Graph neural networks for FPGAs

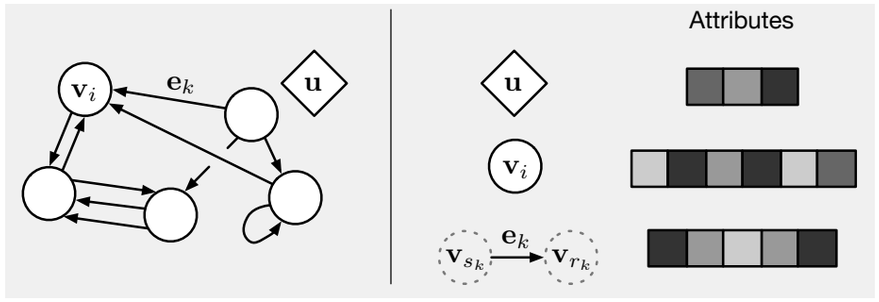
CTD2020 (~~Princeton, NJ~~ Zoom and Mattermost)

G. Cerminara, A. Gupta, J. Kieseler, V. Loncar, J. Ngadiuba, M. Pierini,
M. Rieger, S. Summers, G. Van Onsem, K. Wozniak (CERN)
G. Di Guglielmo (Columbia U.)
S. Jindariani, M. Liu, K. Pedro, N. Tran (FNAL)
E. Kreinar (HawkEye 360)
P. Harris, D. Rankin (MIT)
J. Duarte (UCSD)
Z. Wu (UIC)
Y. Iiyama (U. Tokyo)

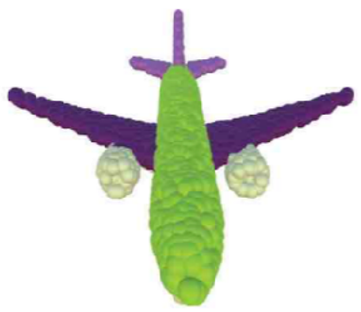
Graph Neural Network

= NN on sets of vertices ($\mathscr{V}$) and edges ($\mathscr{E} \in \mathscr{V} \times \mathscr{V}$)

Input: vertices and edges with intrinsic features
Output: vertex / edge labels, global properties of the graph, etc.

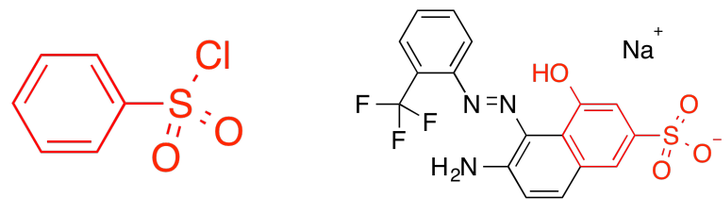


1806.01261



1801.07829

"Toxic fragments"



1509.09292

Interesting properties of graph-represented data:

- List of vertices is variable-length and unordered
- Edges can encode geometry (distance) and topology

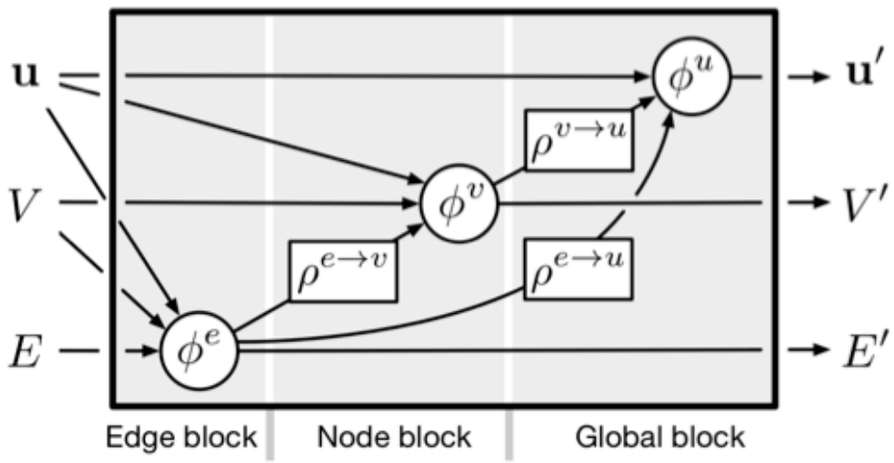
→ GNN is a natural tool for machine learning over sparse data
e.g. sets of points sampled in space (← HEP detector hits)

Graph Network

Formulated in Battaglia et al. (1806.01261)

Network built from graph-to-graph feature transformation blocks

"GN block" =

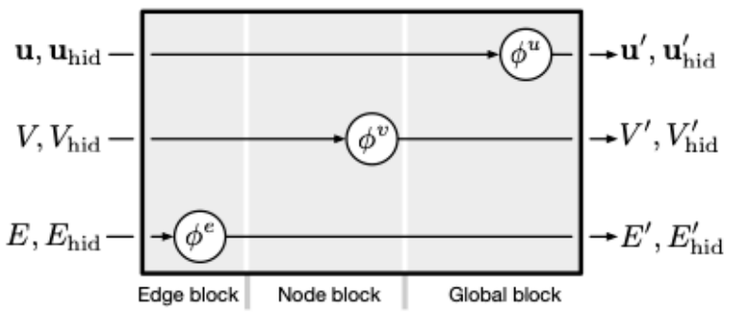


Computation per edge

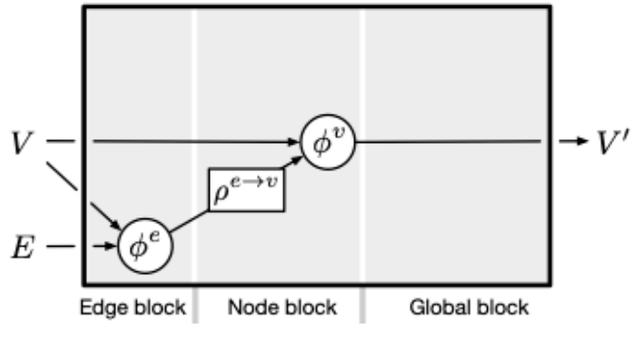
Computation per node

"Aggregate" computation

Many GNN algorithms can be understood as variations of GN:



