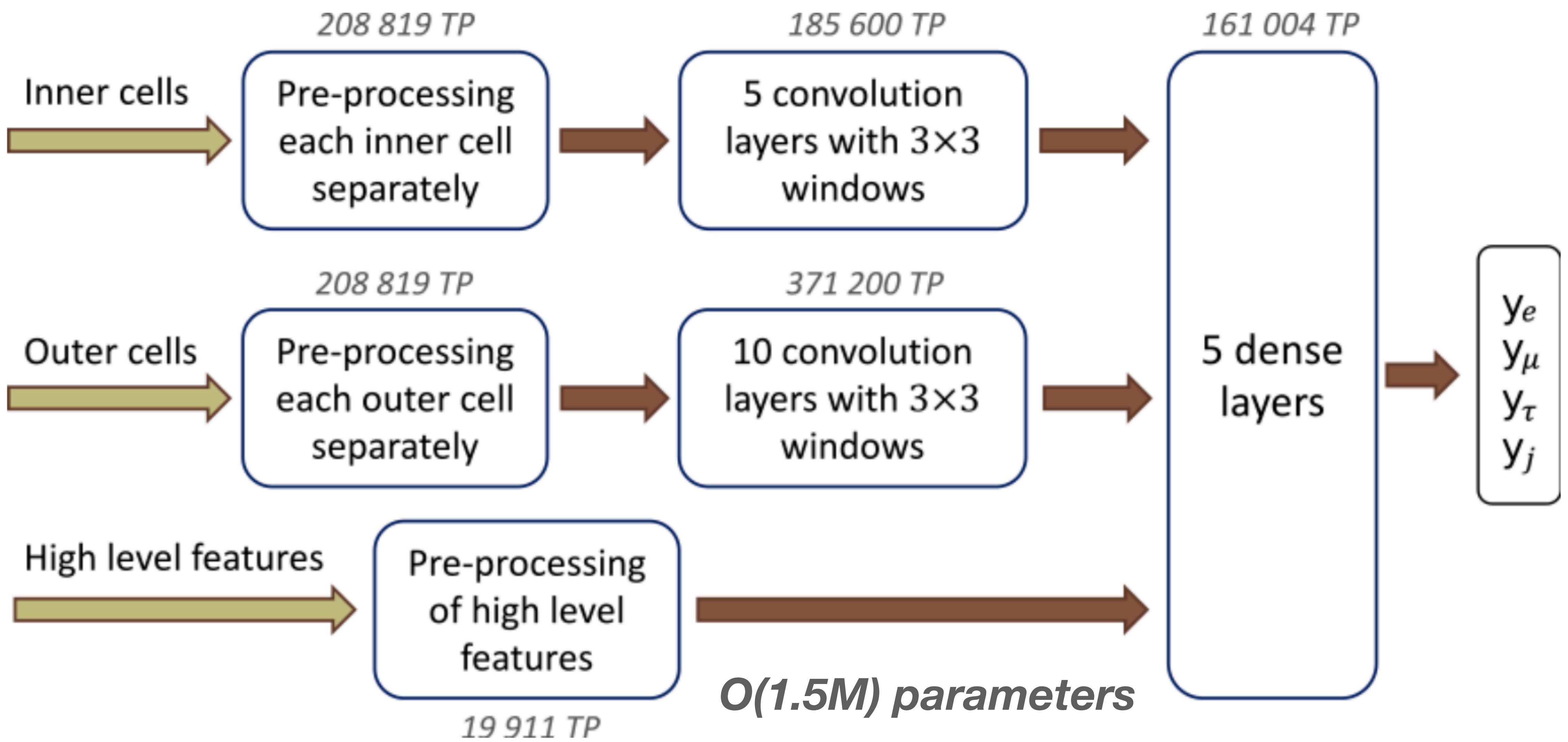


HLT: Tau ID

- ML has become popular tool for tau identification
- Large CNN- and LSTM-based networks from CMS and ATLAS
- Much larger latencies than b-tagging networks, harder to implement into trigger
    - Heterogeneous systems offer some promise (plans for future upgrades)



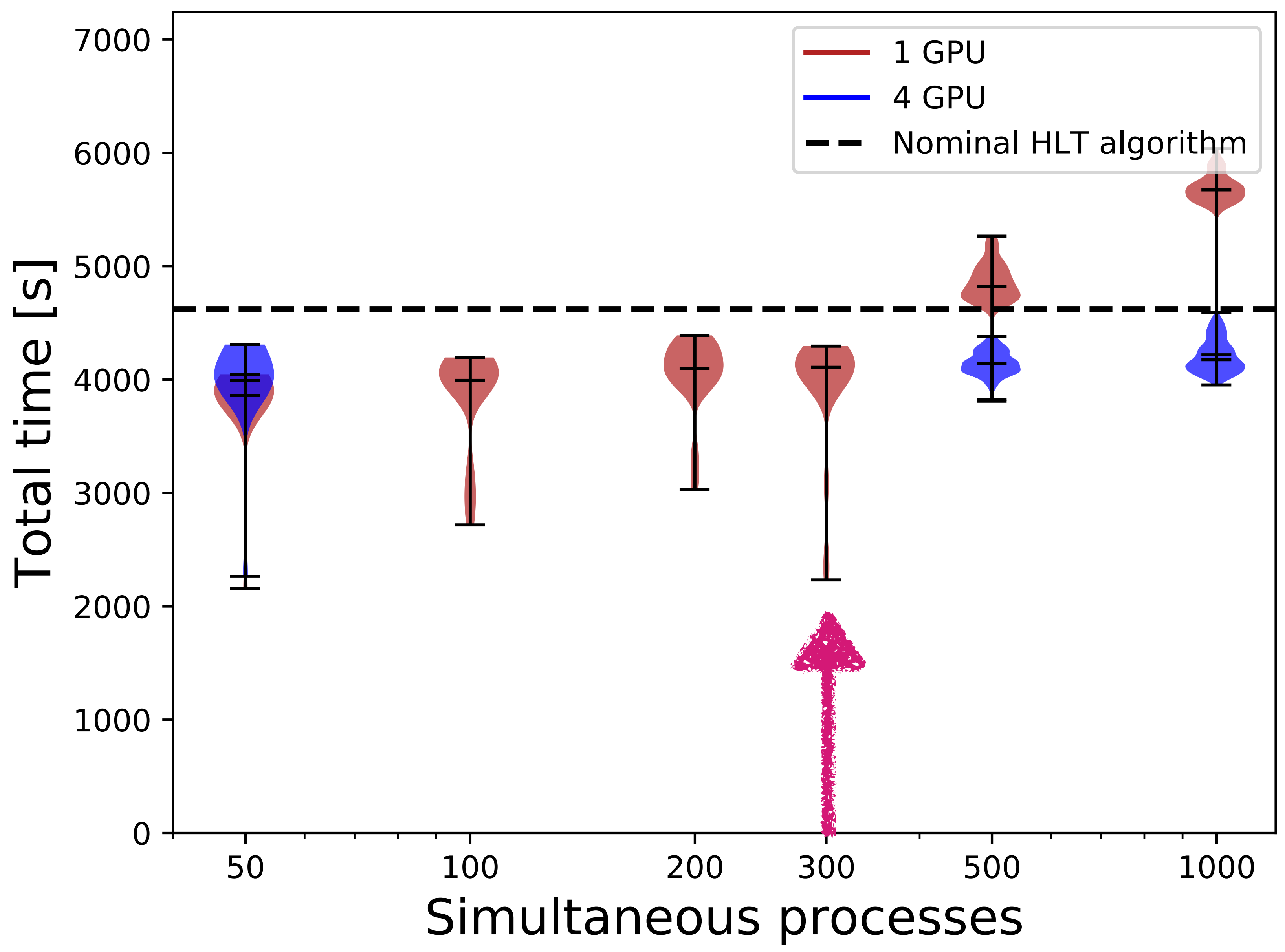




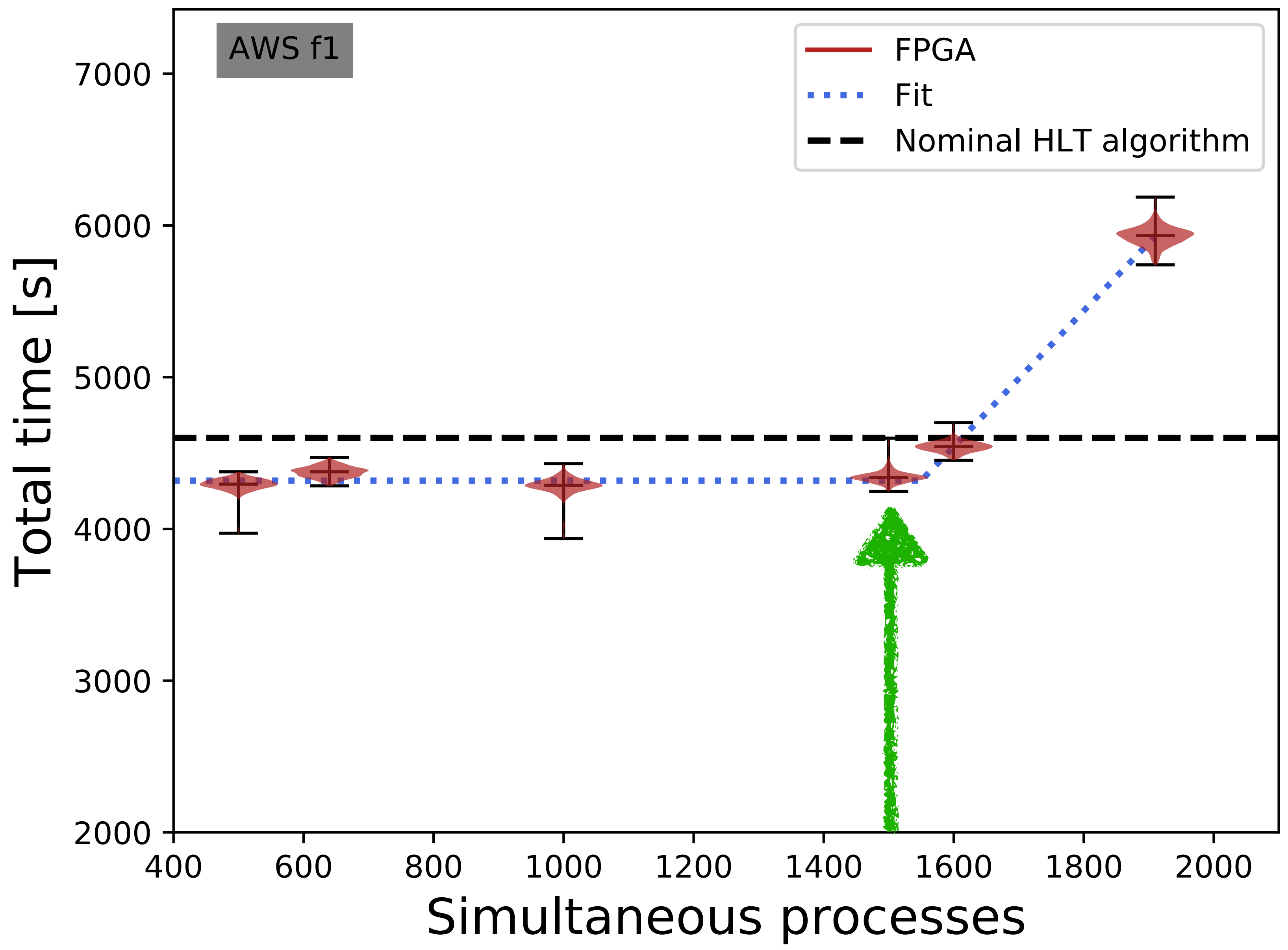
HLT: Heterogeneous Systems

- Heterogeneous systems can speed up ML inference significantly
- SONIC (*see Nhan Tran's talk*) provides framework to take full advantage of coprocessor resources
  - ML as-a-service
- Both FPGAs and GPUs shown to achieve maximal reduction in processing time in HLT tests (ML HCAL energy regression)
  - Single GPU (FPGA) can seamlessly serve up to **300** (**1500**) CPUs
  - arXiv:2007.10359, arXiv:2010.08556

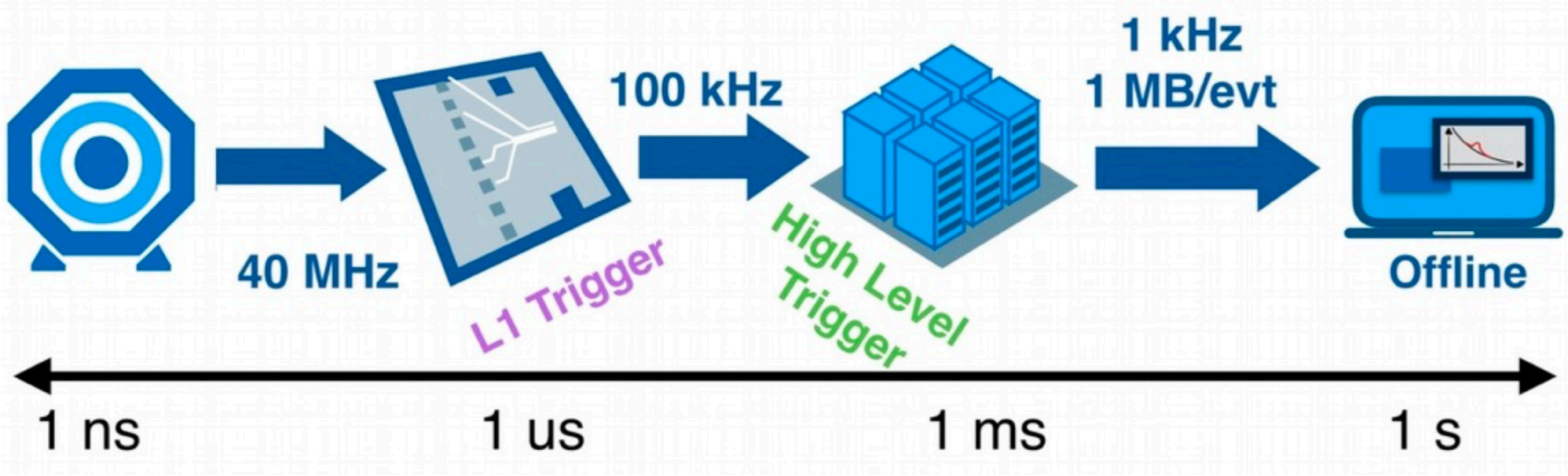
V100 GPU



AWS f1 FPGA (VU9P)



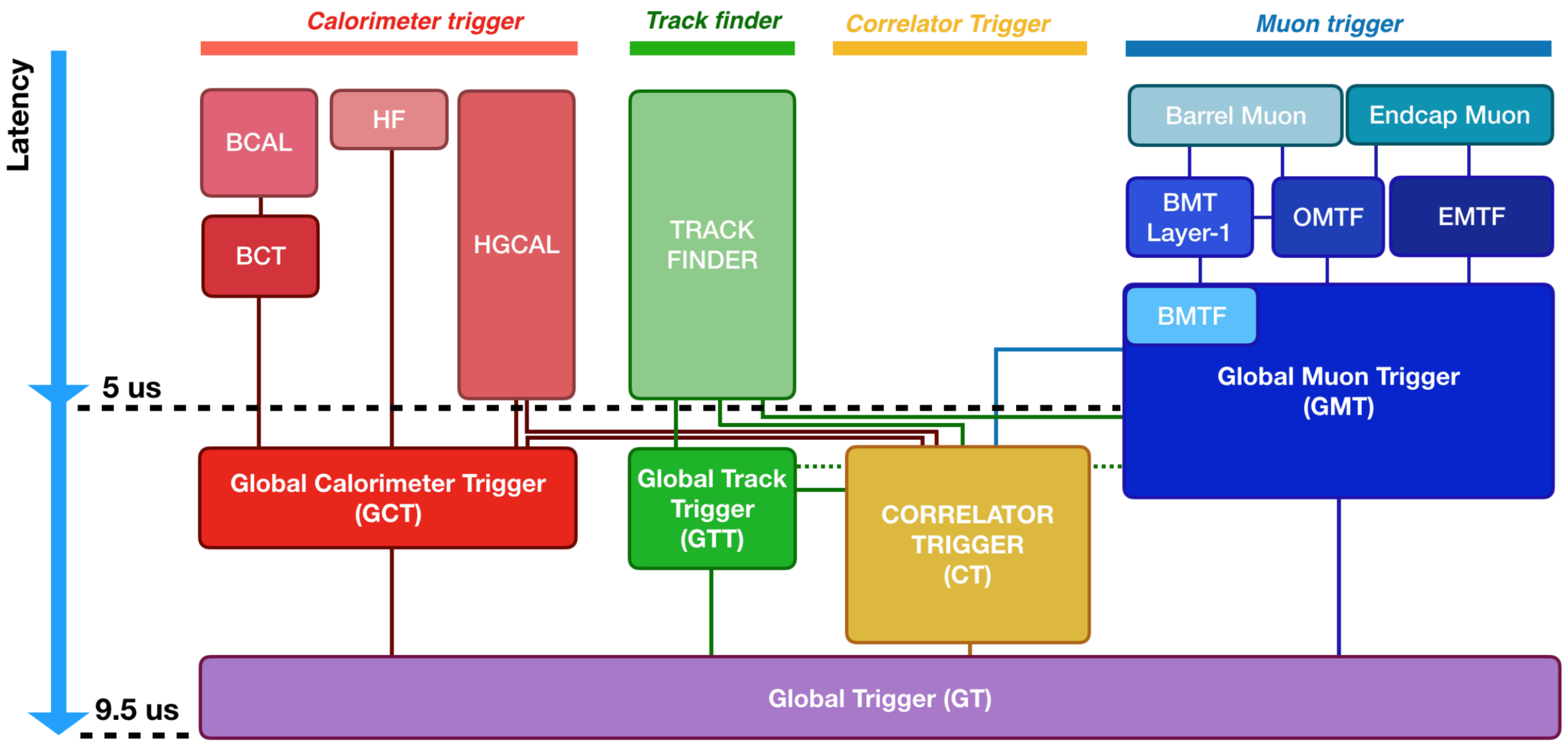
HEP Data Processing / Readout

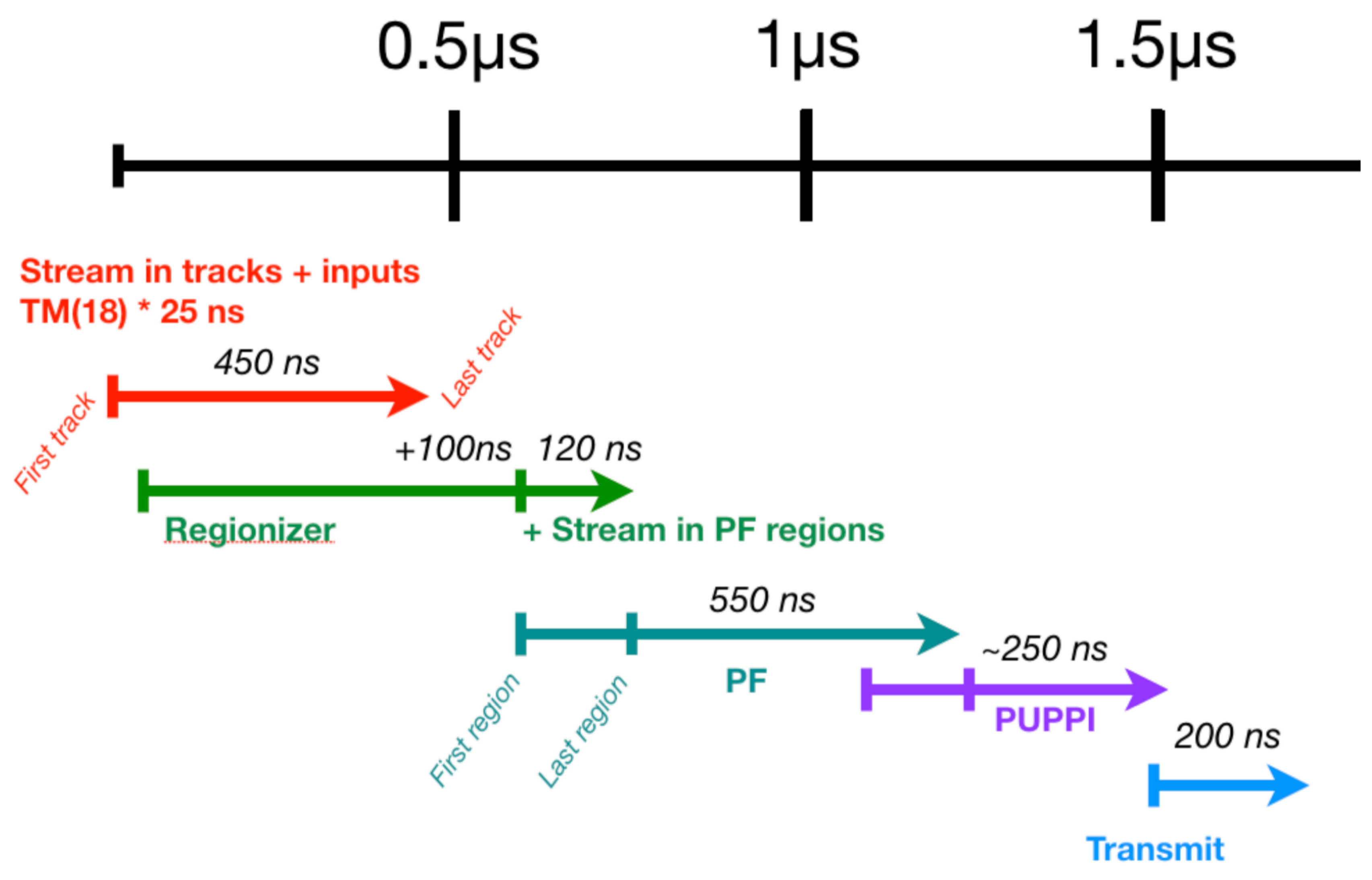


- **Level-1 Trigger** (hardware: FPGAs) - O(μs) hard latency
- **High Level Trigger** (software: CPUs) - O(100 ms) soft latency
- **Offline** (software: CPUs) - >1 s latencies

ML @ L1

- Very different constraints and hardware compared to CPU/GPU
  - Latencies of ~100 ns, scarce resources
  - Can be difficult to develop without lots of specialized knowledge
- hls4ml facilitates usage of ML on FPGAs
  - Support for many different architectures and frameworks
- Growing number of examples/proposed examples in ultra-low latency regime
  - Refer to Nhan Tran's talk for some more examples!