(b) Independent recurrent block



(d) Non-local neural network

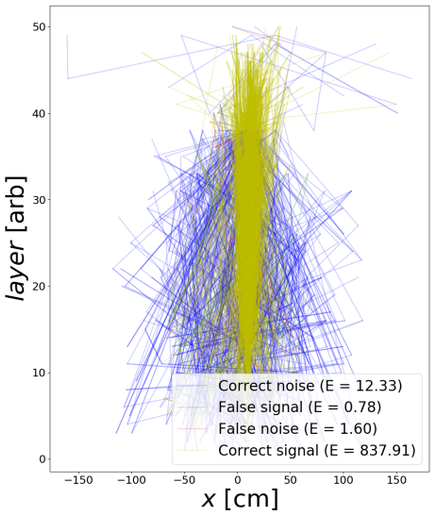
GNNs in HEP

Check CTD2020 talks: 25, 34, 45, 46, 48, 49

Tracking = Clustering = Pattern recognition on sparse data

More so with next-gen high-granularity 3D calorimeters

- Hits → vertices
- Tracking / clustering → Prediction of correct edges from vertex features (x, y, z, t, E, etc.)
- PID / energy regression etc. → Prediction of global graph features given a connected graph



NeurIPS ML4PS 2019 83

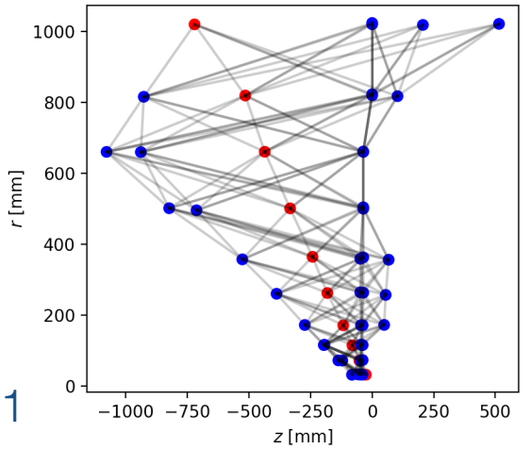
Formulation is detector-agnostic

→ Good base algorithm can be used for all tasks

GNN usefulness well-proven in event reconstruction for LHC and neutrino experiments

Next front: fast GNN inference for triggers (L1T)

i.e. GNN as firmware on FPGA



1810.06111

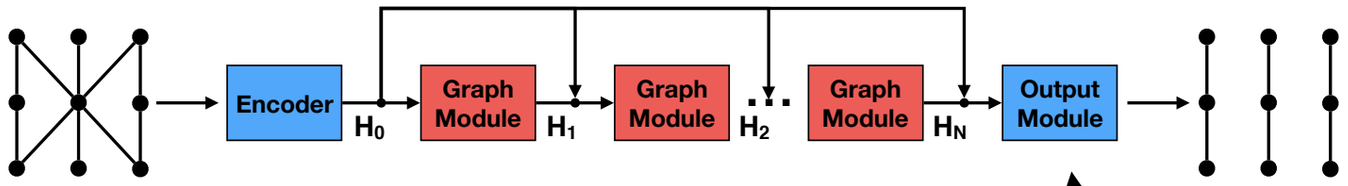
GNN for L1T: General requirements

- Fast
  - Example: Phase 2 (HL-LHC) CMS L1T total latency = 12.5 μs
  - Local reconstruction / PID algorithms need to run in < 1 μs
- High throughput
  - Example: 40 MHz bunch crossings at LHC
    → As a system, need to accept one event every 25 ns
  - In practice, can be time-multiplexed O(10) times
    → Individual components may accept one event per ~250 ns
- Small
  - Algorithm must fit on one chip

GNN in FPGA: Challenges

GNN in FPGA: Challenges

NeurIPS ML4PS 2019 83

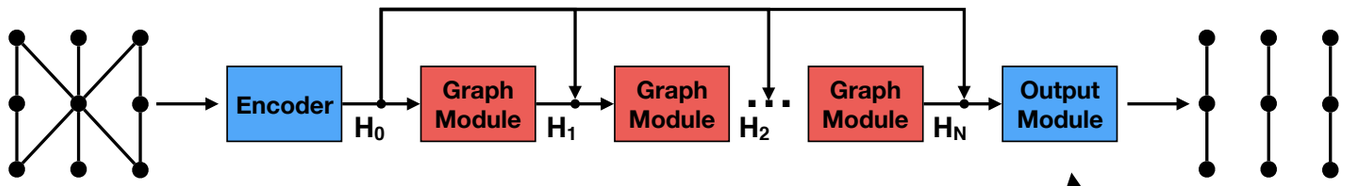


- Number of operations
  - For realistic problems, V ~ $O(10^{3\text{-}4})$, E ~ $O(10^{4\text{-}5})$ or higher
  - Typically, transformations are multilayer perceptrons
    → Number of multiplications explodes very quickly
  - Example: Exa.TrkX GNN segment classifier (not optimized for resource)
    $O(10^6)$ / $O(10^7)$ multiplications per vertex / edge
    ⇒ 2.6×10^{10} multiplications[1]