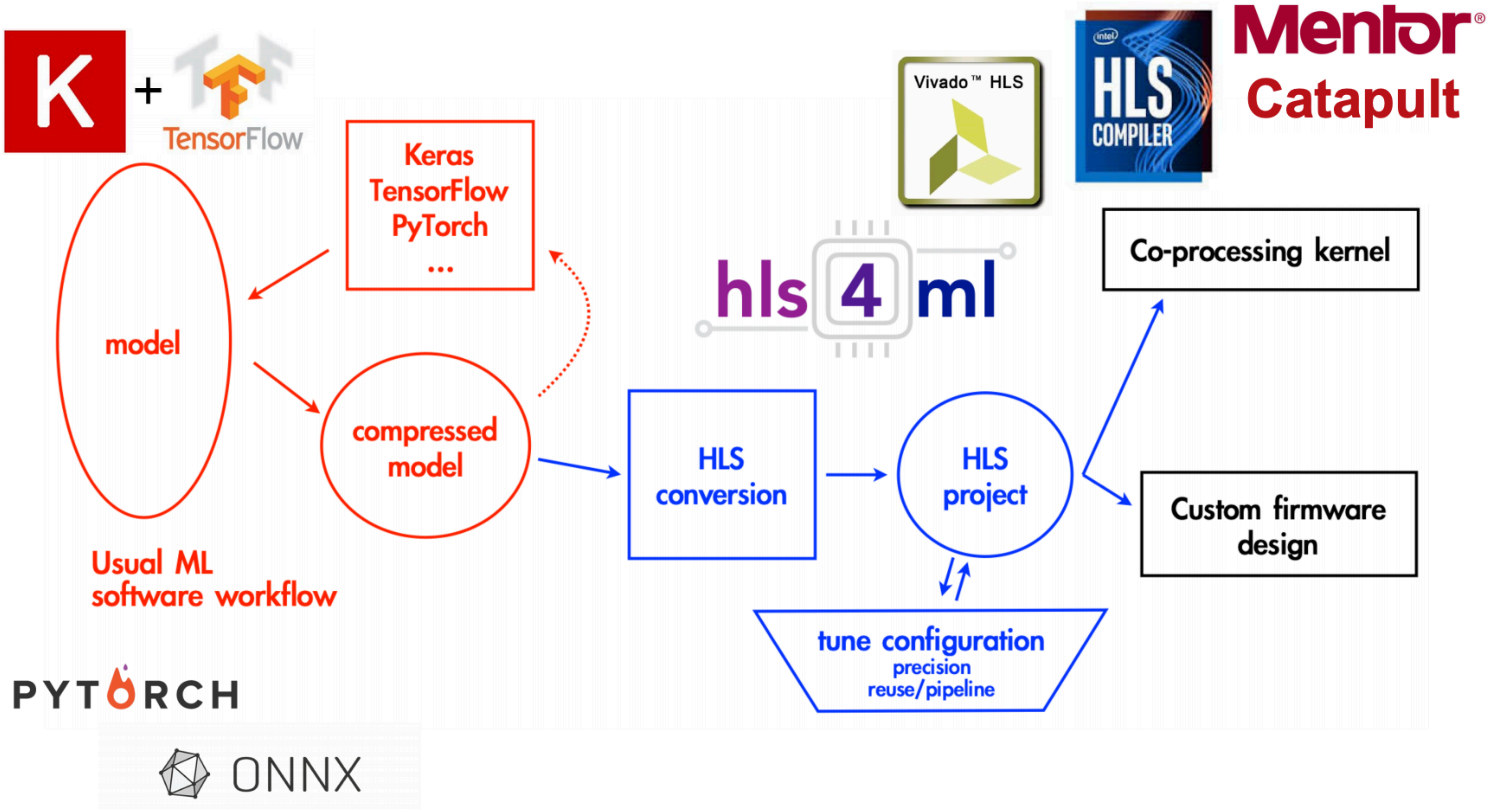




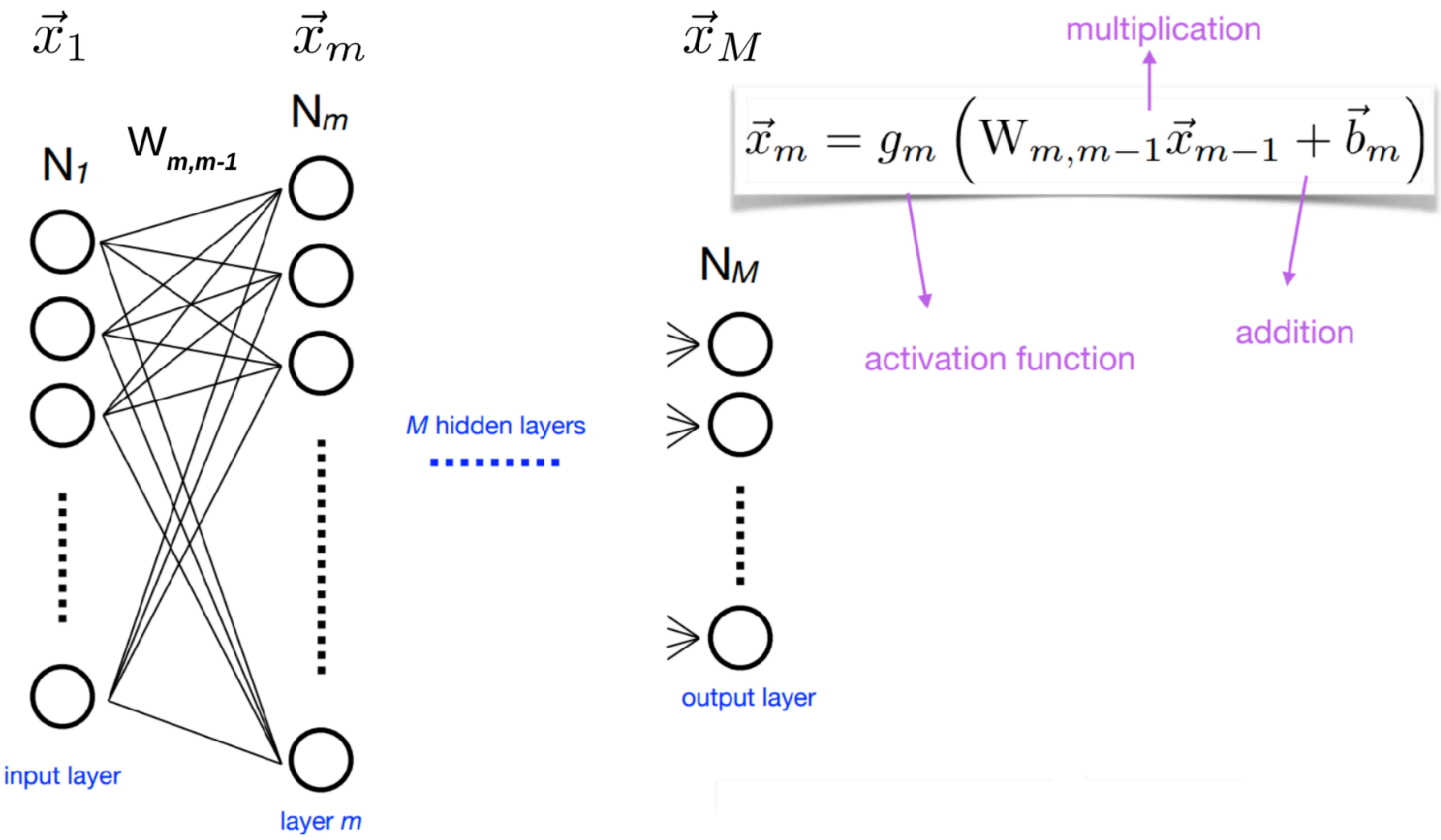
hls4ml

- hls4ml is a software package for creating implementations of neural networks for FPGAs and ASICs
  - https://fastmachinelearning.org/hls4ml/
  - arXiv:1804.06913
- Supports common layer architectures and model software, options for quantization/pruning
  - Output is a fully ready high level synthesis (HLS) project
- Customizable output
  - Tunable precision, latency, resources

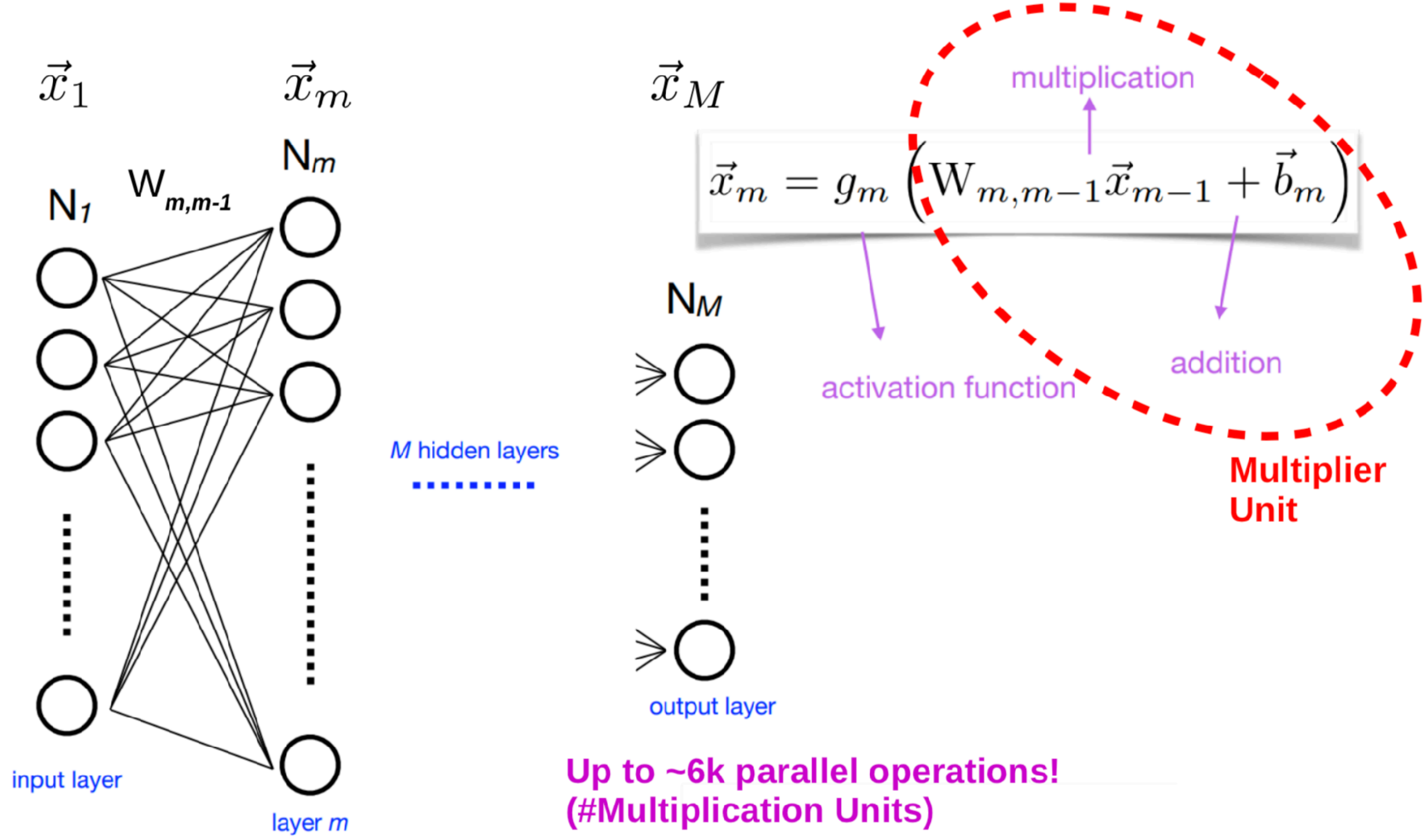
hls4ml Workflow



Inference on FPGAs

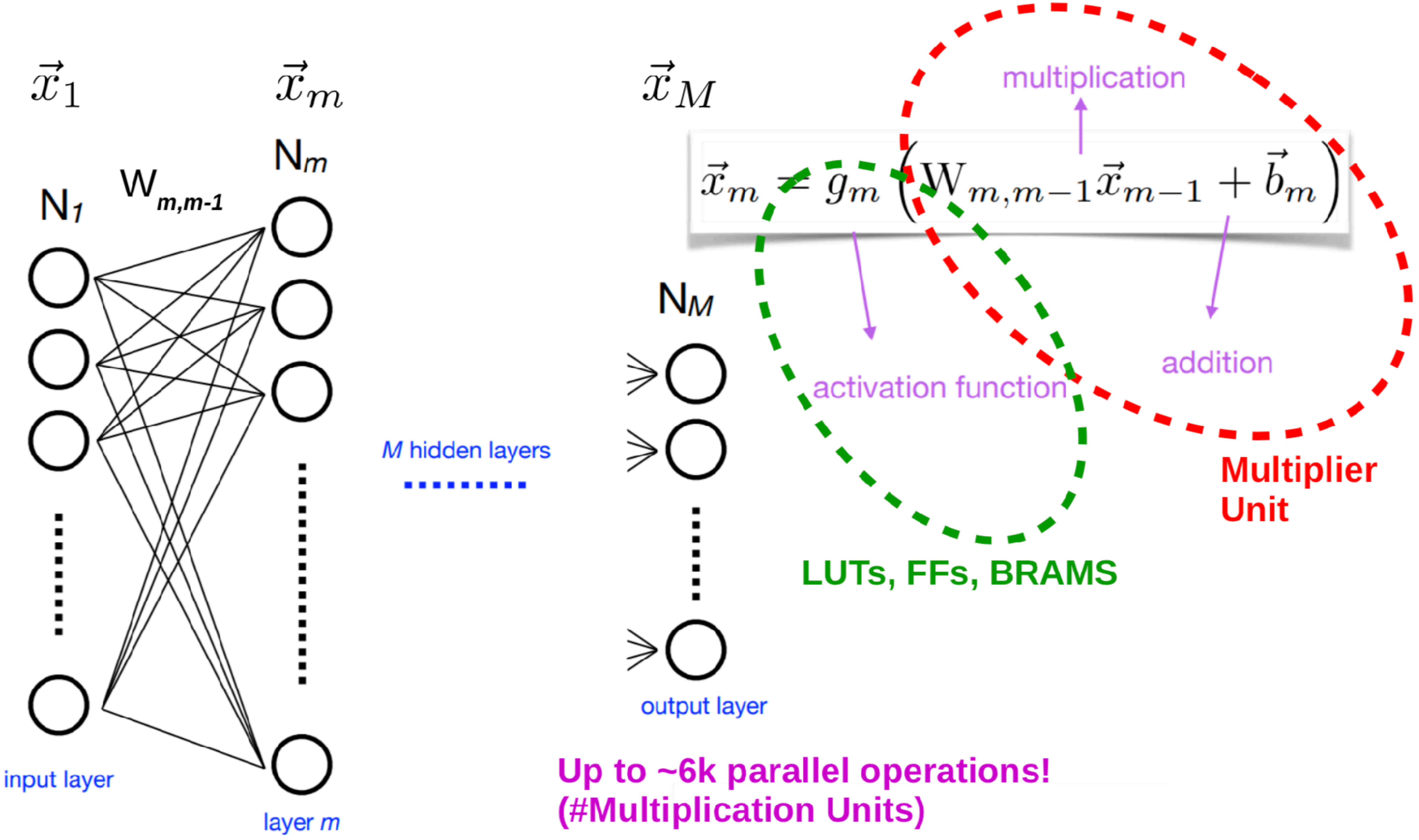


Inference on FPGAs



Up to ~6k parallel operations!
(#Multiplication Units)

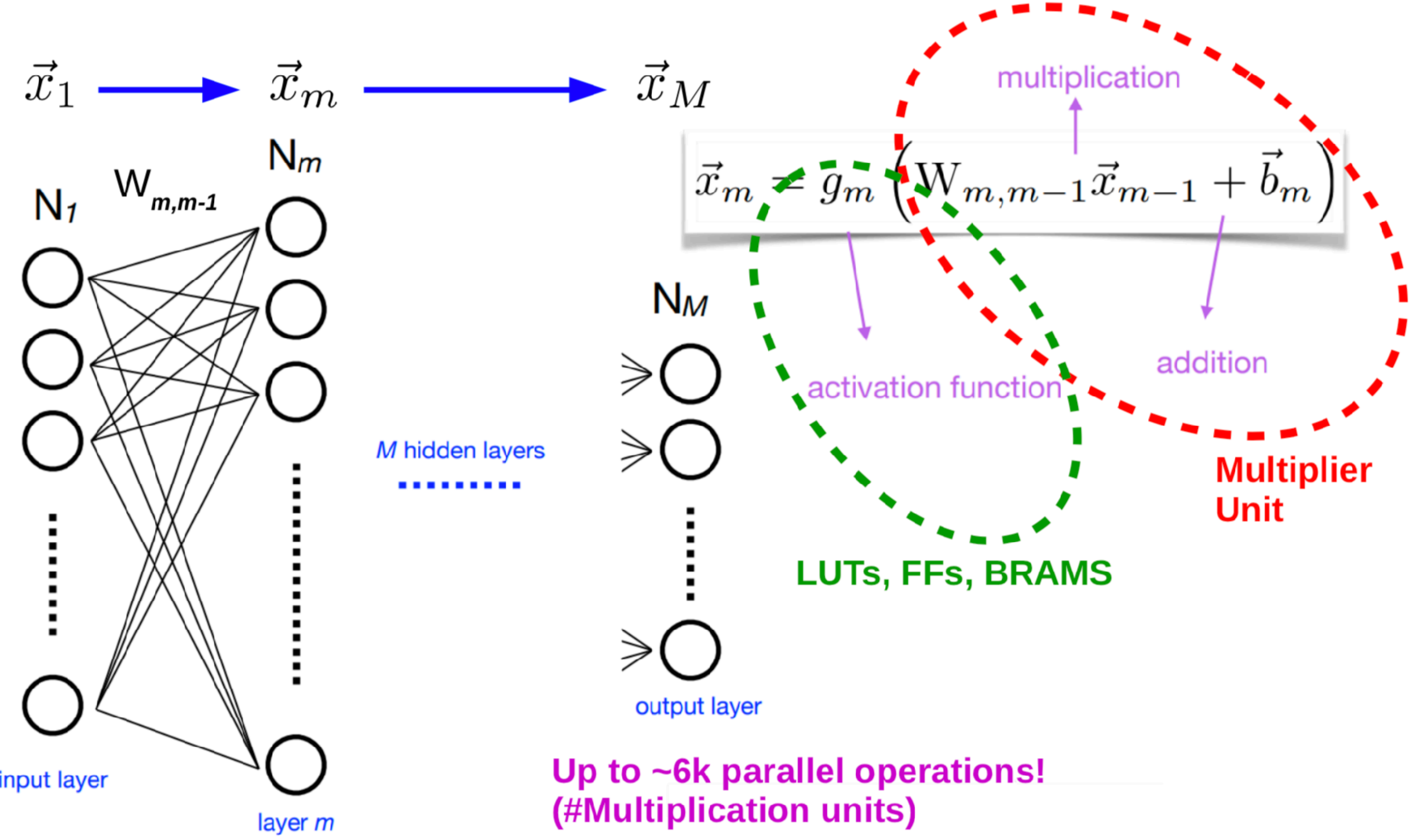
Inference on FPGAs



Up to ~6k parallel operations!
(#Multiplication Units)