[1] https://gist.github.com/jmduarte/c8783a4c9efa6cfd6c0638c03d3bd561

GNN in FPGA: Challenges

NeurIPS ML4PS 2019 83

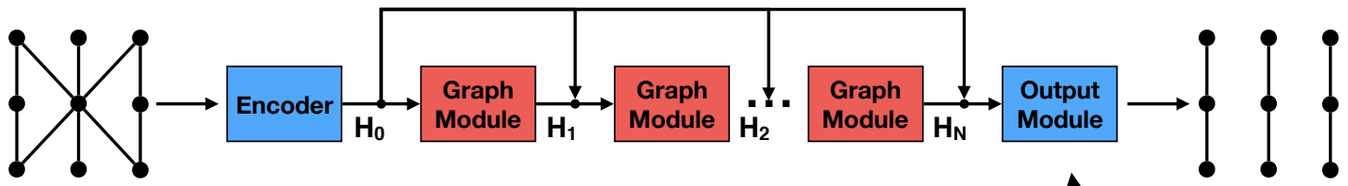


- Number of operations
  - For realistic problems, V ~ O($10^{3\text{-}4}$), E ~ O($10^{4\text{-}5}$) or higher
  - Typically, transformations are multilayer perceptrons
    → Number of multiplications explodes very quickly
  - Example: Exa.TrkX GNN segment classifier (not optimized for resource)
    O(10^6) / O(10^7) multiplications per vertex / edge
    ⇒ 2.6×10^{10} multiplications[1]
- Lots of algorithms use substantial memory (gigabytes)
  - FPGA on-chip RAM is O(10-100) Mb

[1] https://gist.github.com/jmduarte/c8783a4c9efa6cfd6c0638c03d3bd561

GNN in FPGA: Challenges

NeurIPS ML4PS 2019 83



- Number of operations
  - For realistic problems, V ~ O($10^{3\text{-}4}$), E ~ O($10^{4\text{-}5}$) or higher
  - Typically, transformations are multilayer perceptrons
    → Number of multiplications explodes very quickly
  - Example: Exa.TrkX GNN segment classifier (not optimized for resource)
    O(10^6) / O(10^7) multiplications per vertex / edge
    ⇒ 2.6×10^{10} multiplications[1]
- Lots of algorithms use substantial memory (gigabytes)
  - FPGA on-chip RAM is O(10-100) Mb
- Sparse adjacency matrix → irregular memory access
  - Edges make reference to entries in the array of vertices
  - But FPGA logic performs better when array access pattern is known at synthesis time

[1] https://gist.github.com/jmduarte/c8783a4c9efa6cfd6c0638c03d3bd561

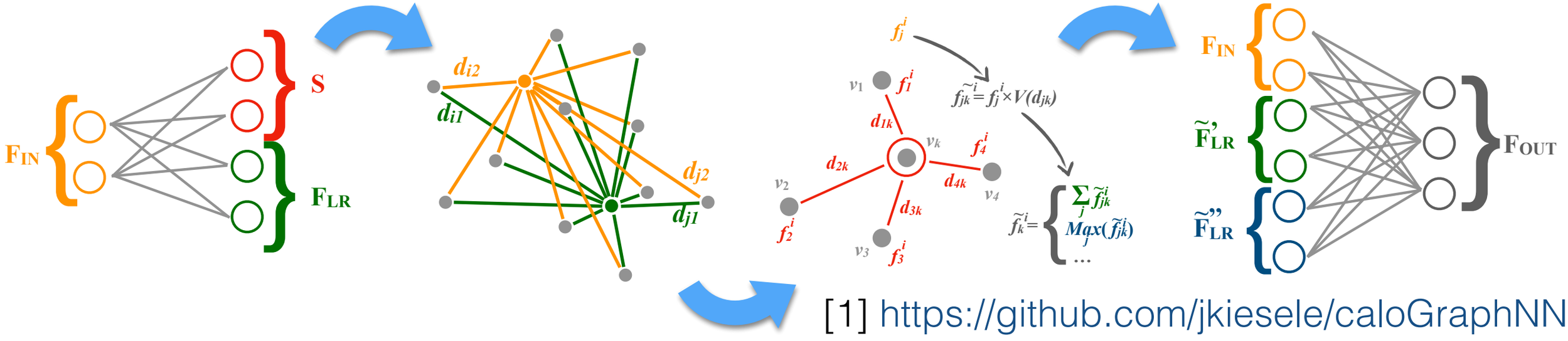
GarNet: light-weight GNN

1902.07987

General-purpose graph network suited for L1T usage

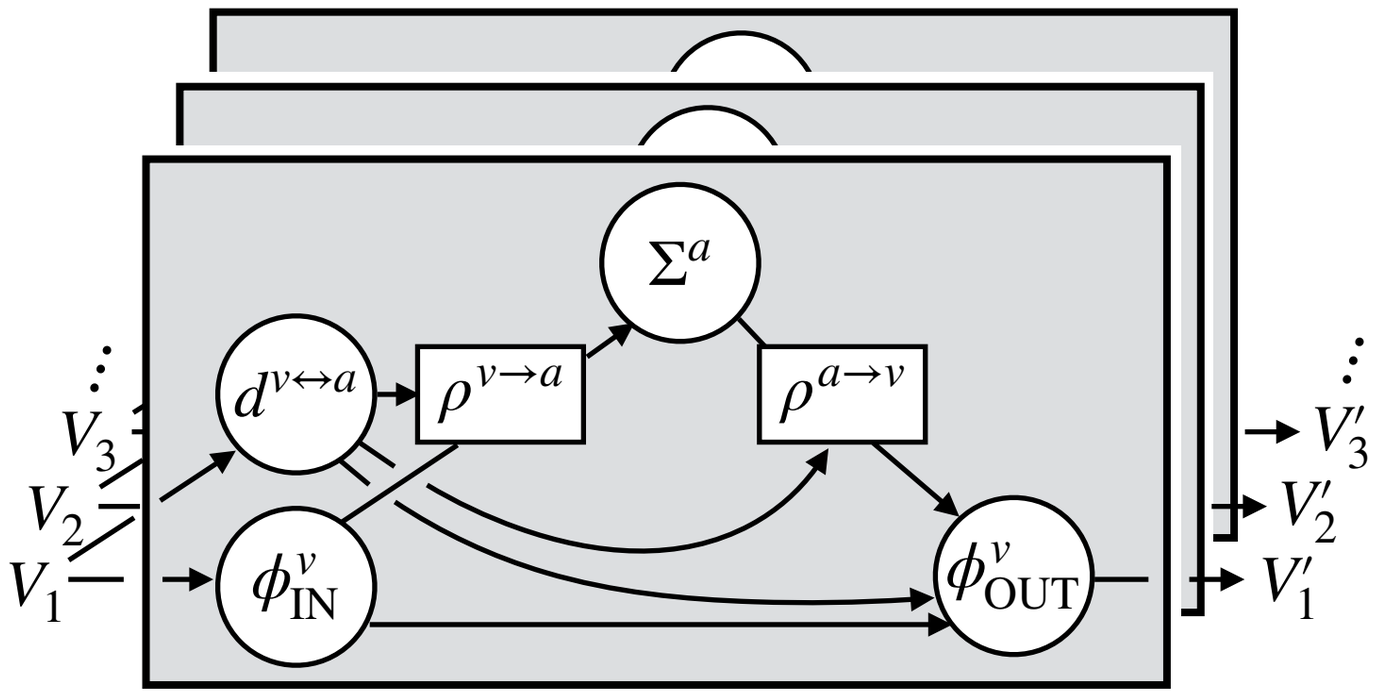
- No $\mathscr{E}$
  → Small memory footprint
  → No dynamic access into vertices array
- Vertex features weighted by a nonlinear function
  → Can learn nontrivial features with shallow transformation networks
  → Requires small number of operations