Inference on FPGAs

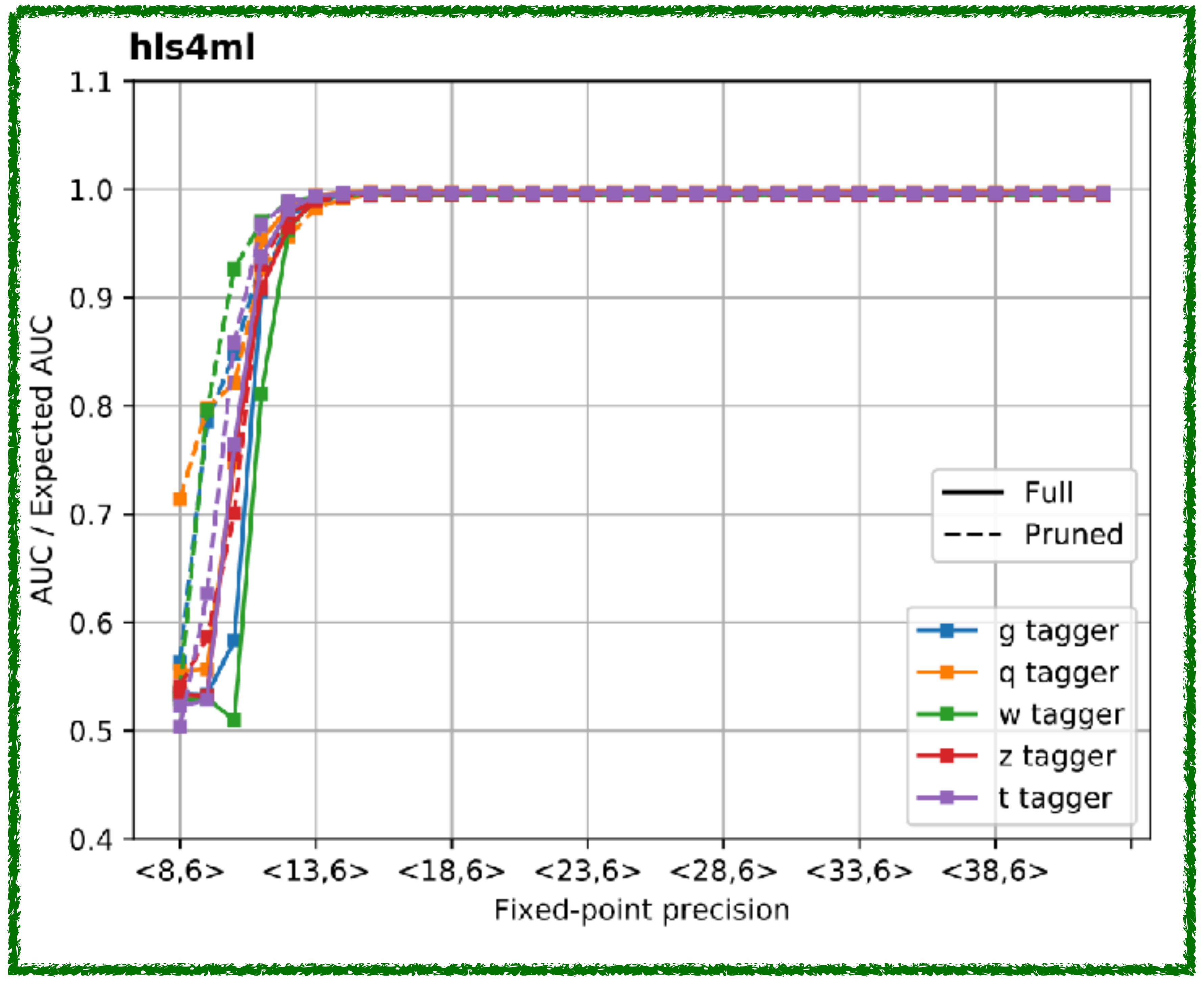
Every clock cycle
(all layer operations
can be performed
simultaneously)

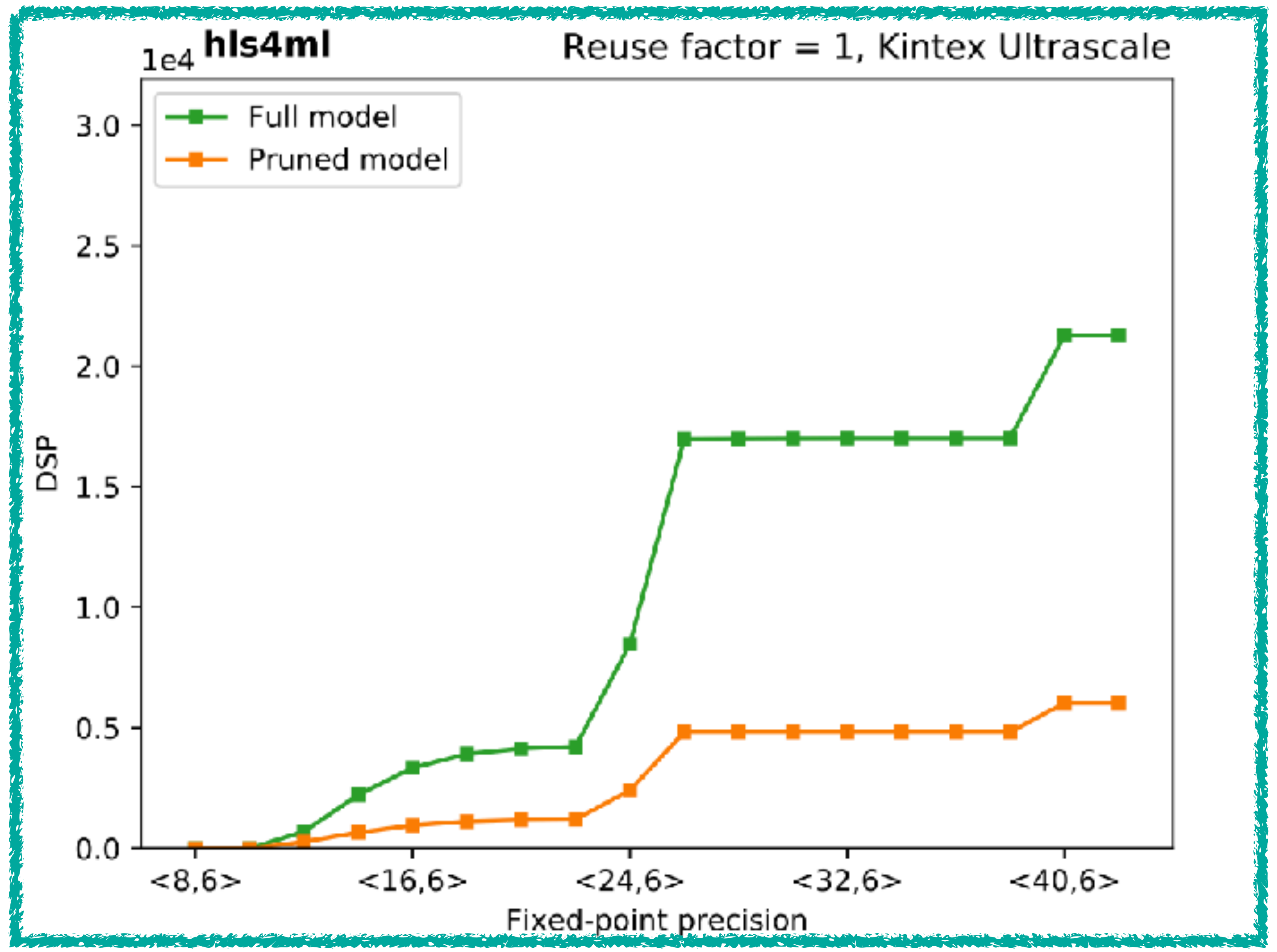


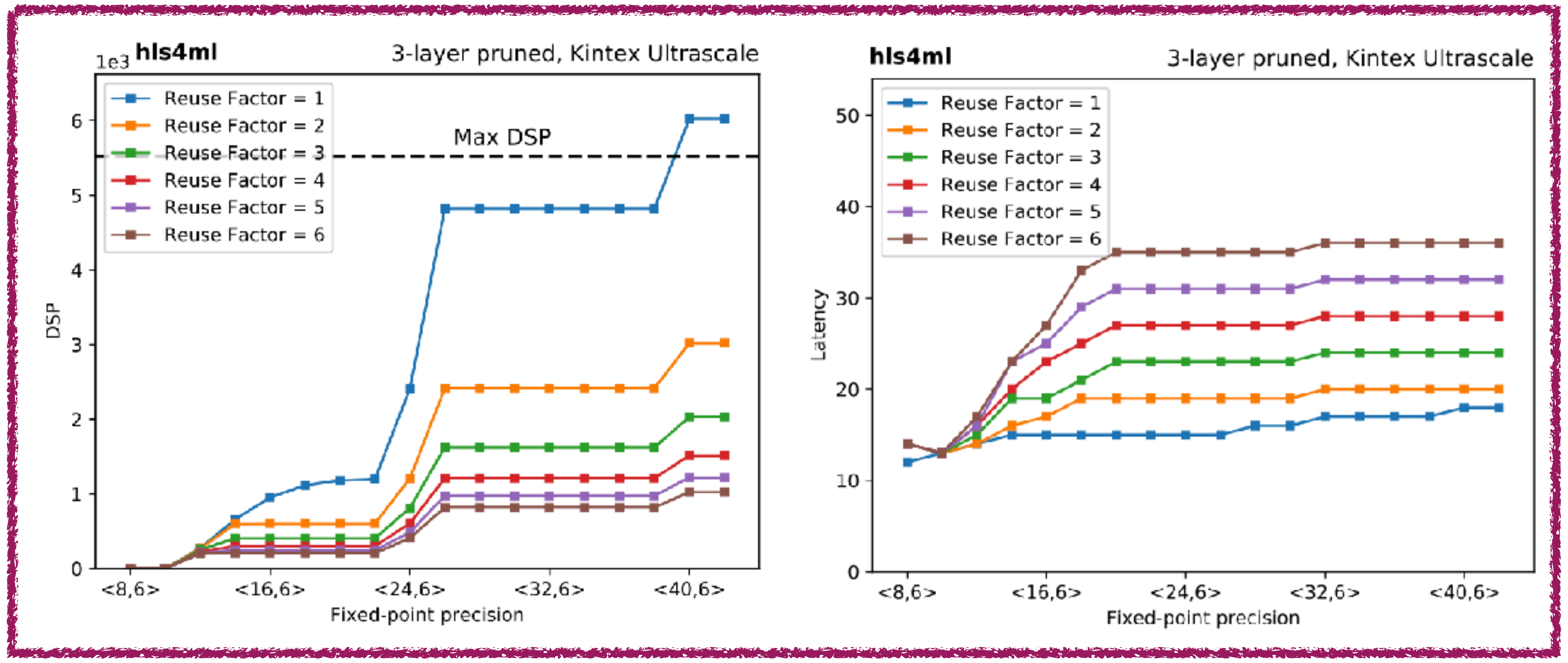
Up to ~6k parallel operations!
(#Multiplication units)

hls4ml Customization

- Multiple different knobs to adjust design for desired performance/latency/resource usage
  - Pruning
  - Quantization
  - Reuse

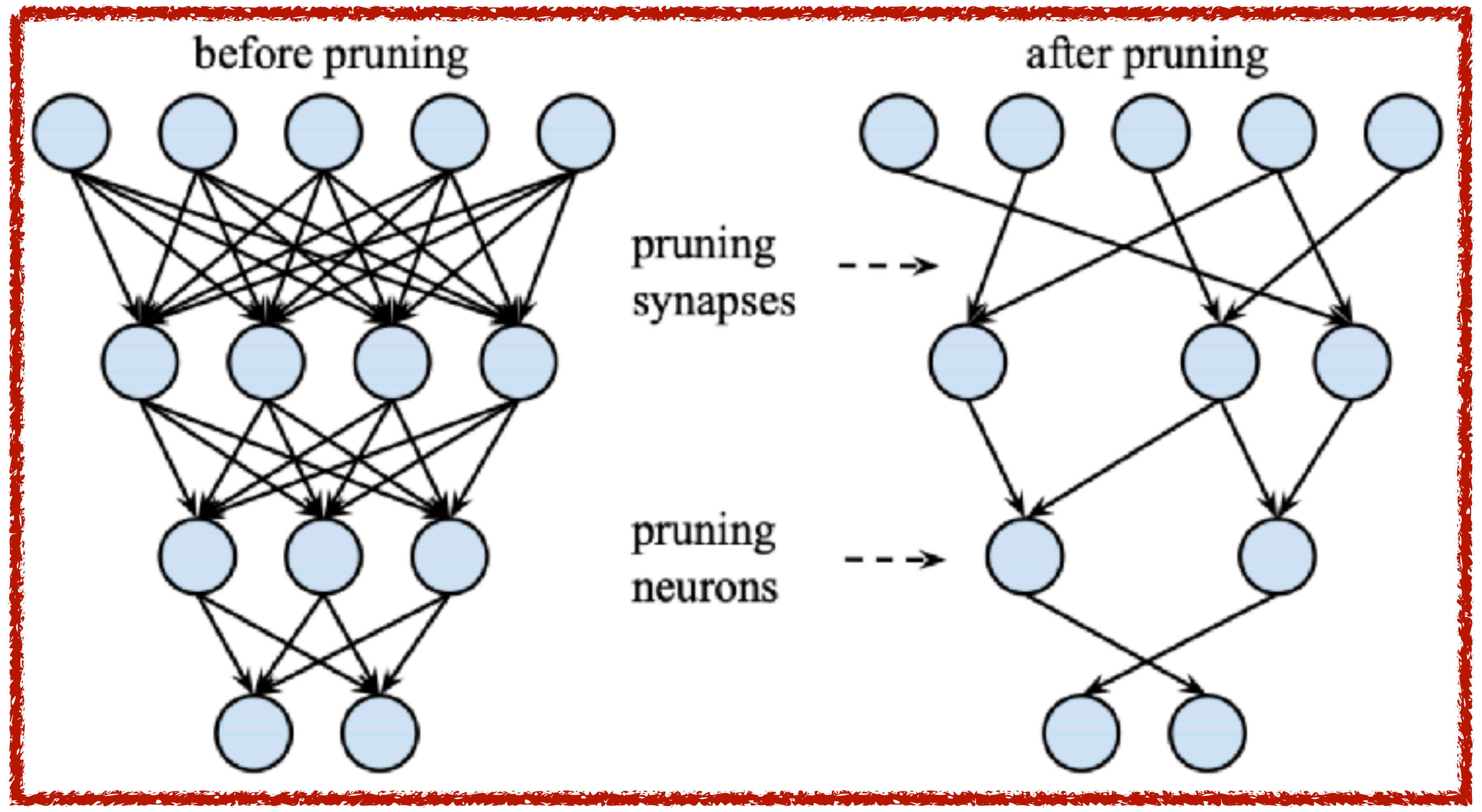




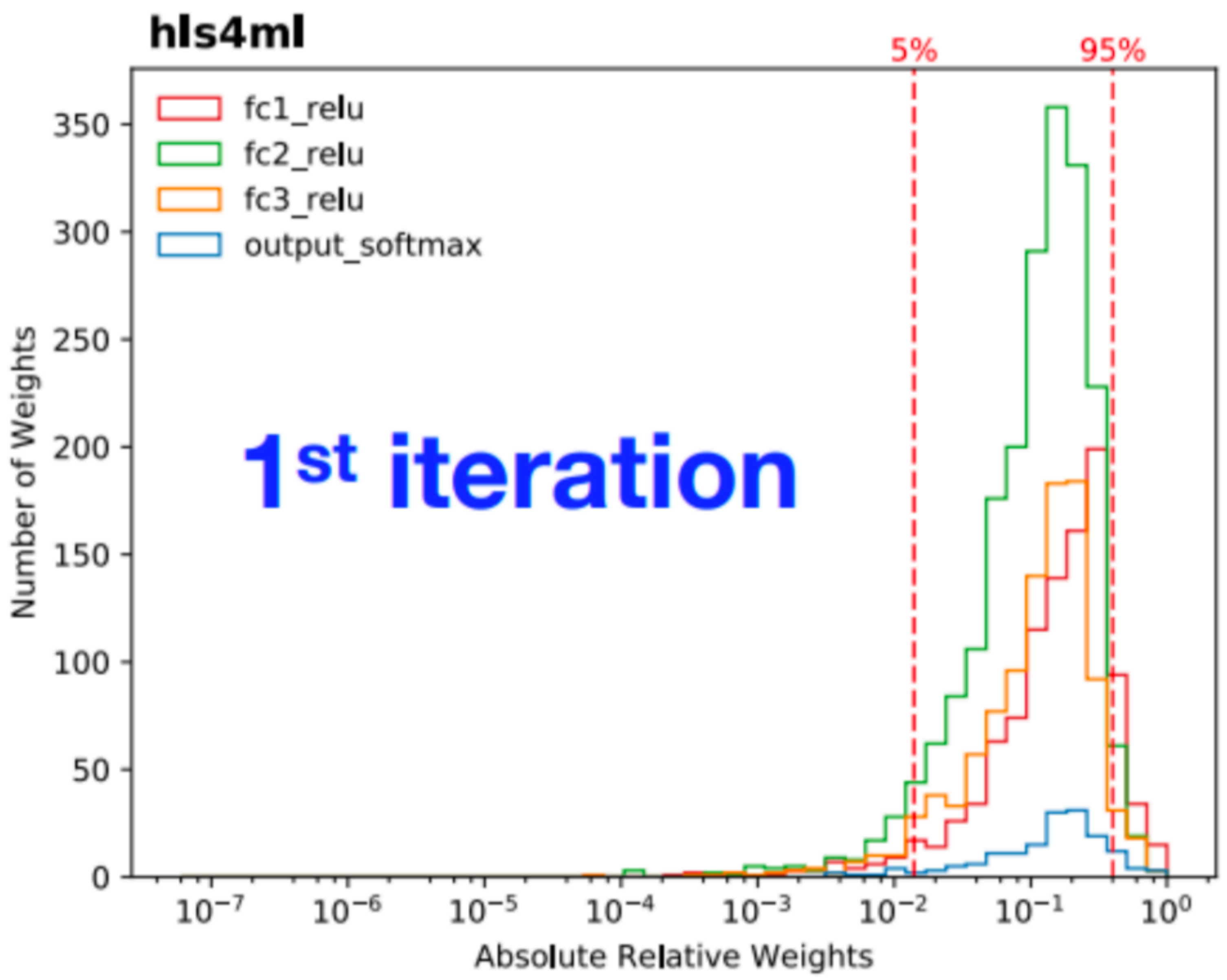


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining “best” way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
    - Use L1 regularization (penalty term in loss function for large weights)
    - Remove smallest weights
    - Repeat
- HLS automatically removes multiplications by 0