Original implementation[1] in TensorFlow and Keras
→ Not optimized for FPGA
→ Let's use GarNet to illustrate how to fit a GNN on FPGA



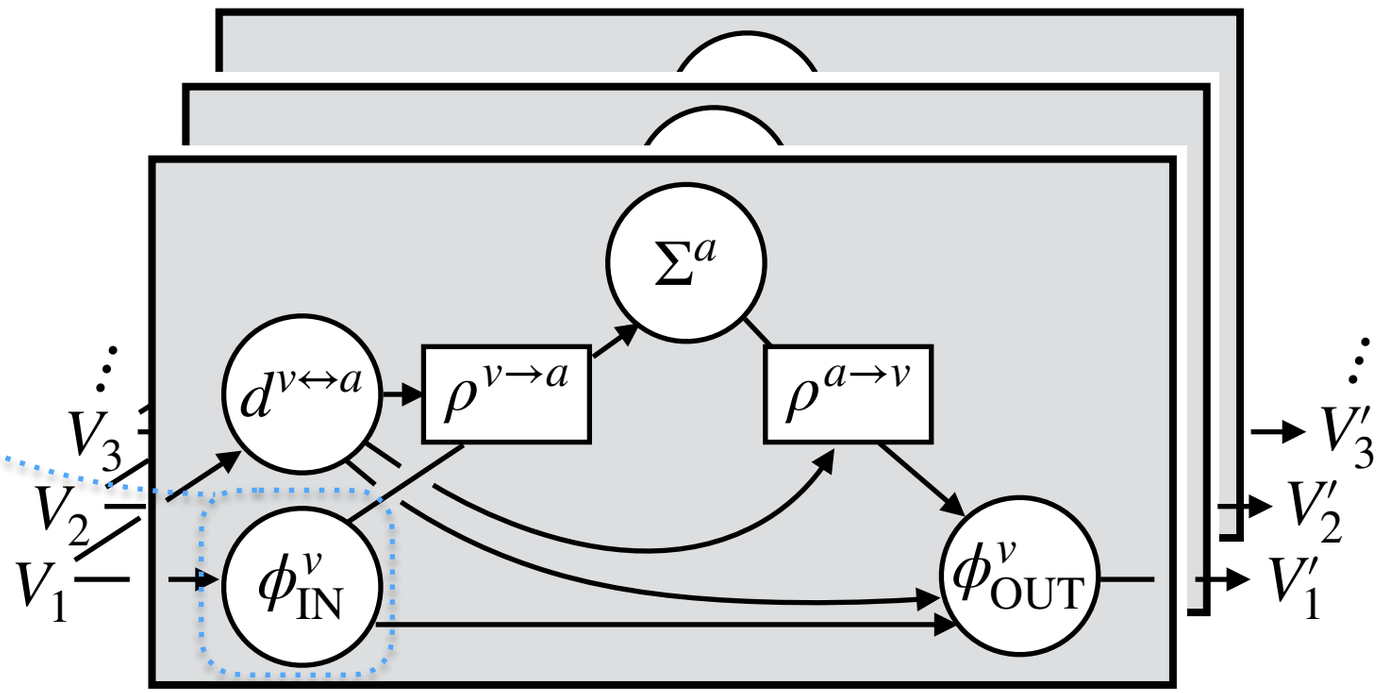
[1] https://github.com/jkiesele/caloGraphNN

GarNet layer in a nutshell



GarNet layer in a nutshell

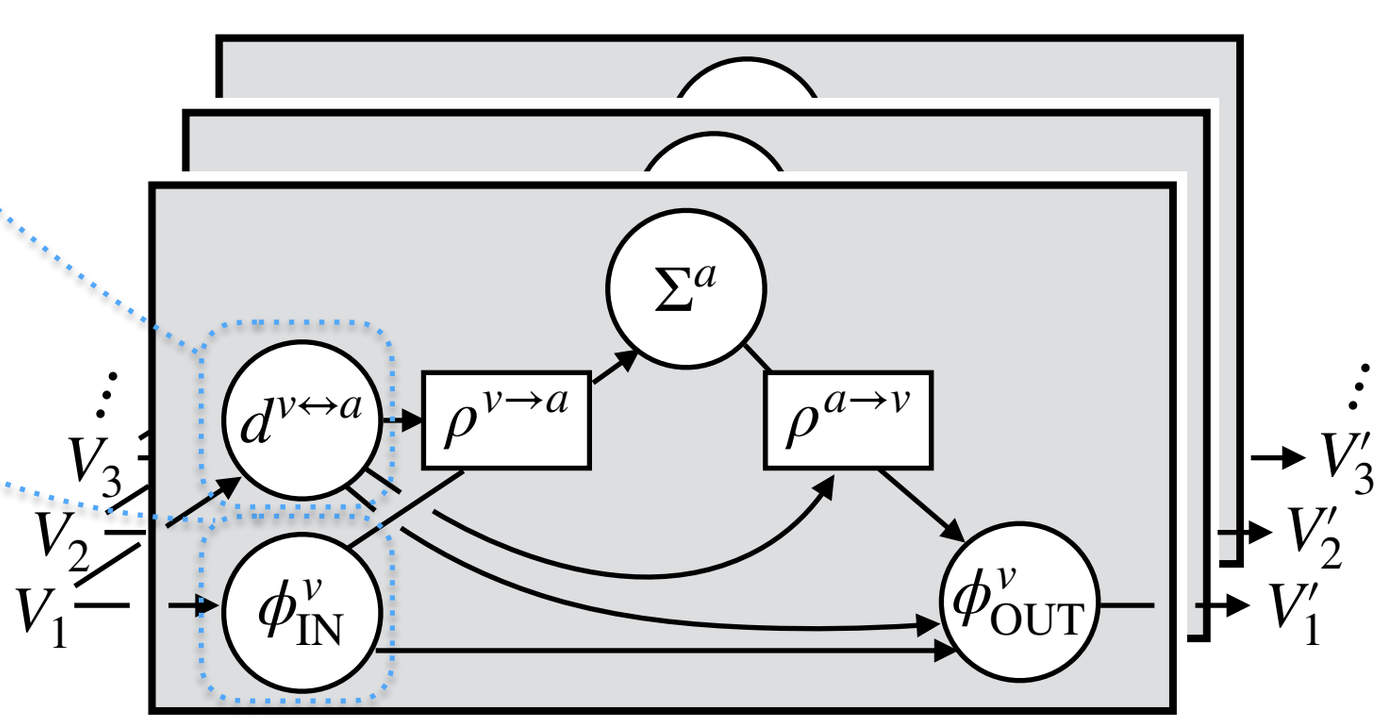
1. Encoder



GarNet layer in a nutshell

2. Distance of the vertex to virtual "aggregator nodes" a

1. Encoder

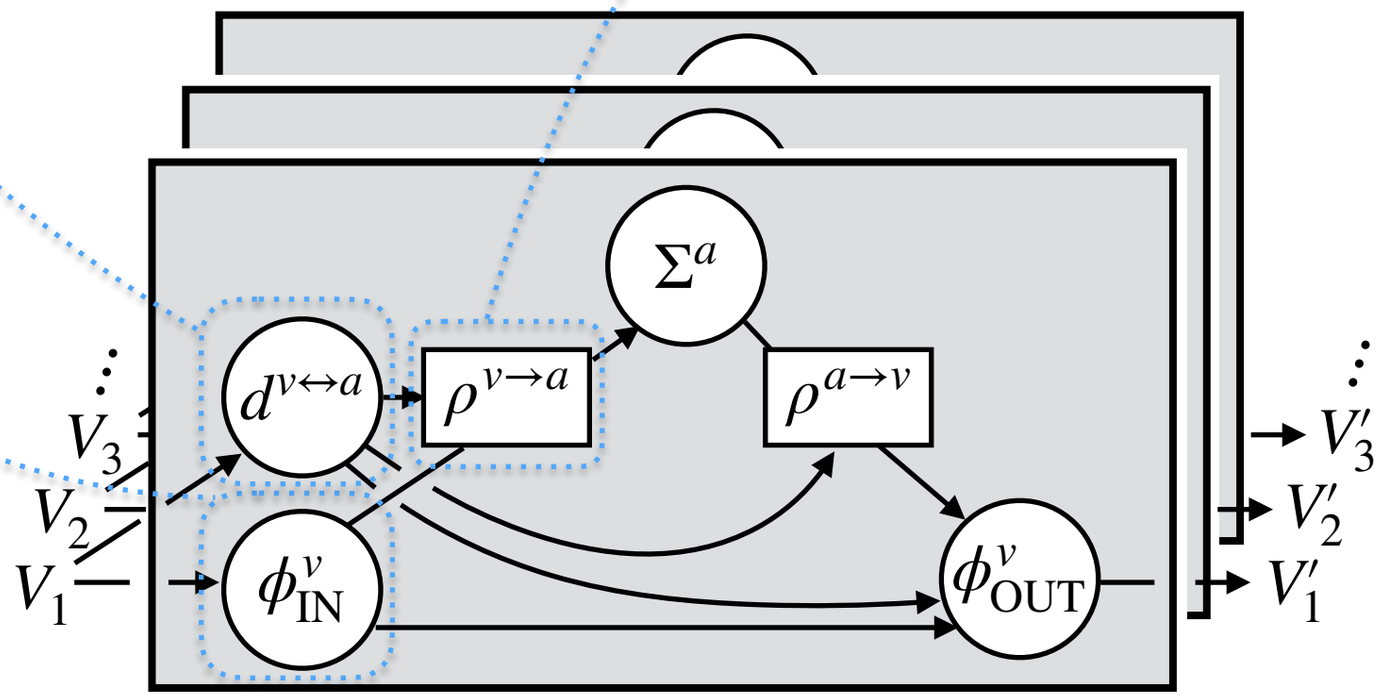


GarNet layer in a nutshell

3. Encoded features are weighted by $\mathbf{Gaus}(d^a)$

2. Distance of the vertex to virtual "aggregator nodes" a

1. Encoder



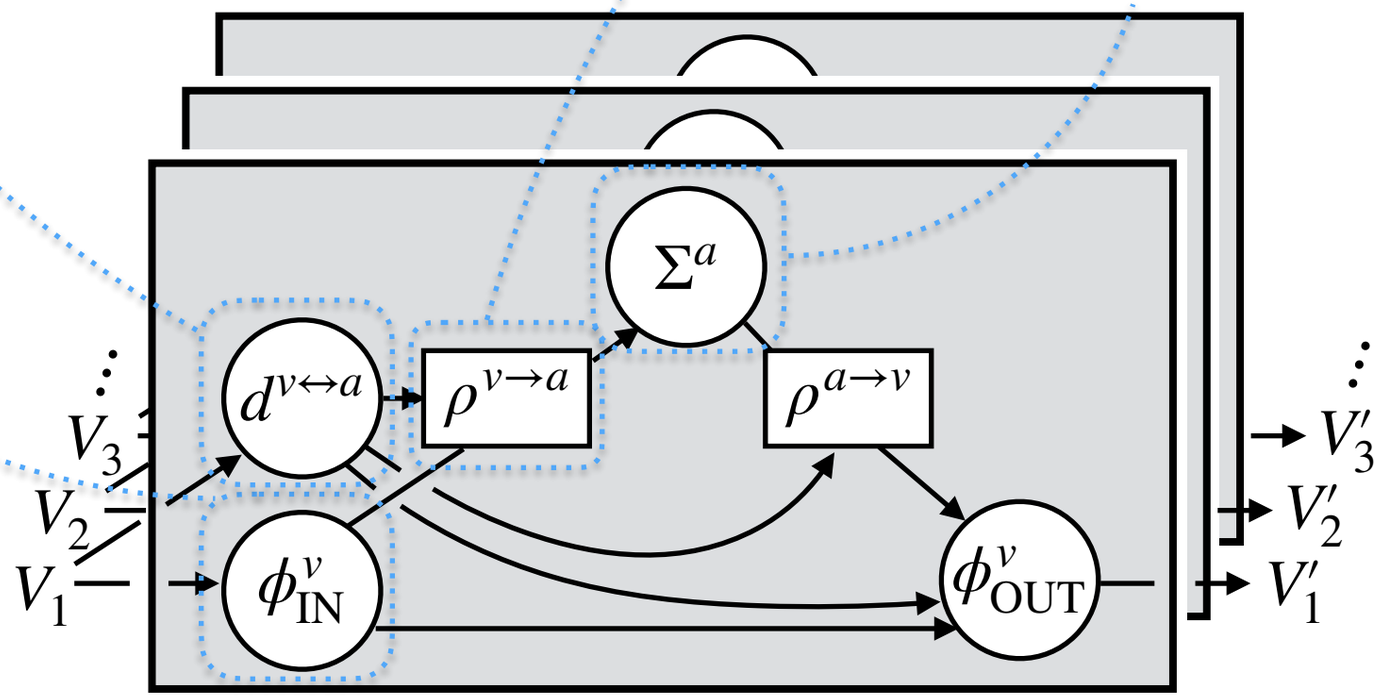
GarNet layer in a nutshell

3. Encoded features are weighted by $\mathbf{Gaus}(d^a)$

2. Distance of the vertex to virtual "aggregator nodes" a

4. And aggregated (mean & max) over all vertices

1. Encoder



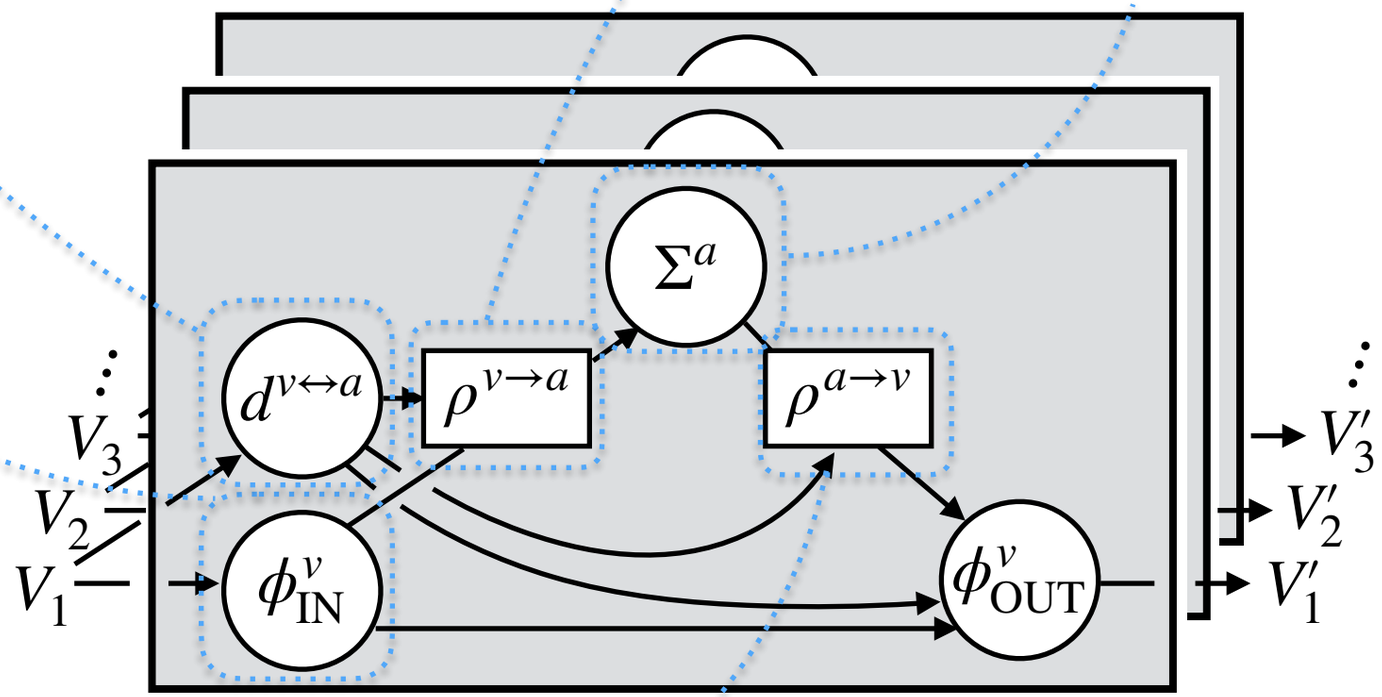