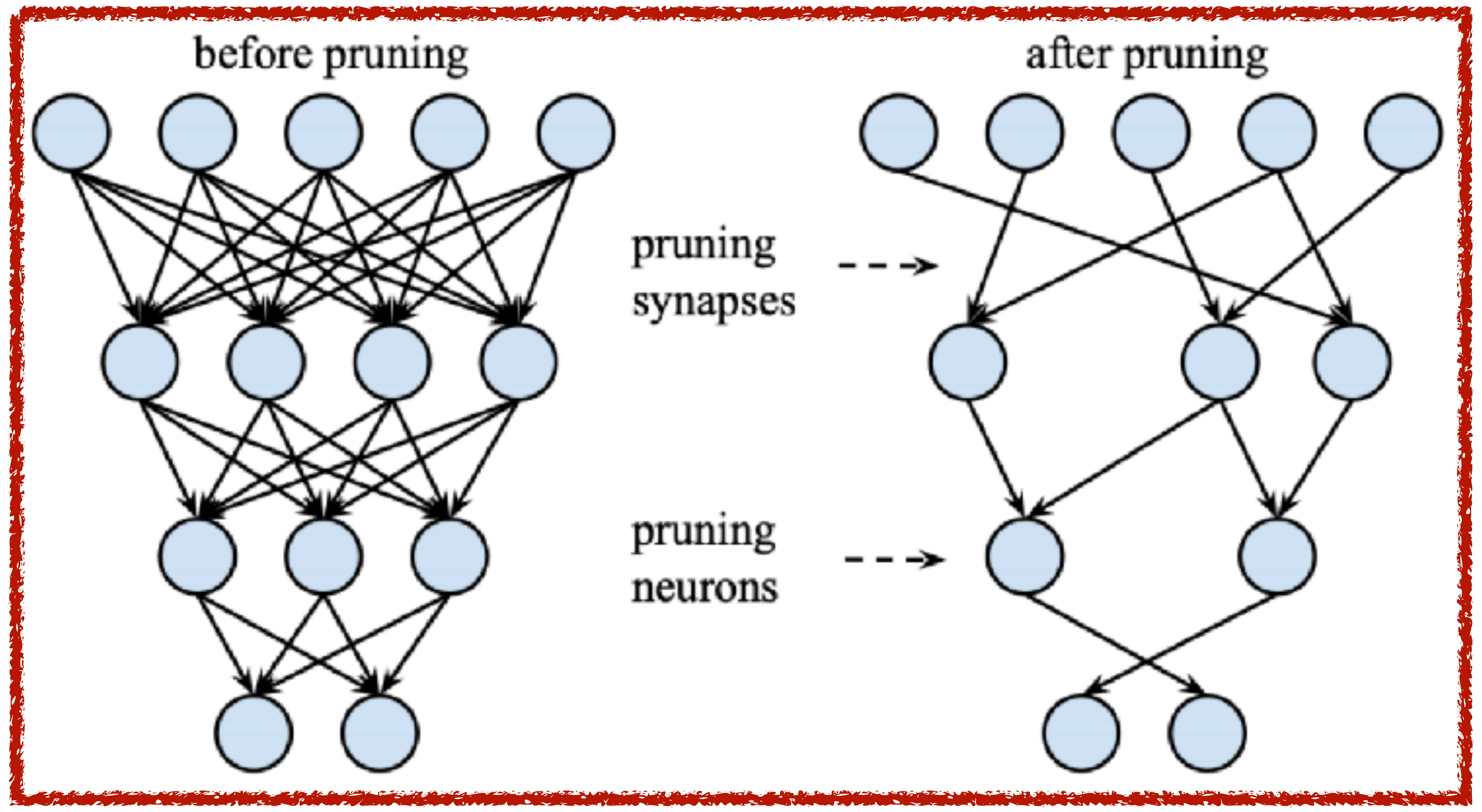


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda||\mathbf{w}||$$

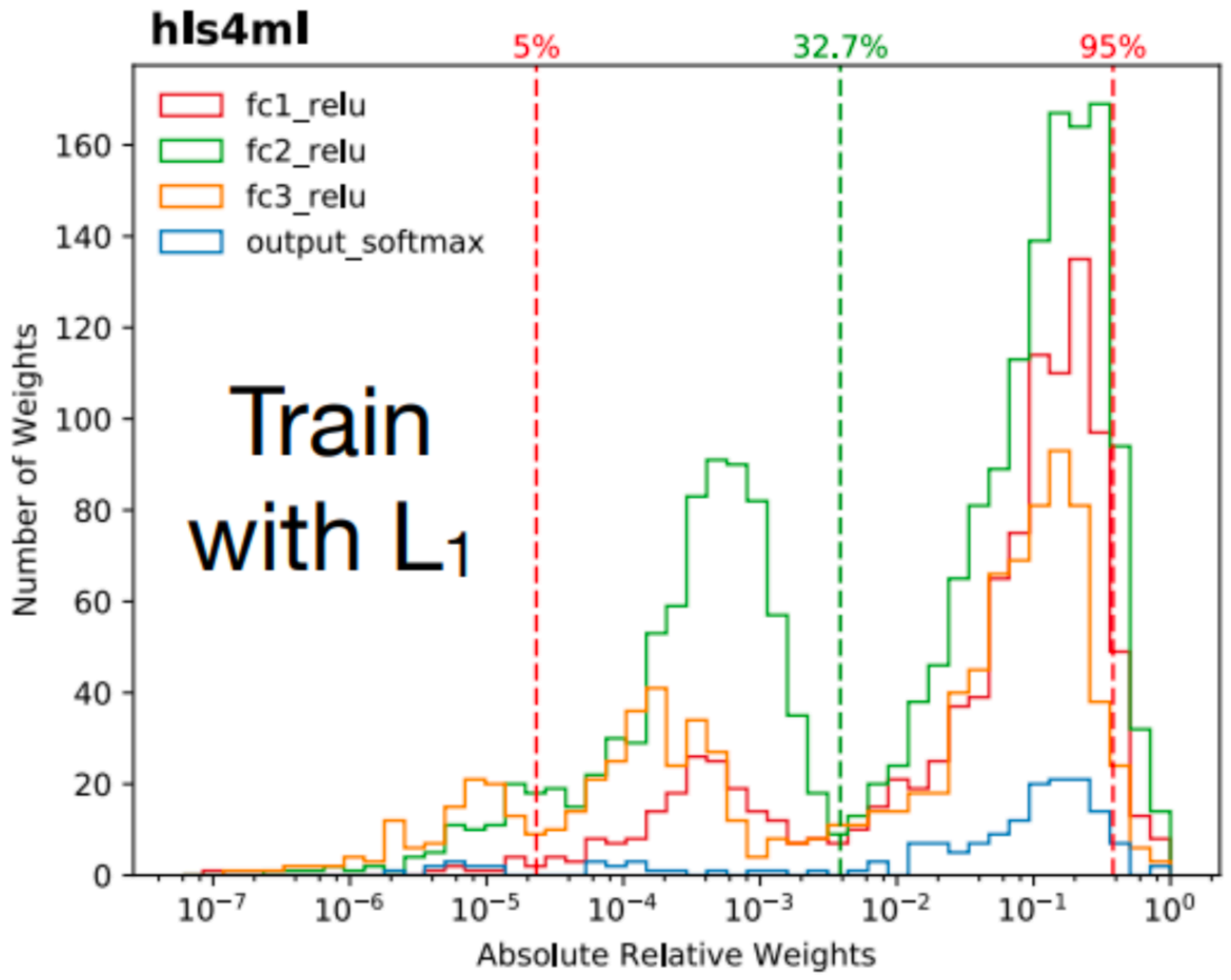


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
  - Use L1 regularization (penalty term in loss function for large weights)
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0

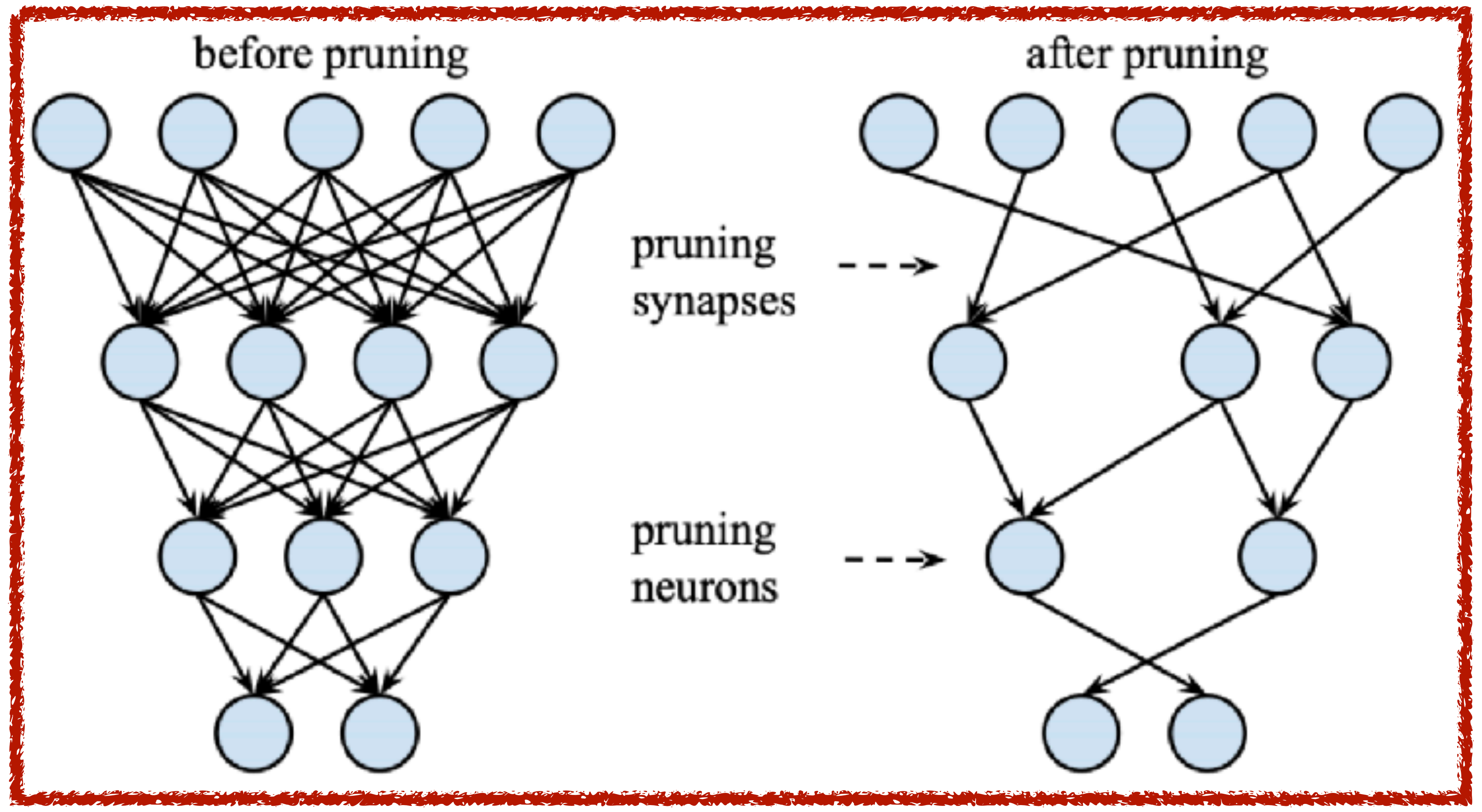


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$

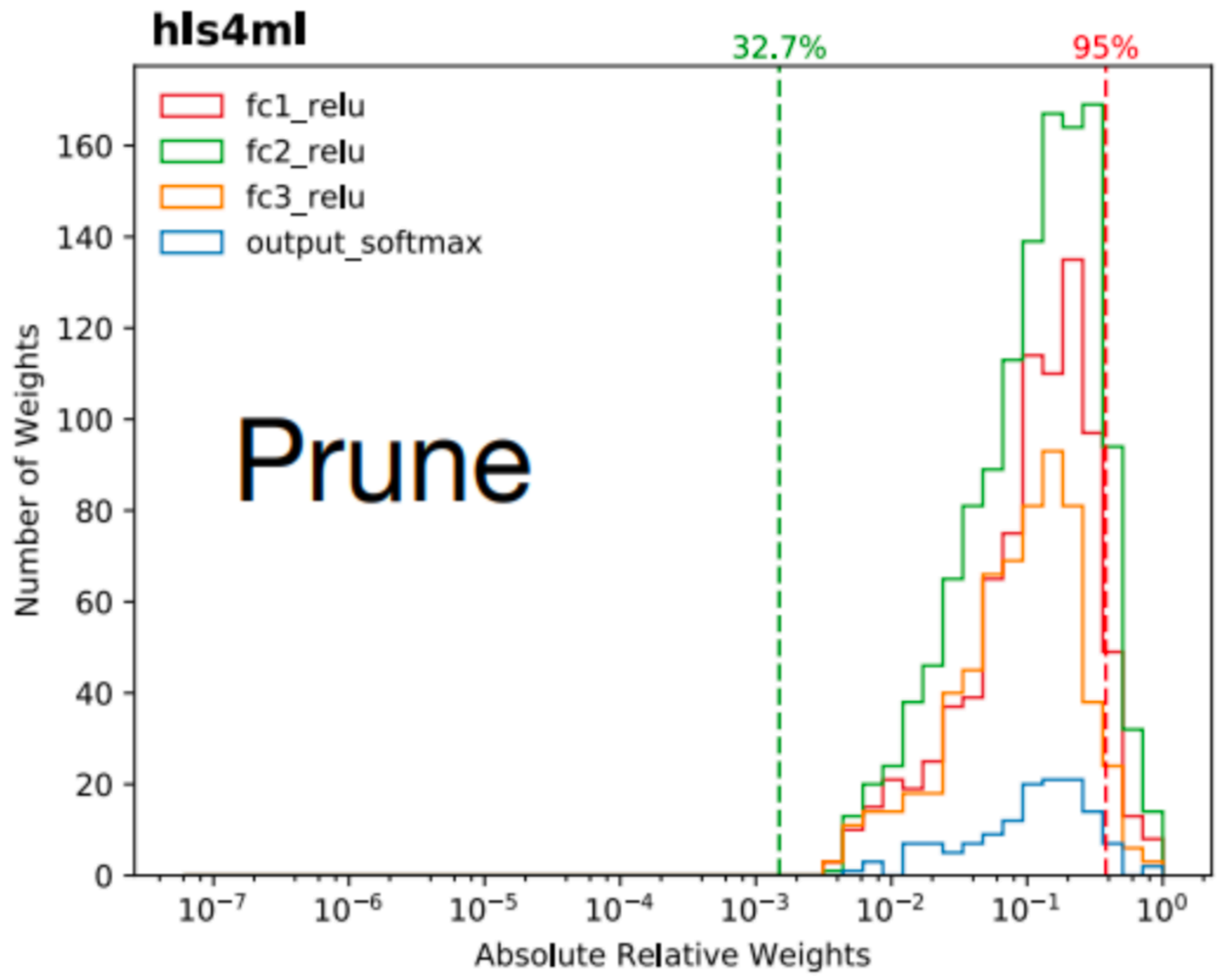


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
  - Use L1 regularization (penalty term in loss function for large weights)
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0

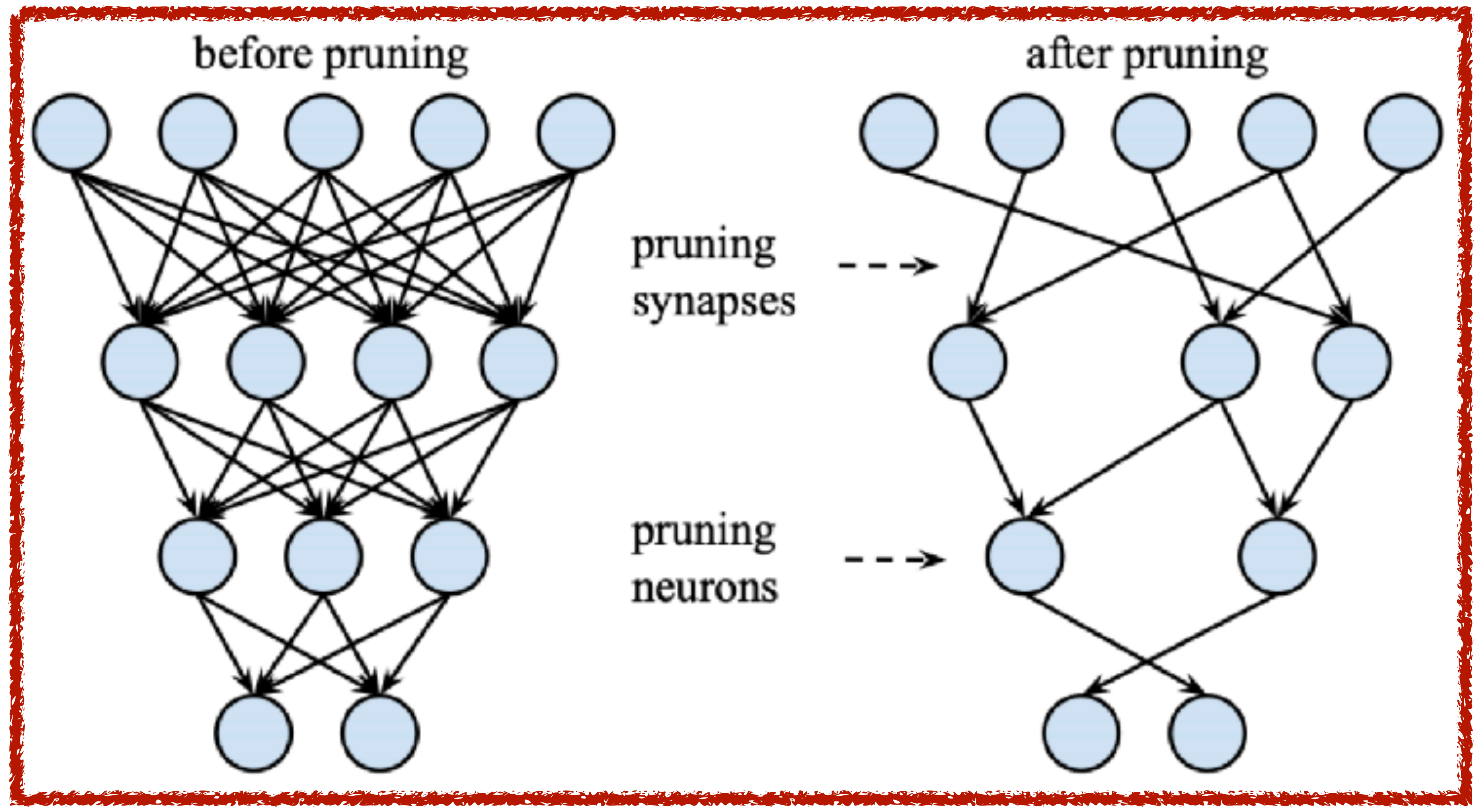


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda||\mathbf{w}||$$

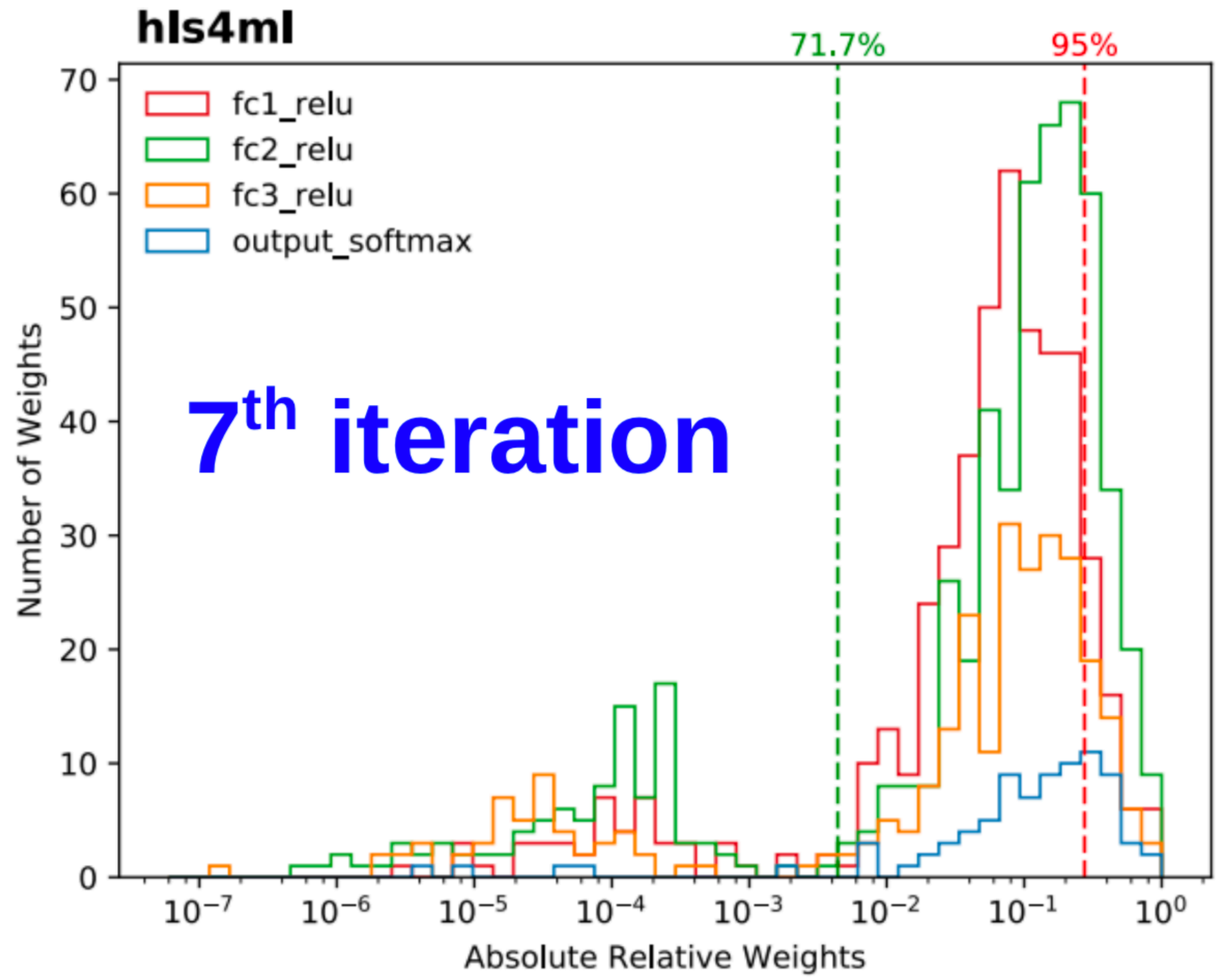


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining “best” way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
  - Use L1 regularization (penalty term in loss function for large weights)
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0

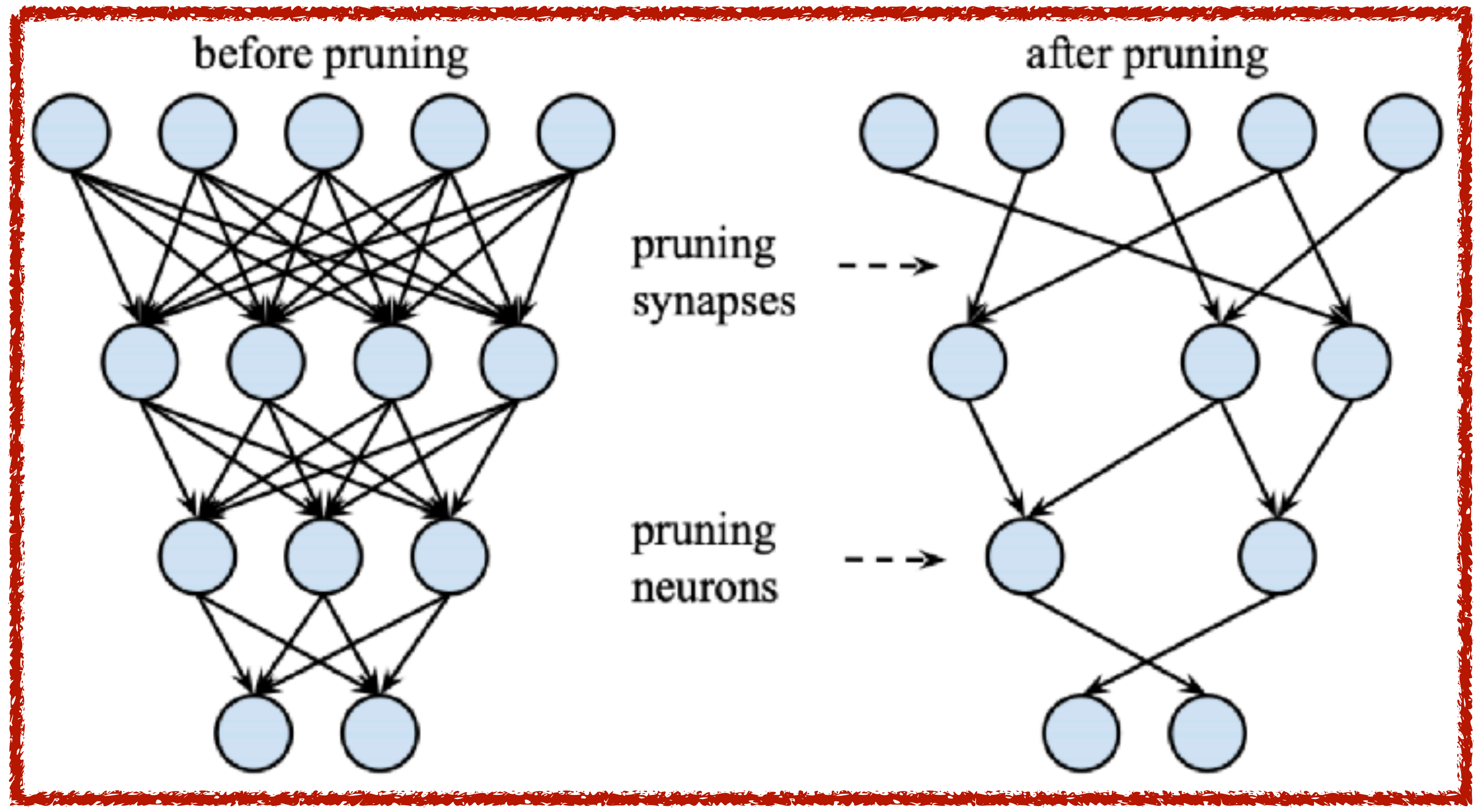


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$

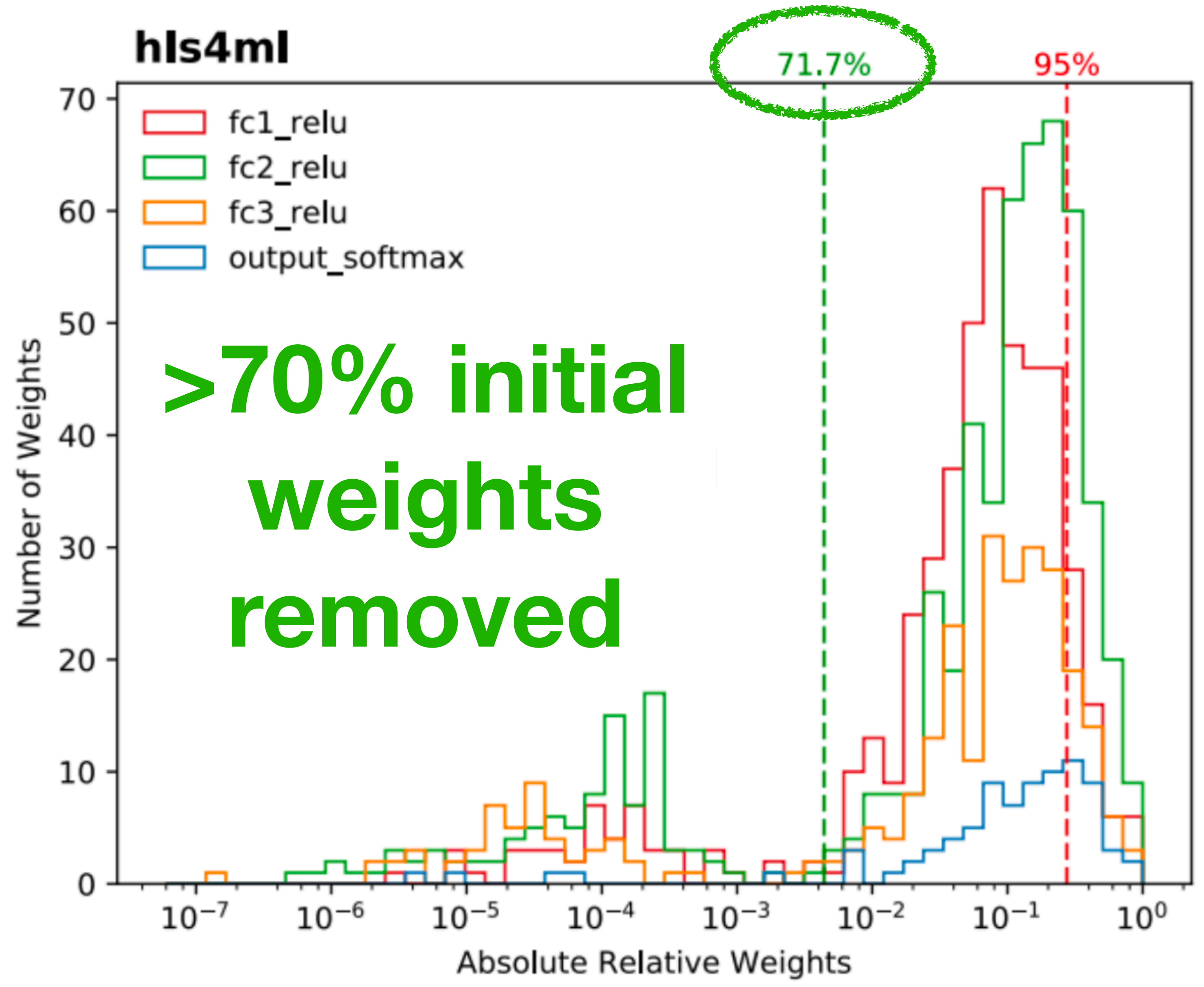


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
    - Use L1 regularization (penalty term in loss function for large weights)
    - Remove smallest weights
    - Repeat
- HLS automatically removes multiplications by 0

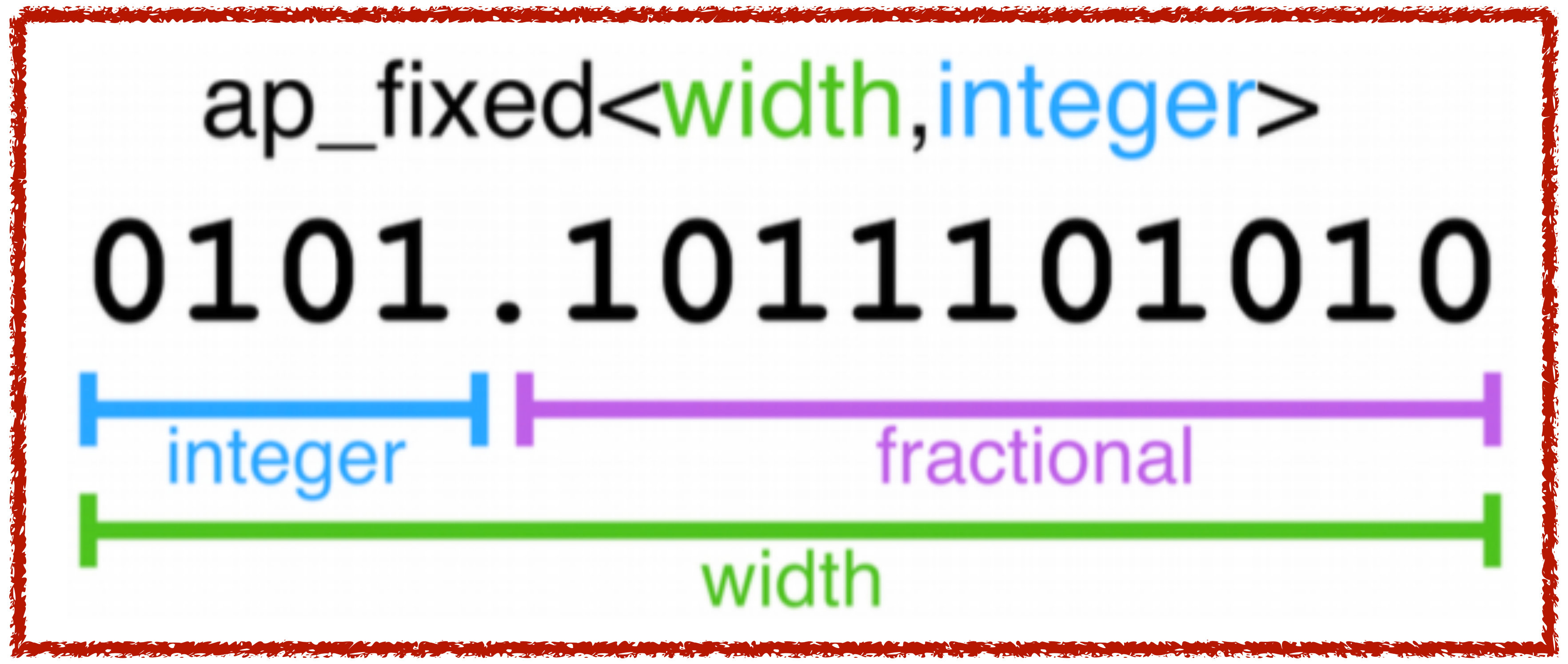


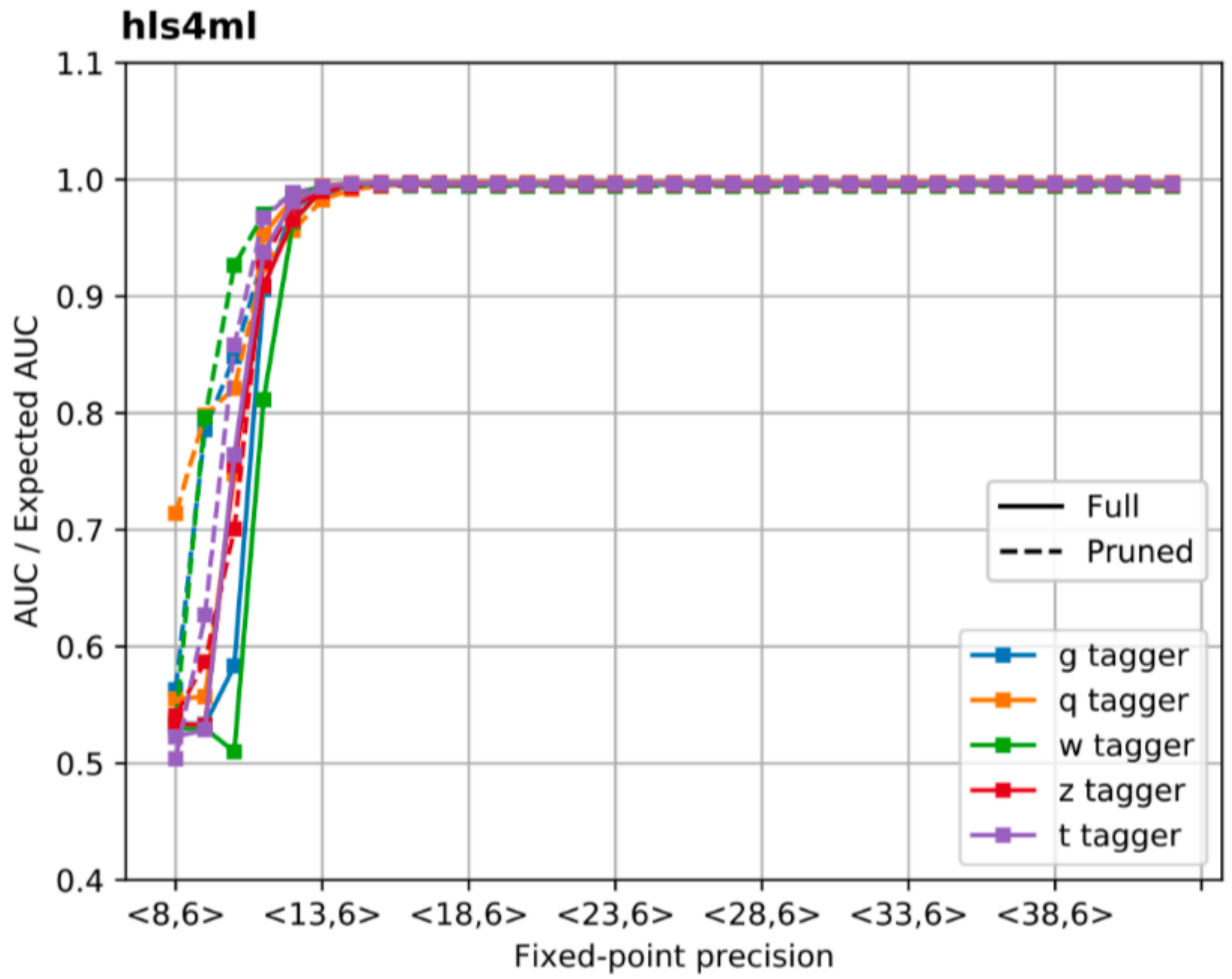
$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda||\mathbf{w}||$$

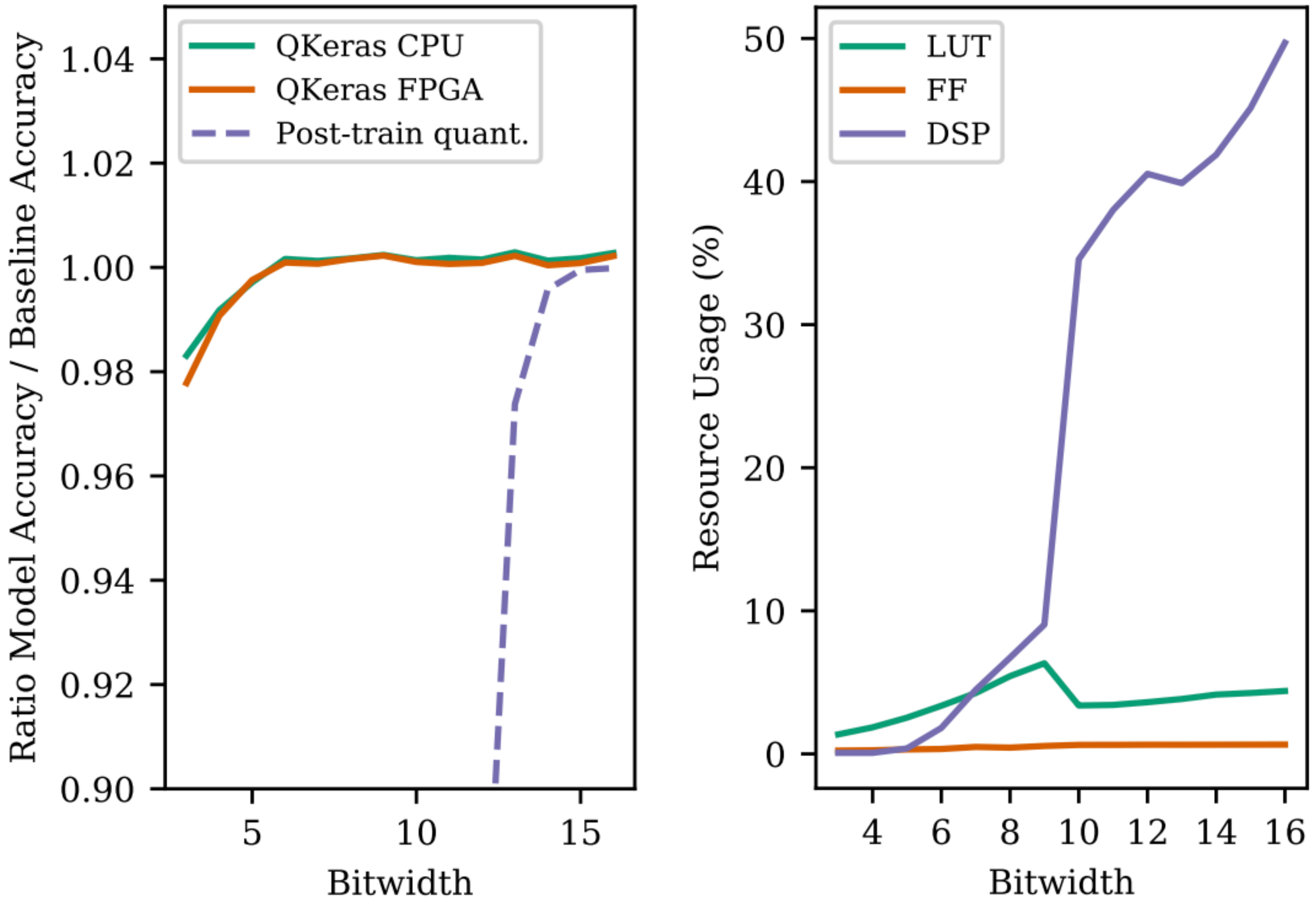


Quantization

- hls4ml uses fixed-point classes for all computations
- Precision can be adjusted as needed (impacts accuracy, performance, resources)
  - Can be combined with other customizations
- Binary & Ternary neural networks take this to very low precision: [2020 Mach. Learn.: Sci. Technol]
- Quantization-aware training - QKeras + support in hls4ml: [arXiv:2006.10159]