GarNet layer in a nutshell

3. Encoded features are weighted by $\mathbf{Gaus}(d^a)$

2. Distance of the vertex to virtual "aggregator nodes" a

4. And aggregated (mean & max) over all vertices

1. Encoder



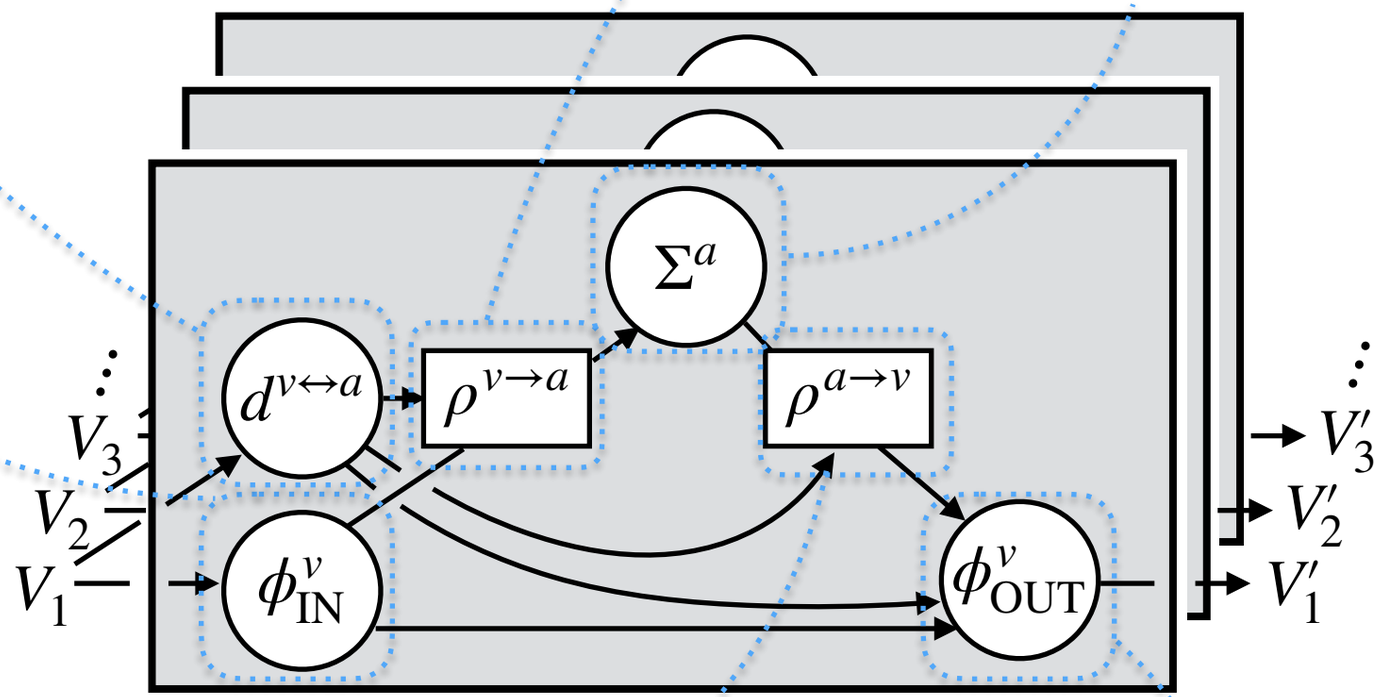
5. Aggregated features are given the same weights and sent back to vertices

GarNet layer in a nutshell

3. Encoded features are weighted by $\mathbf{Gaus}(d^a)$

2. Distance of the vertex to virtual "aggregator nodes" a

4. And aggregated (mean & max) over all vertices



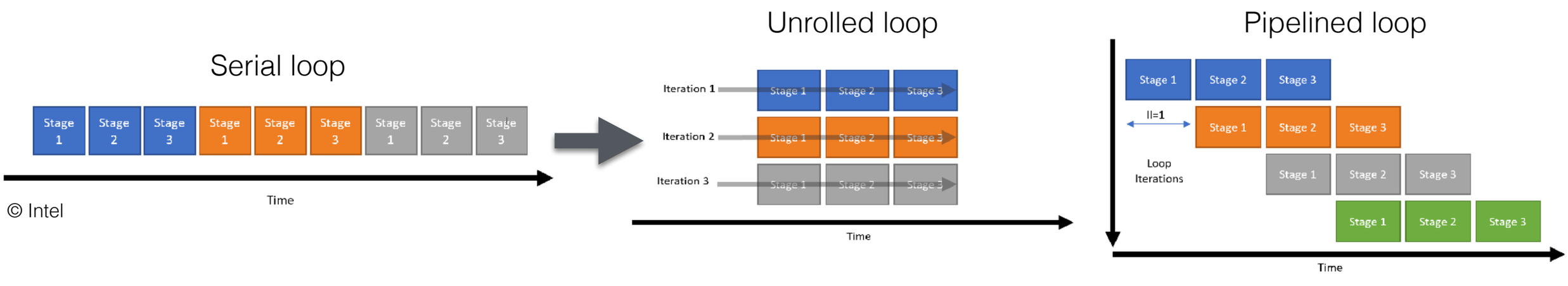
1. Encoder

5. Aggregated features are given the same weights and sent back to vertices

6. Decoder computes the output features for each vertex

Key concepts in FPGA logic design

- Resources
  - Lookup tables (LUTs)
    Perform logic operations
  - Digital signal processing units (DSPs)
    Perform $a \times (b + c) + d$ in one clock
  - Flip-flops (FFs)
    Registers. Basic storage unit.
  - Memory
    Block RAM (BRAM), High bandwidth memory (HBM), etc.
- Parallelization
  - Loops can be unrolled
    - Each unrolled loop body executes in an independent circuit
      → Saves execution time (latency) but costs resources
- Pipelining of functions and loops
  - Start processing the next input before the current one is processed
  - Time before next input is accepted = initiation interval (II)