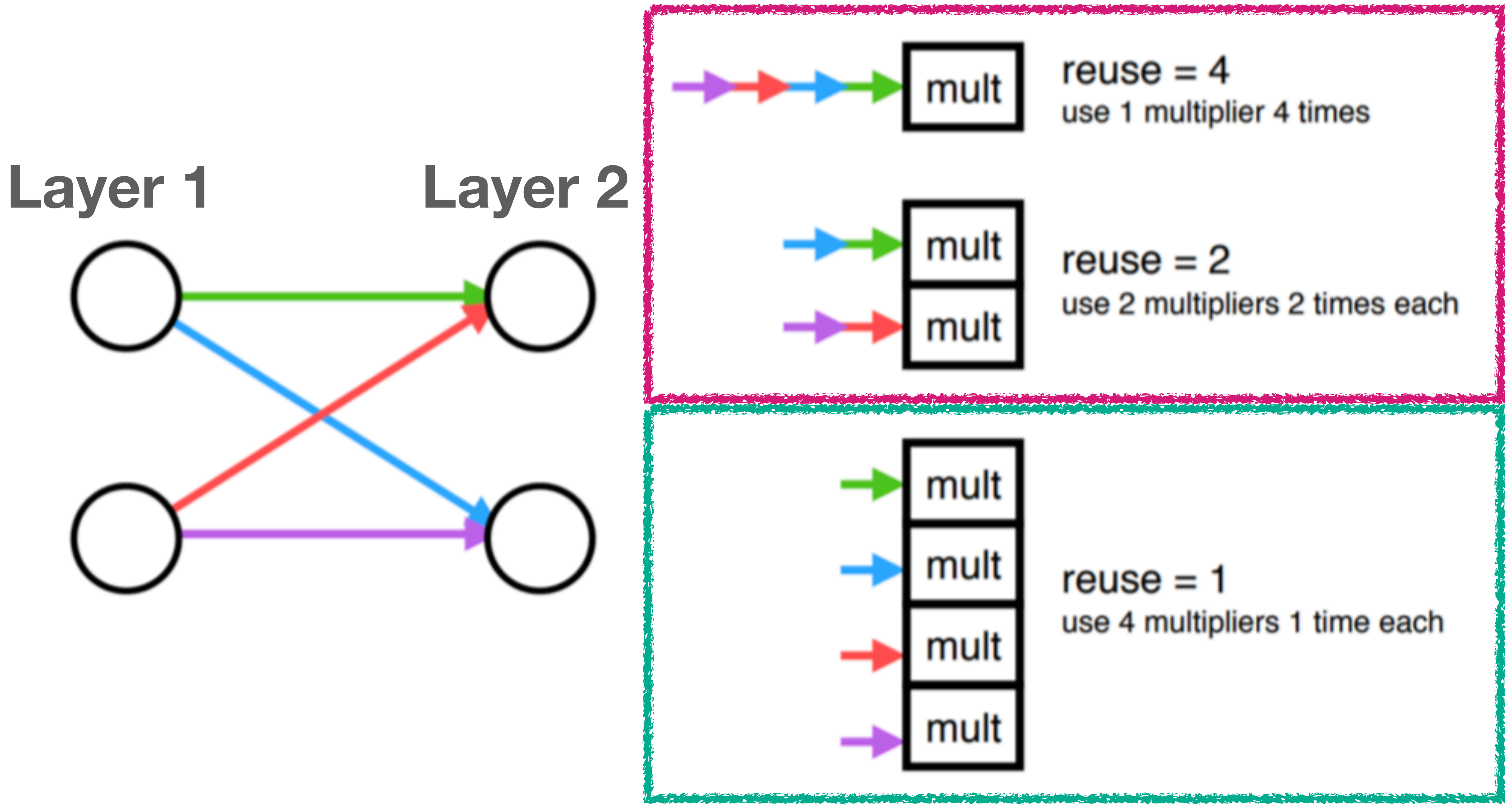


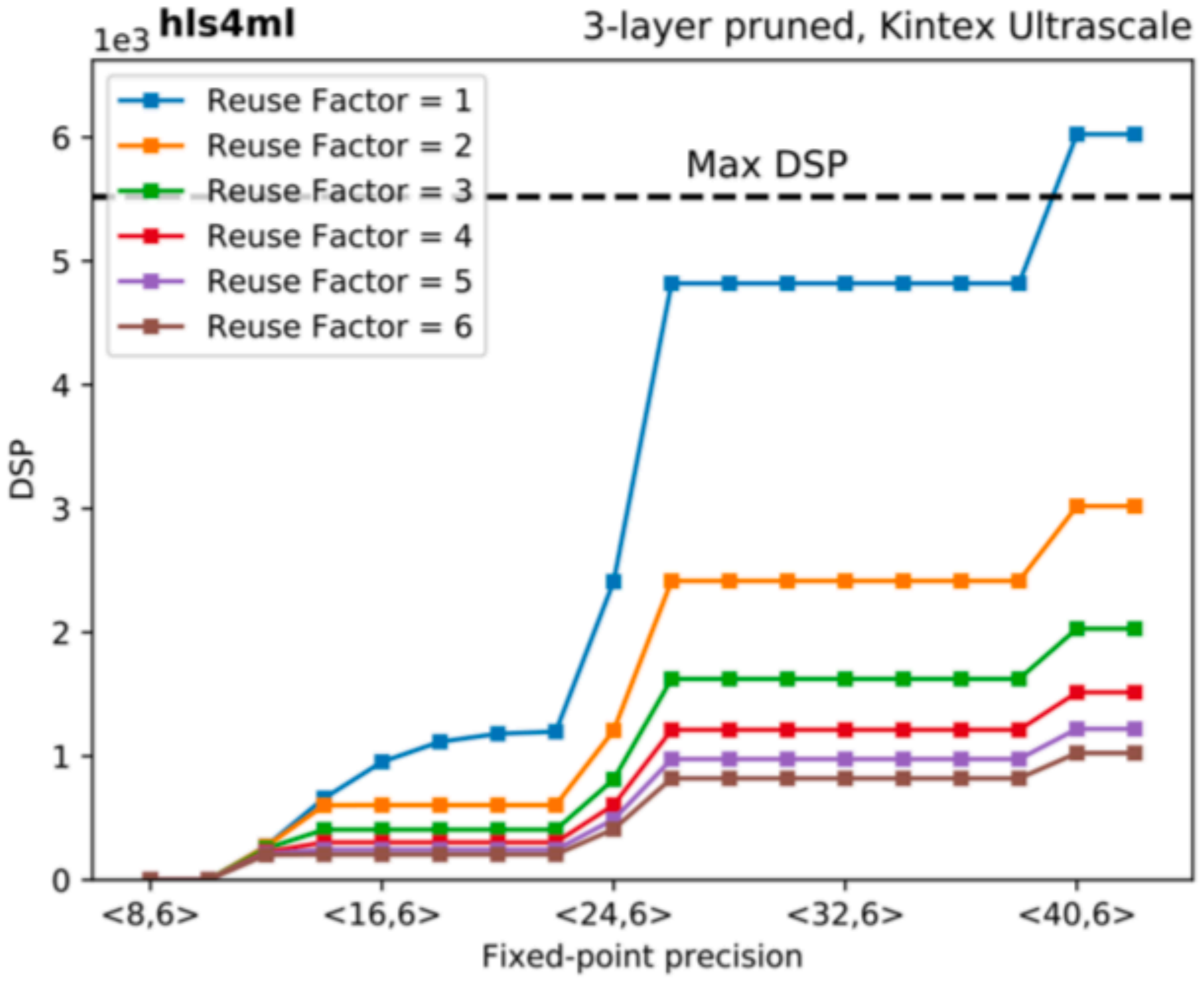


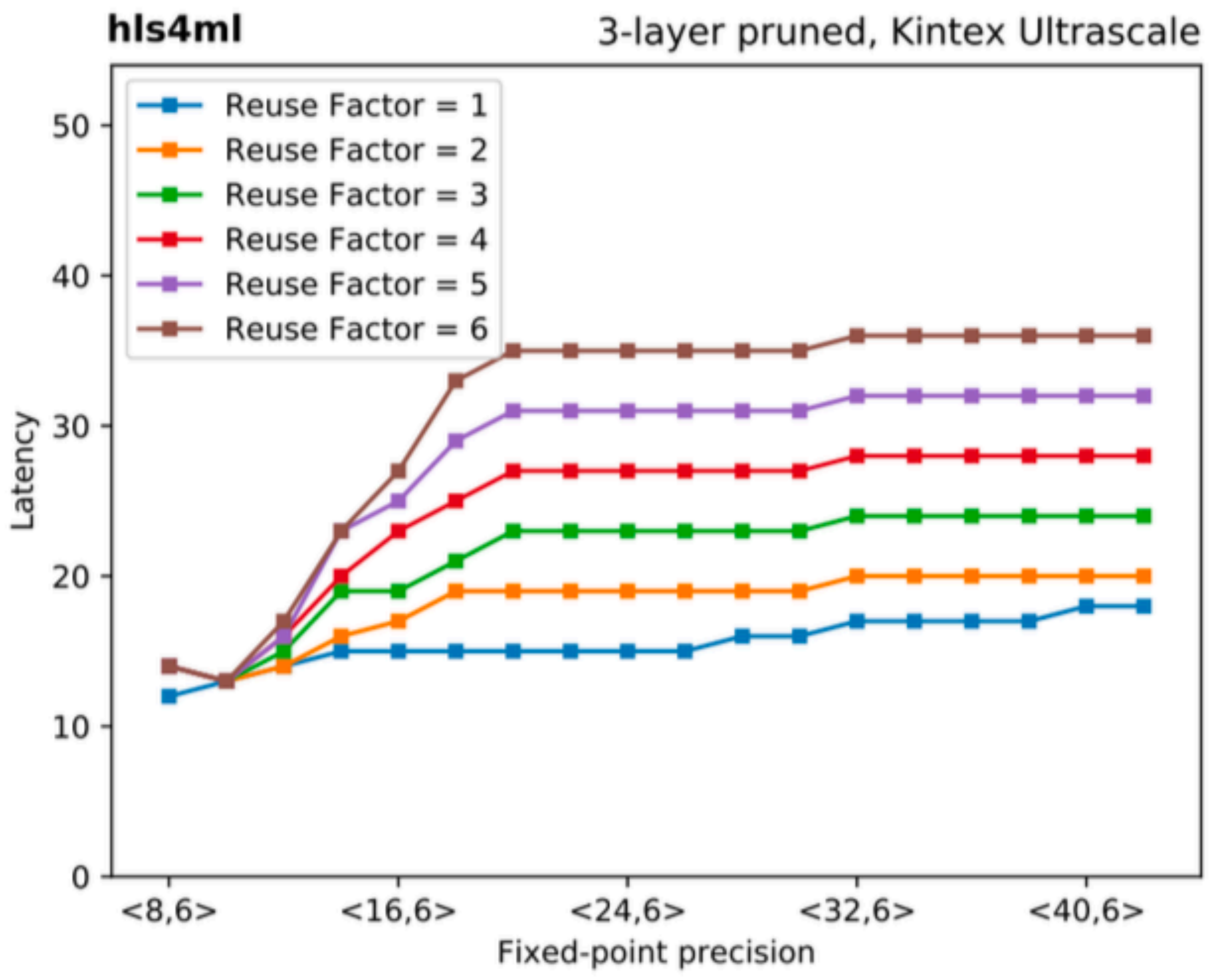


Reuse

- For lowest latency, compute all multiplications at once
  - Reuse = 1 (fully parallel) → latency = # layers)
- Larger reuse implies more serialization
- Allows trading higher latency for lower resource usage

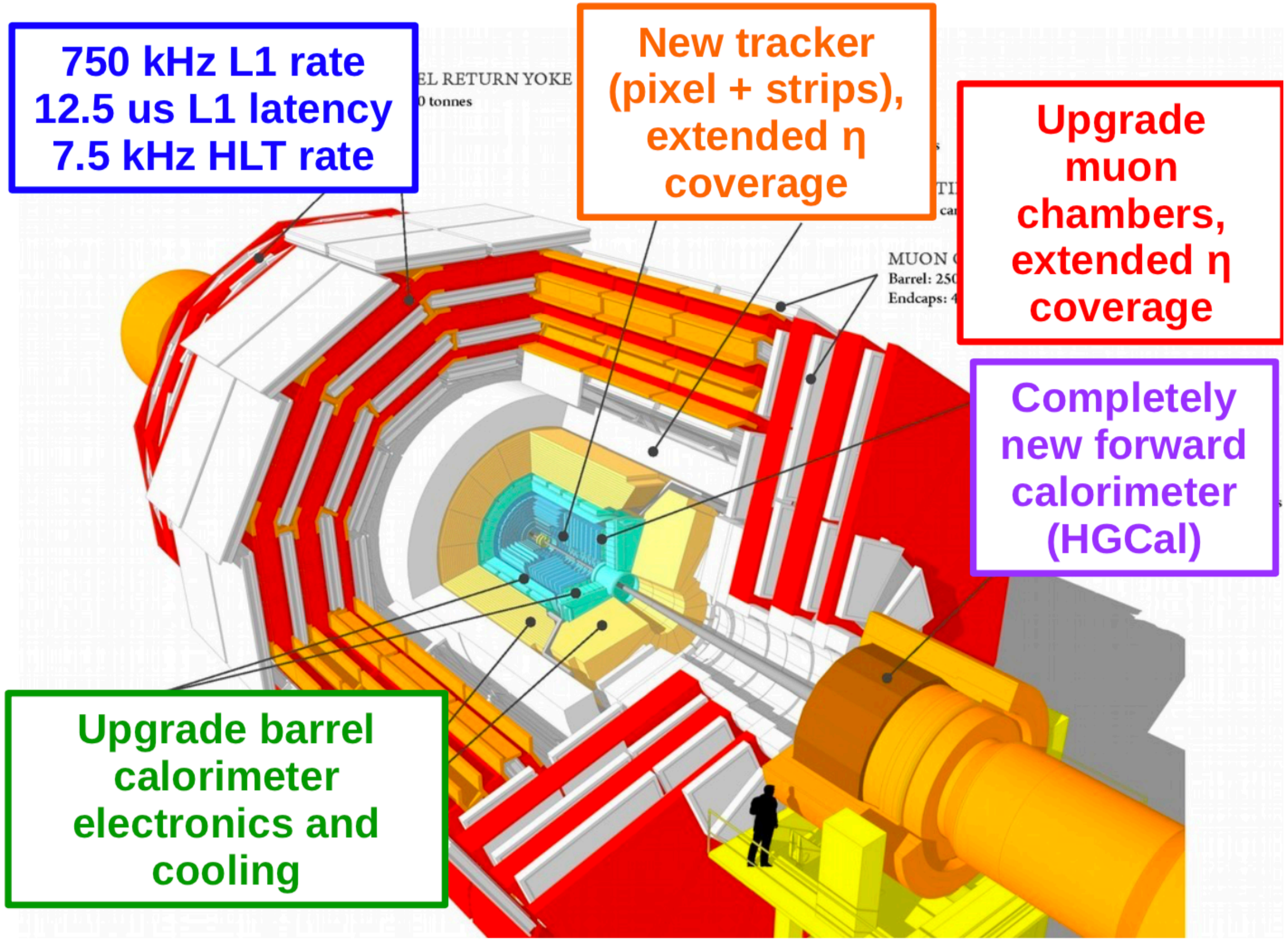






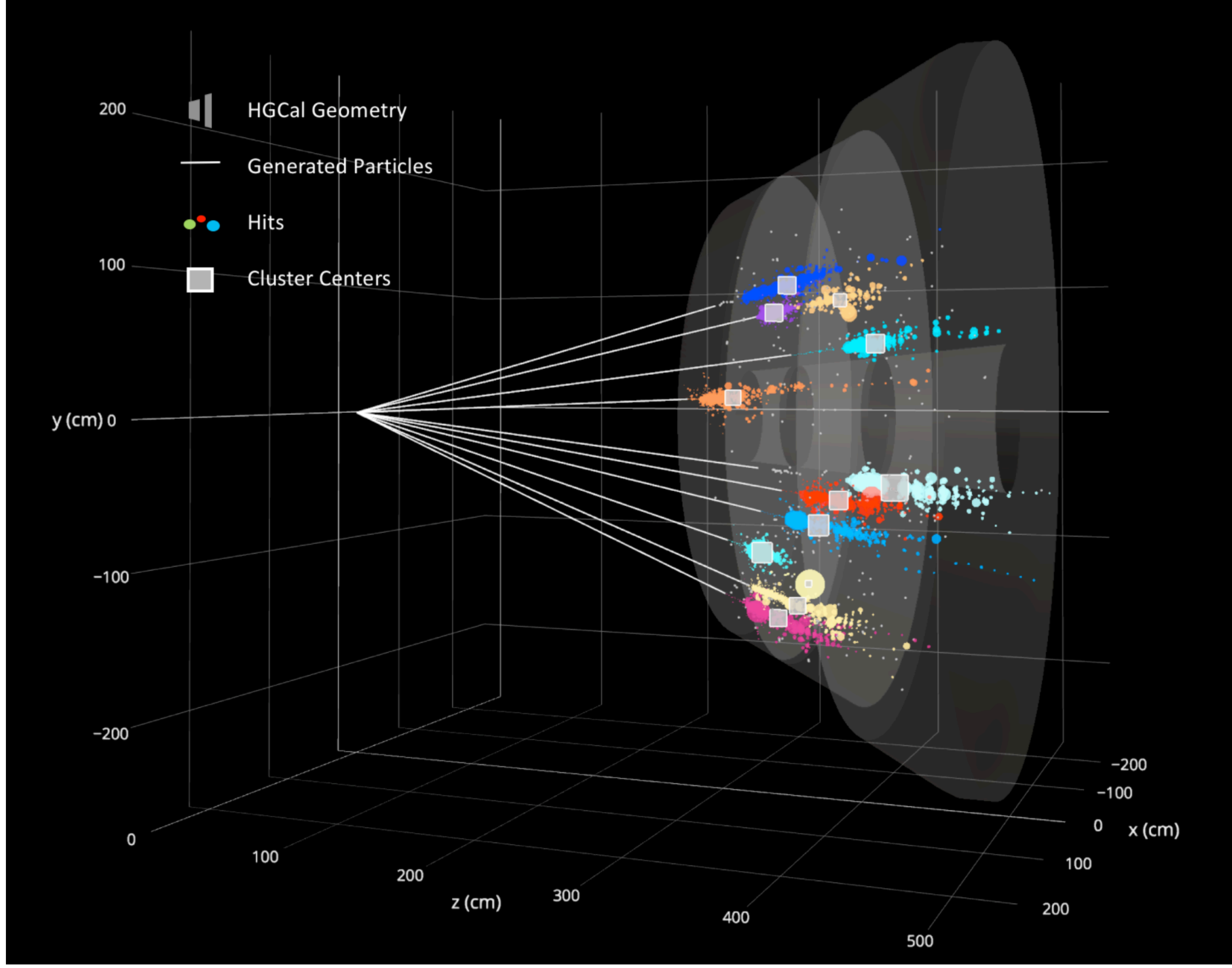
CMS Phase 2 Upgrade

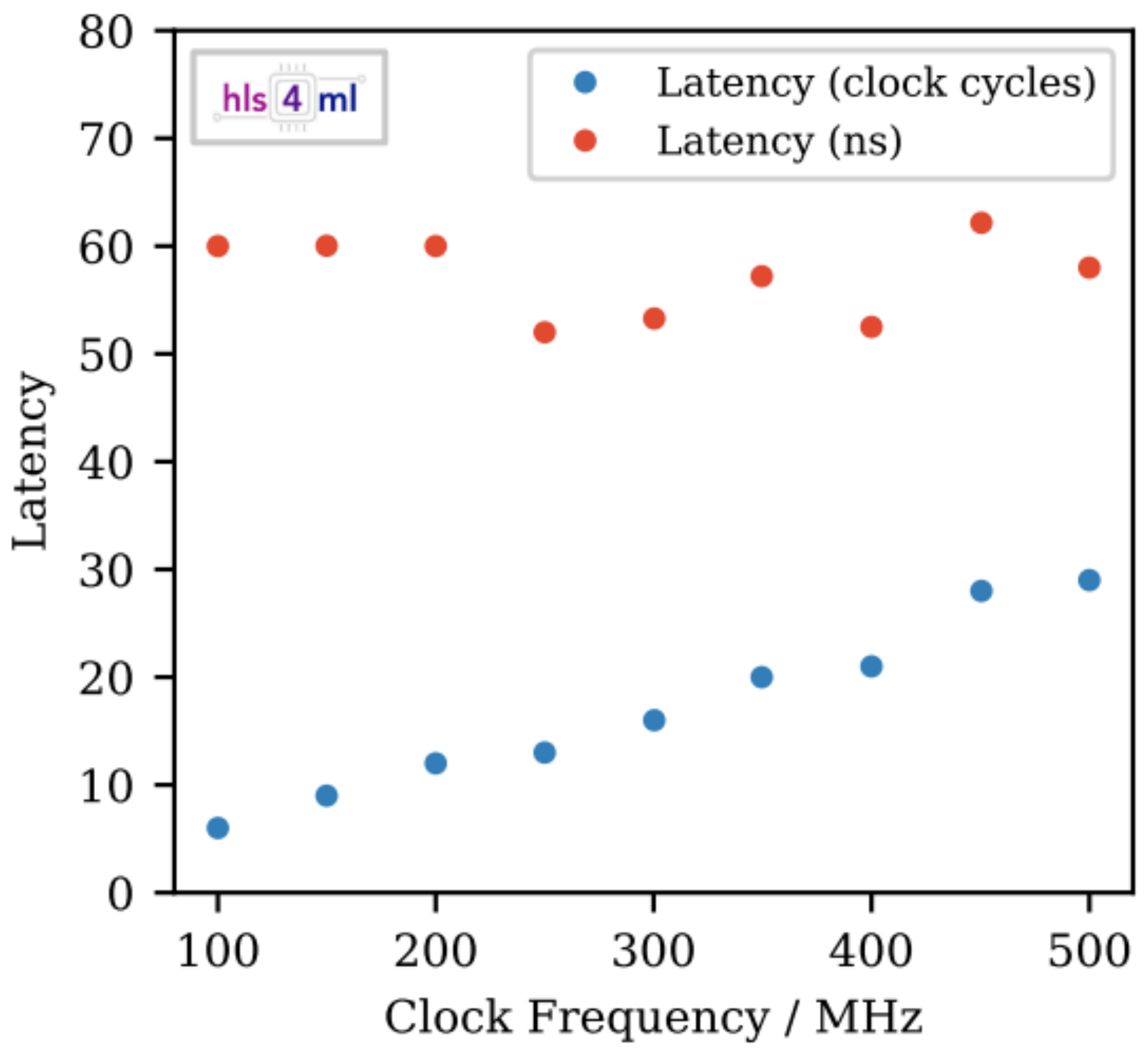
- For HL-LHC, CMS will upgrade every subdetector
- Trigger upgrade will provide strip tracking information at L1
  - L1 will have (almost) full detector information
- Forward calorimeter completely upgraded with 3-dimensional readout
- Many new ML algorithms are looking to take advantage of this upgrade

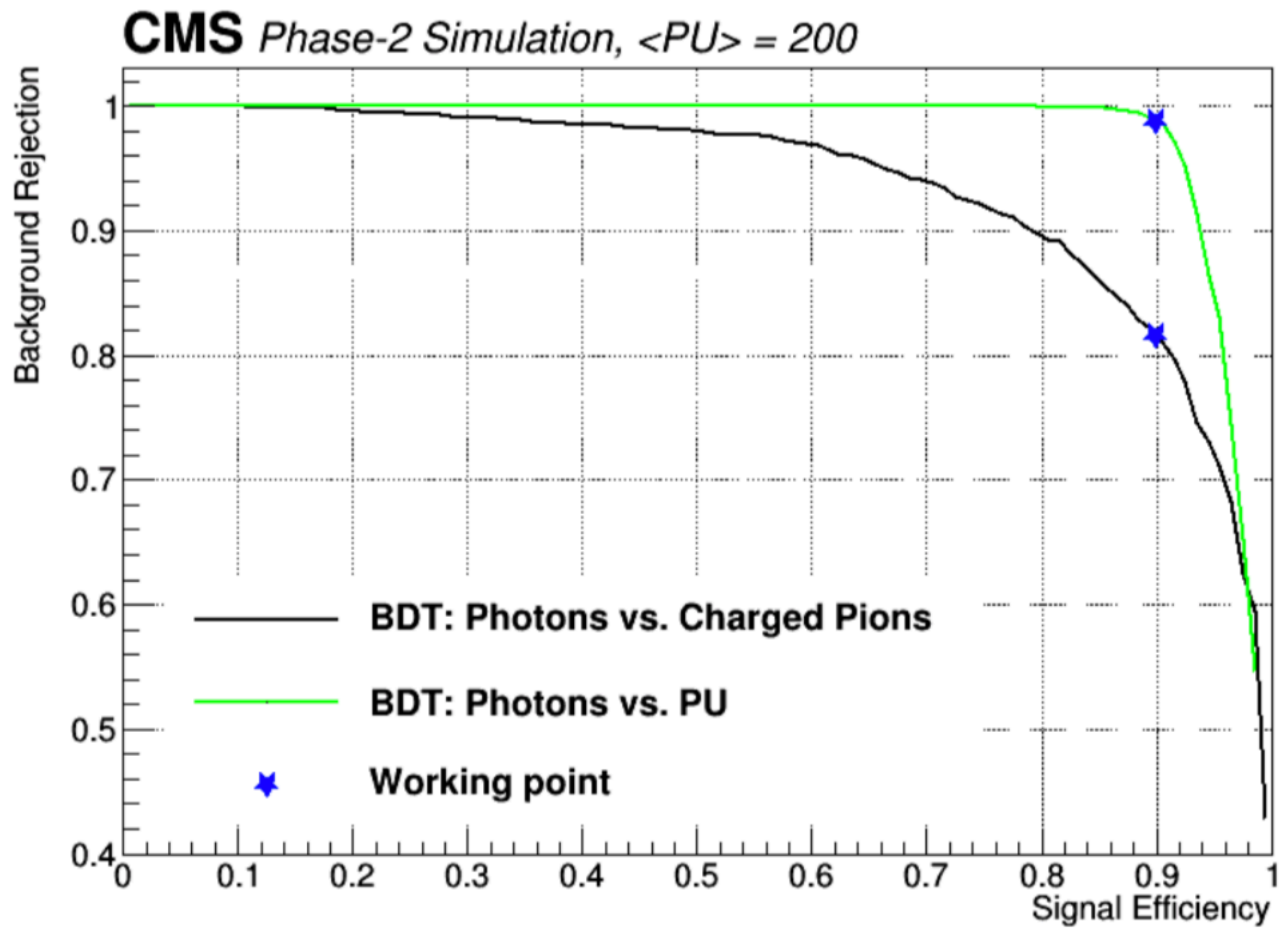


HGCal PU ID

- CMS upgrade will install entirely new high granularity calorimeter
    - 3-dimensional, silicon-based
- Without ID, extremely large number of clusters at 200 pileup
- BDTs developed to reject PU, discriminate between γ and π
    - Highly efficient vs PU
- hls4ml supports BDTs through Conifer [JINST 15 P05026 (2020)]

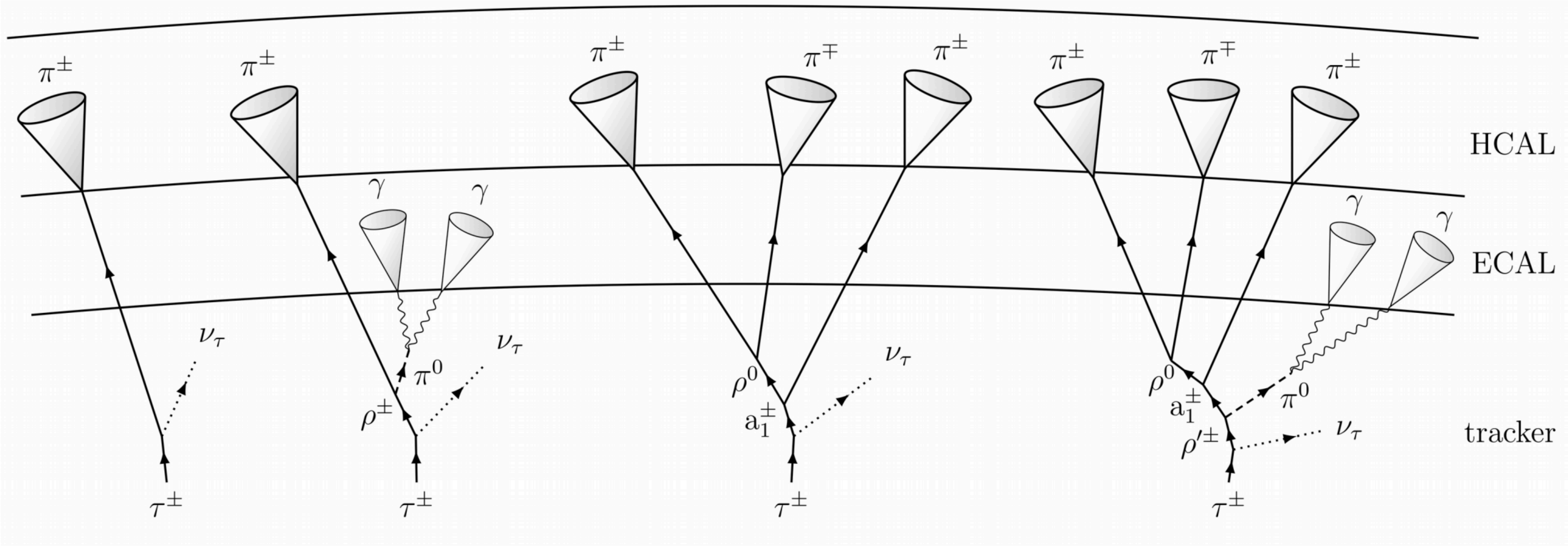


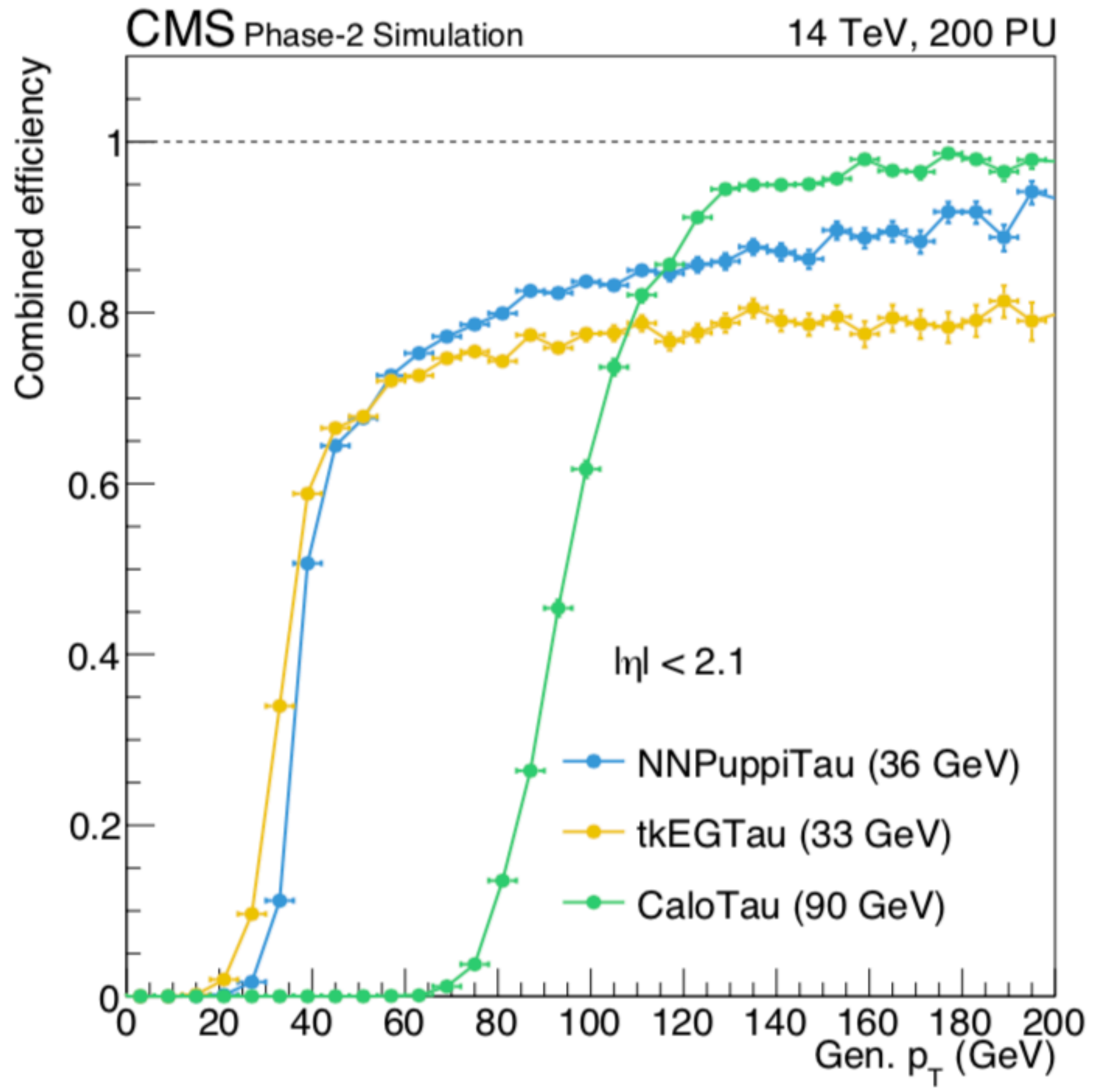


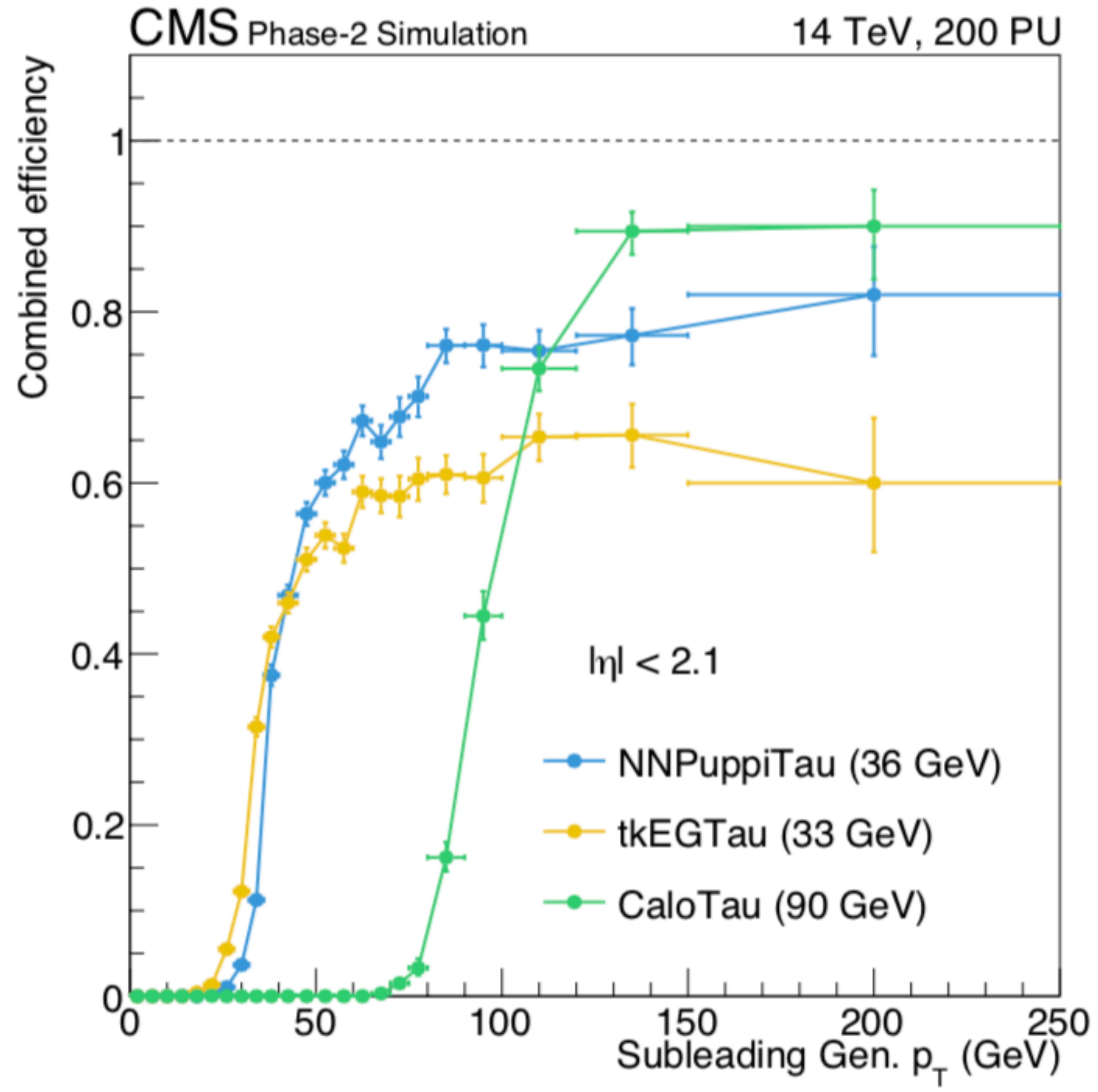


L1 Tau ID

- Tracks at L1 allow more sophisticated tau identification
- Offline-like algorithm combines e/γ clusters with tracks to construct known tau decays
- Small MLP (3-layer) trained to identify taus using particle candidates
  - Latency of 36 ns
  - Benchmark L1 tau algorithm for CMS
  - Improved performance potentially with CNNs/RNNs

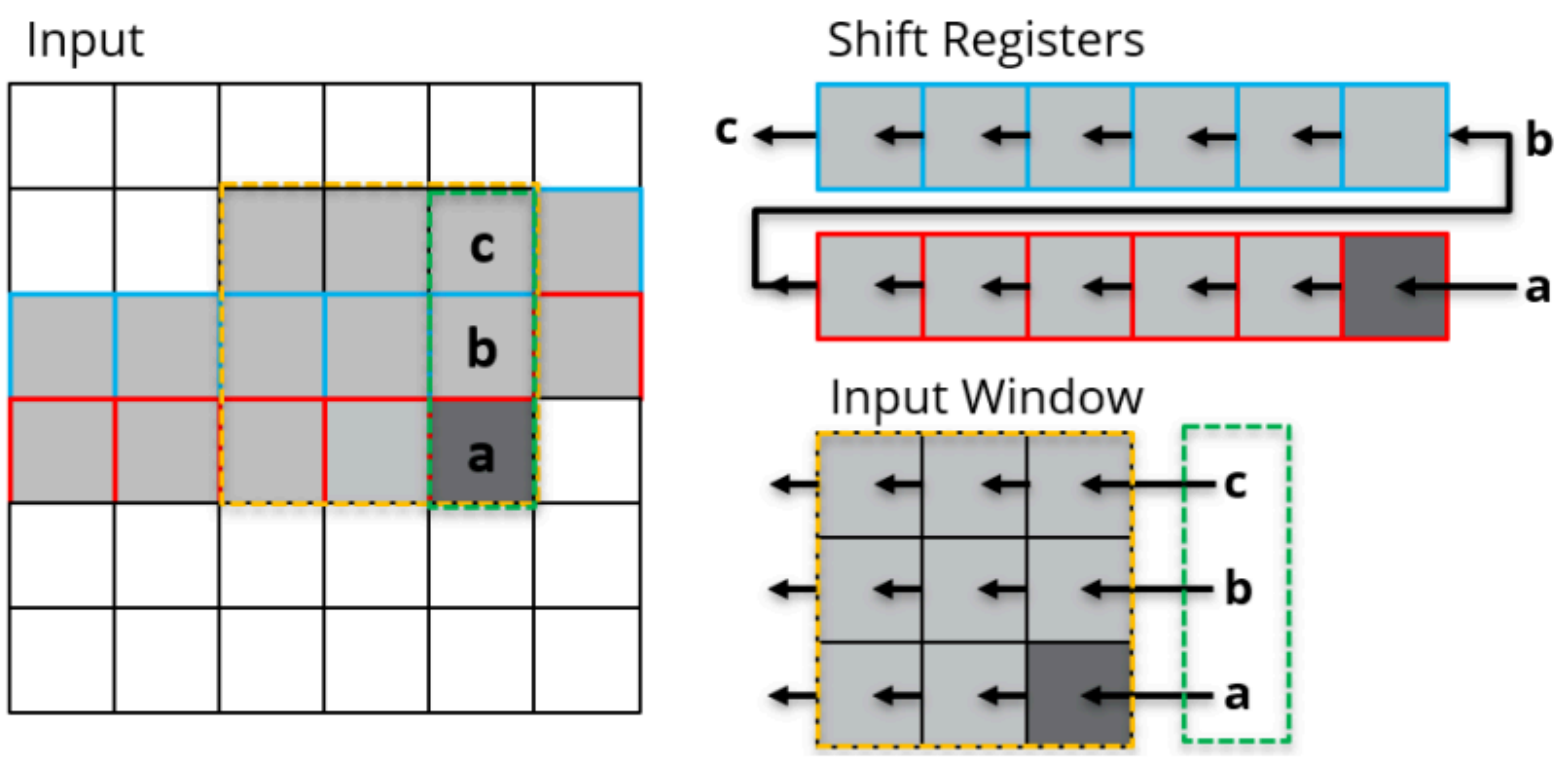


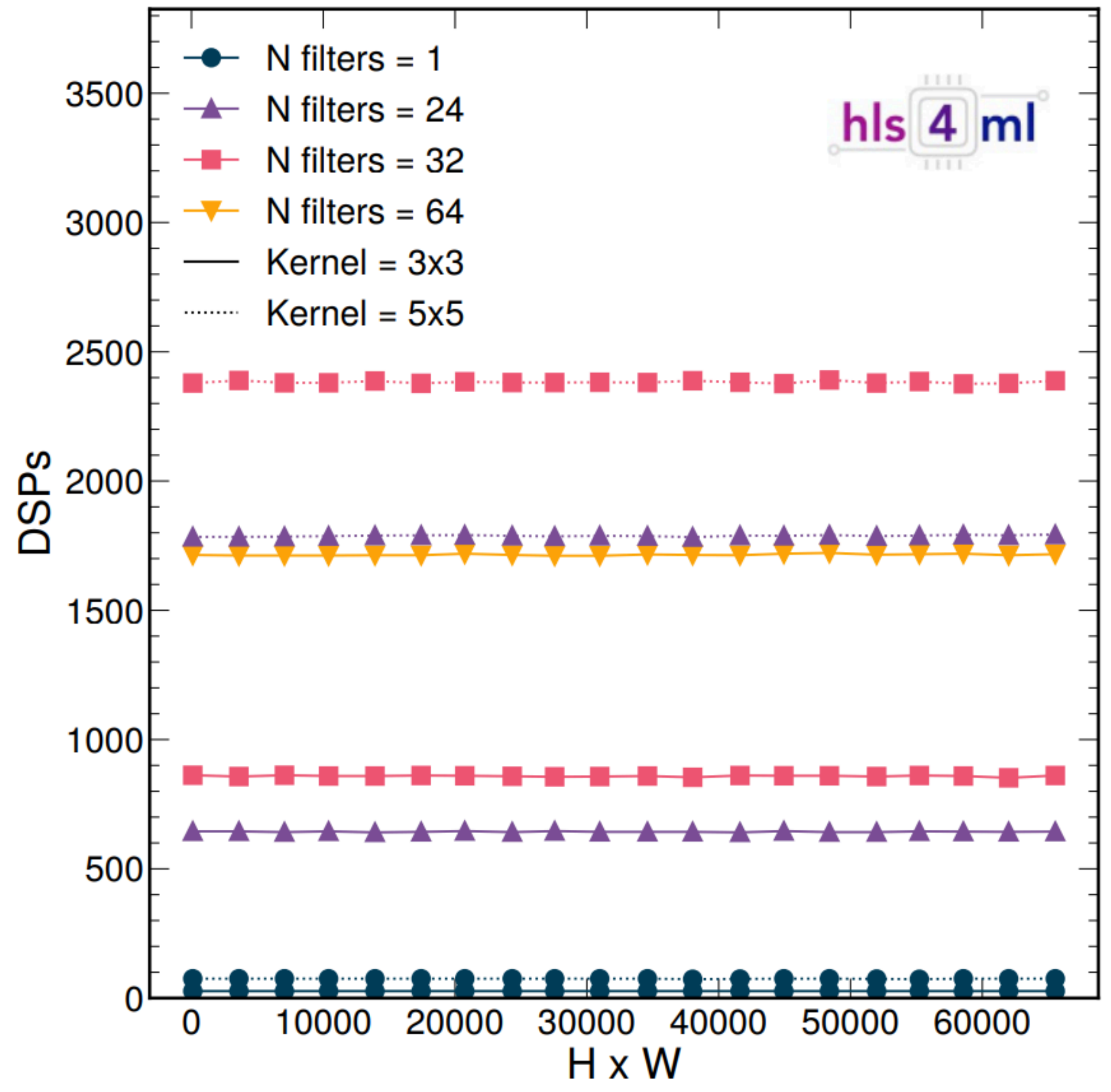


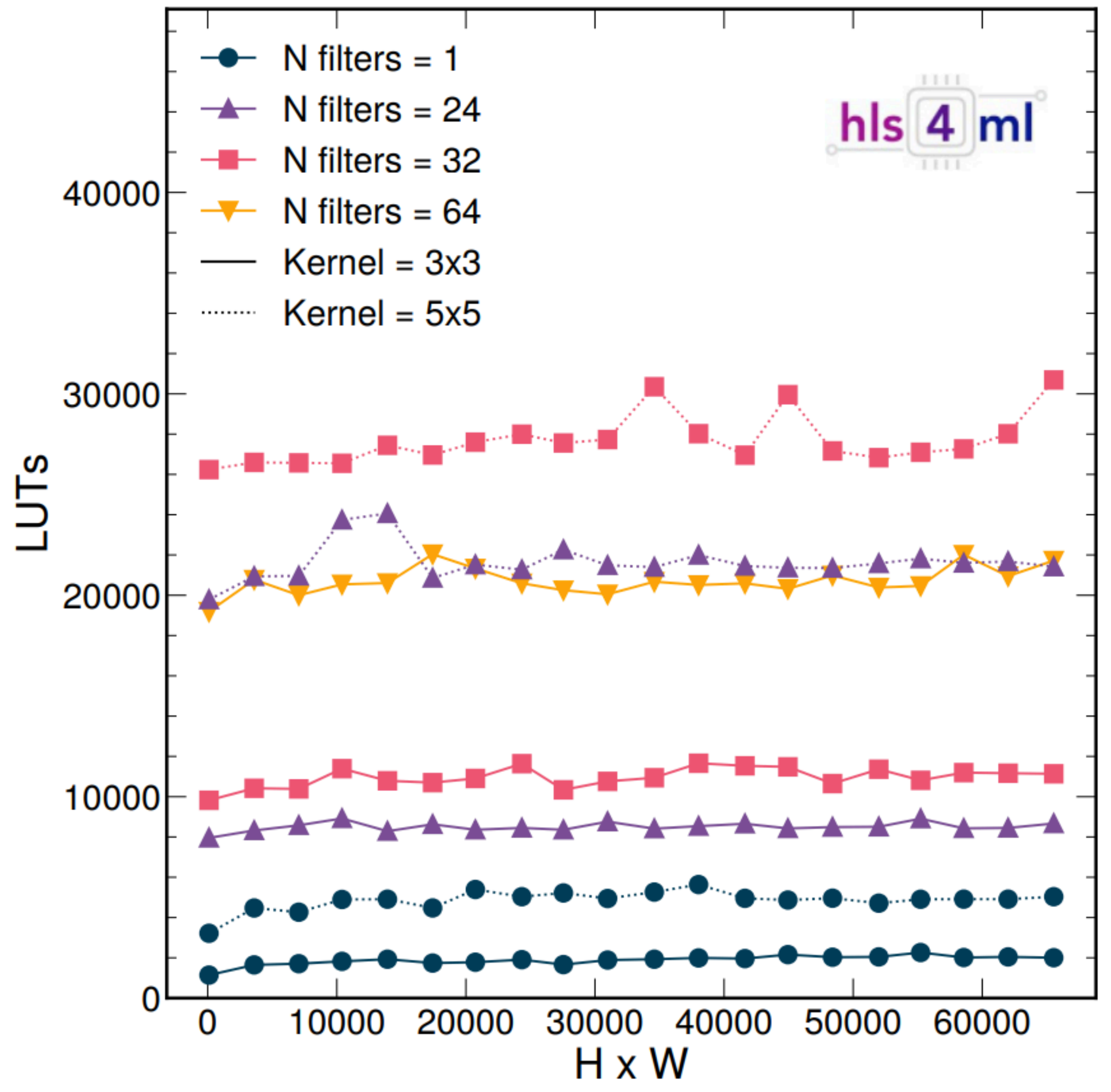


CNNs

- Special adjustments necessary to implement convolutional networks on FPGAs
  - HLS struggles with very long (nested) loops
- hls4ml is now able to synthesize large CNNs with good resource scaling
- Further optimizations possible for lower latencies
- arXiv:2101.05108

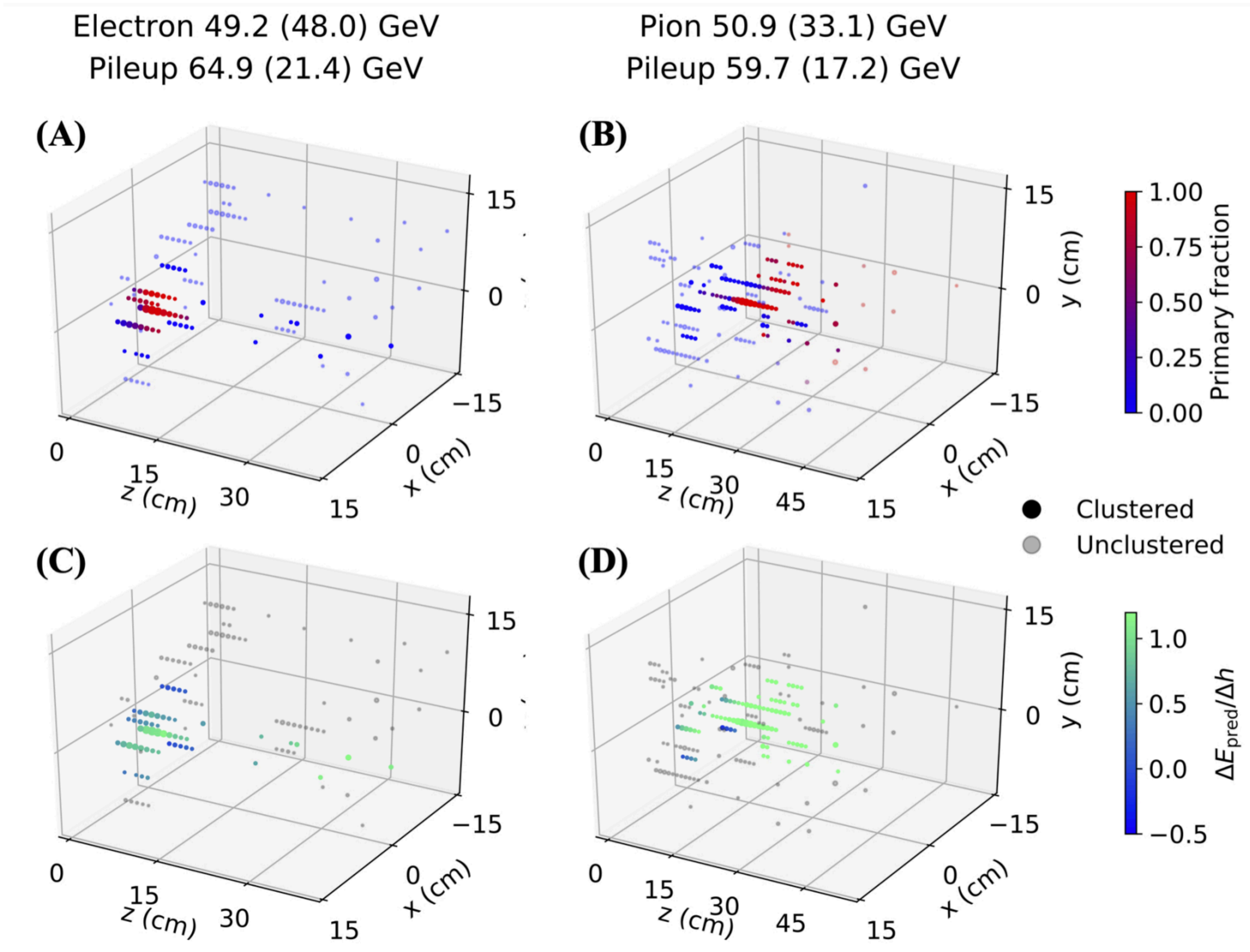


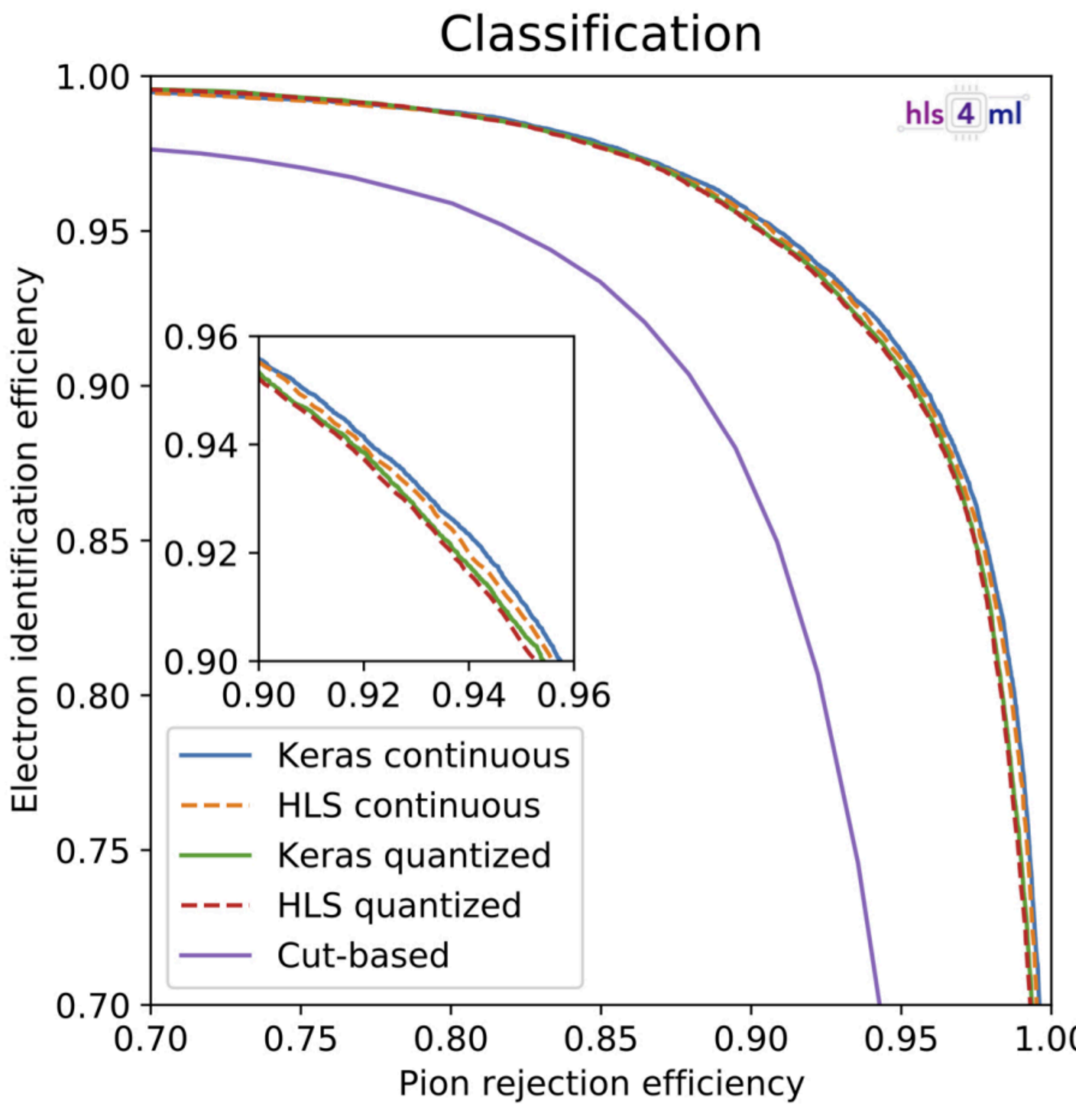


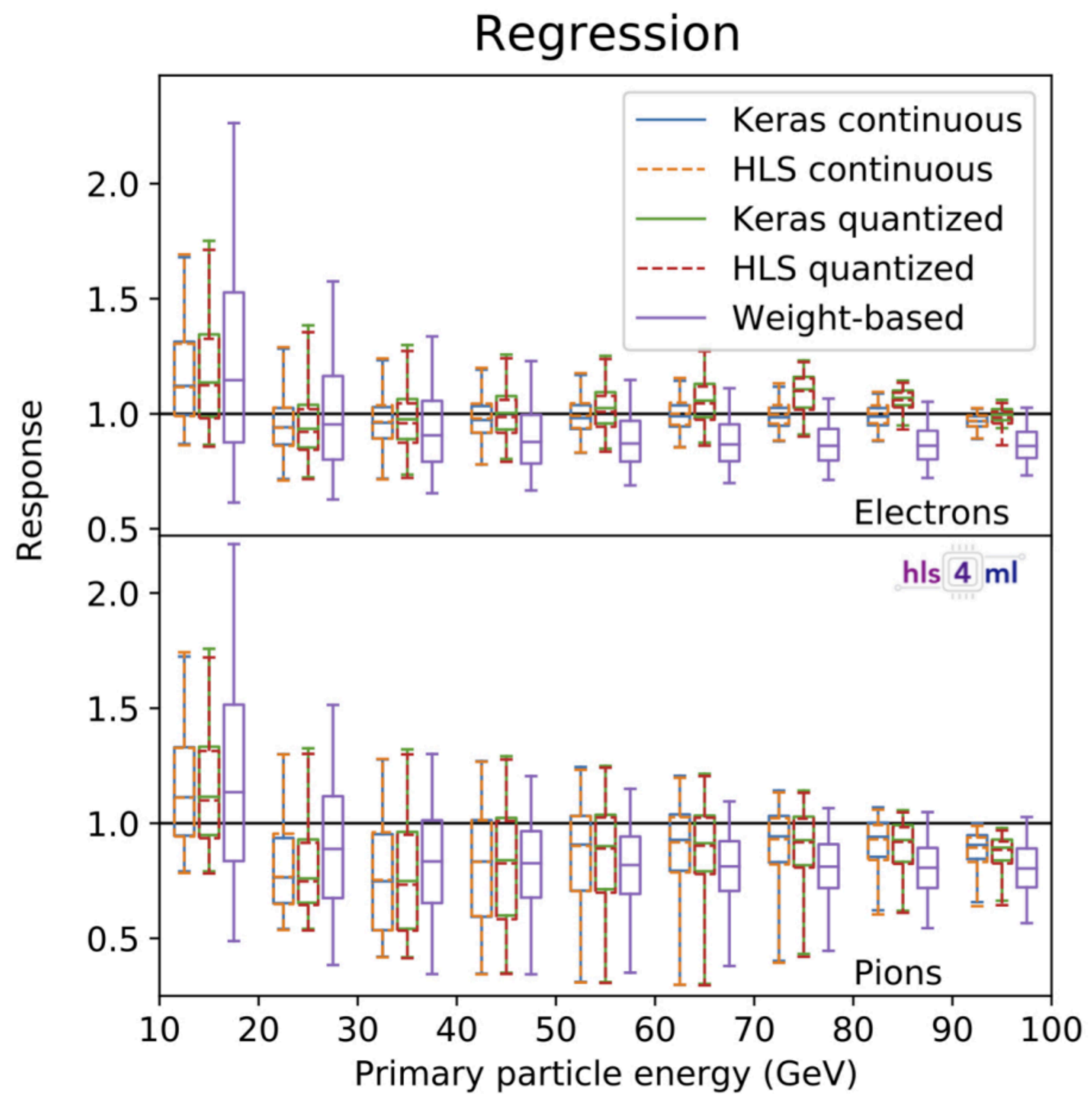


GarNet

- Graph networks have become very popular for complex geometric problems
  - Iterative nature difficult for FPGAs
- Modified GarNet architecture implemented in hls4ml
  - arXiv:2008.0360
- Model developed for HGCal cluster ID and energy regression
  - Able to run in under 1 μs, fit within a single VU9P SLR

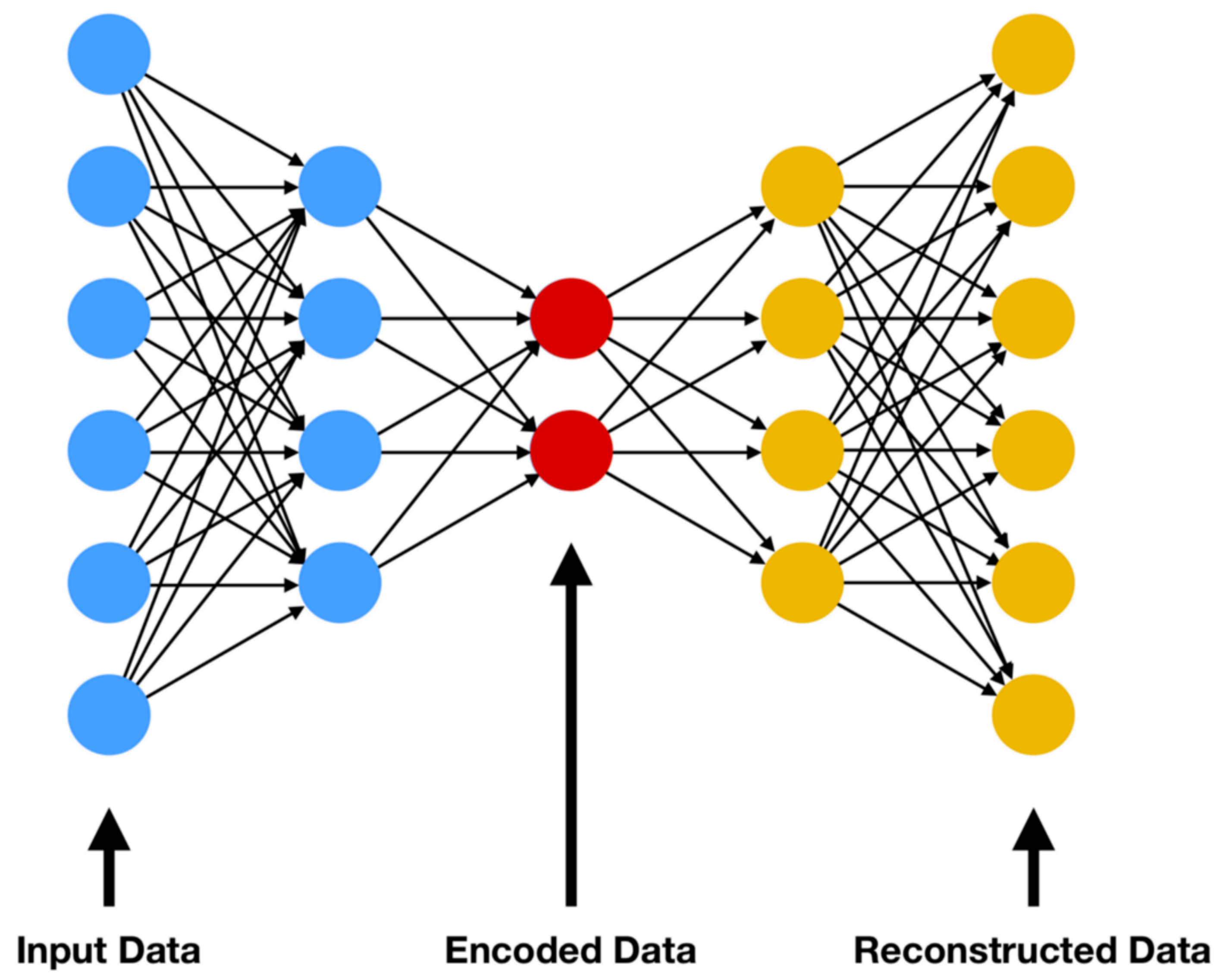


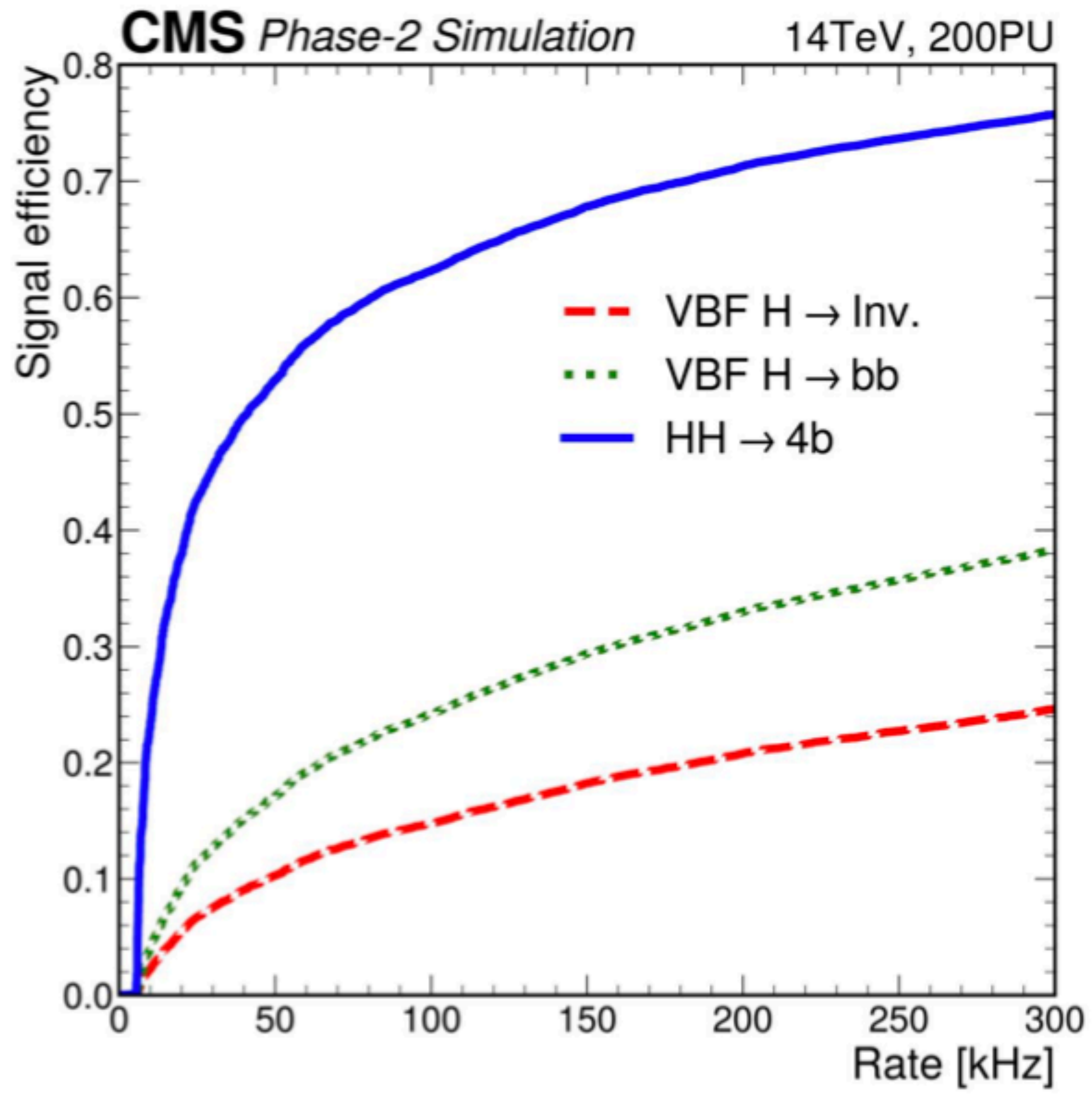




Other examples

- Many other possibilities for ML in trigger (and ongoing development)
- One highlight: New Physics auto encoder
  - 8-layer dense network
  - Trained only on minimum-bias background events
  - Sensitive to anything that doesn't look like standard background
  - Can be run in ~100 ns





Conclusions

- Machine learning is an increasingly important part of HEP workflows
    - Full advantage of the gains from ML requires integration with trigger/readout systems
- Conventional CPU inference can only be done so fast
    - Alternative architectures can offer major speedups (FPGAs, GPUs, others)
- hls4ml opens up possibilities for low latency ML inference on FPGAs
- Many possibilities for ML applications in trigger/readout on the horizon

BACKUP