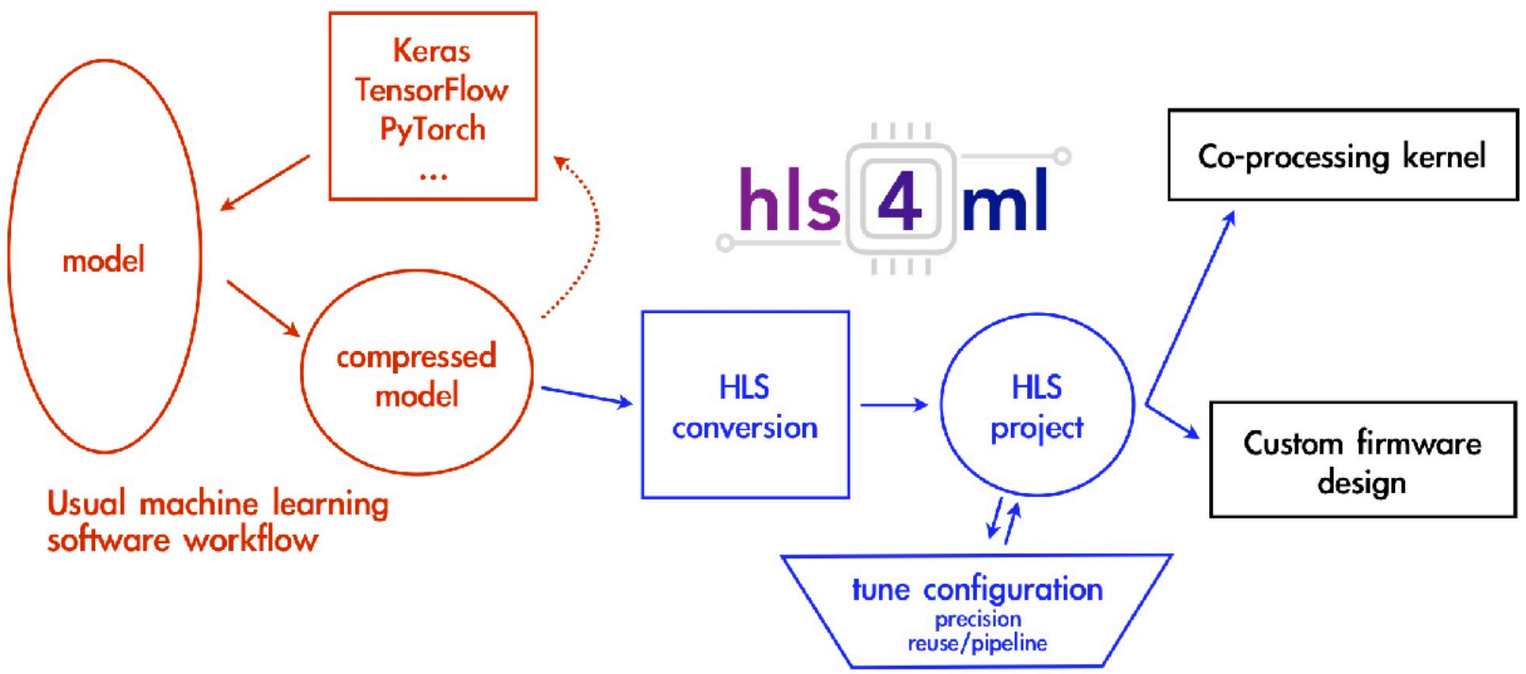


FPGA key metrics

- Latency
  - Depends on the number of serial operations
  - High-precision arithmetic also requires more clock cycles
- Initiation interval (II)
  - Shorter II → higher throughput
  - Mostly depends on the logic implementation
  - Simple data flow leads to shorter II
- Resource usage
  - Often in trade-off with latency
  - DSP tends to be the bottleneck
    → high impact if multiplications are reduced

HLS4ML

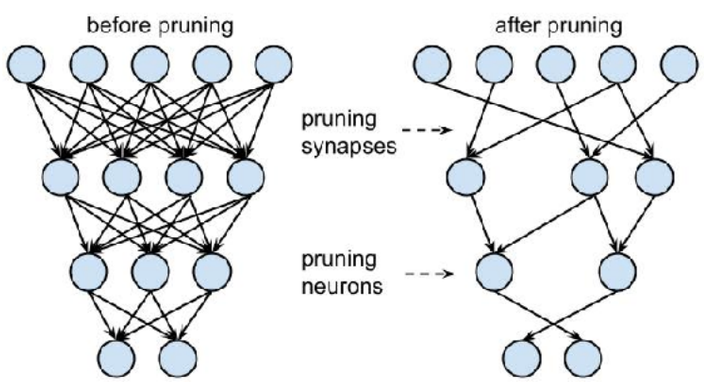
- A package for machine learning inference in FPGAs
- ML models in Keras etc. ⇒ High-Level Synthesis project
  - Project converted to firmware with Xilinx Vivado HLS
- Core asset: Original library of C++ (HLS) templates
  - Implements ML algorithm layers (Dense, Conv2D, etc.)



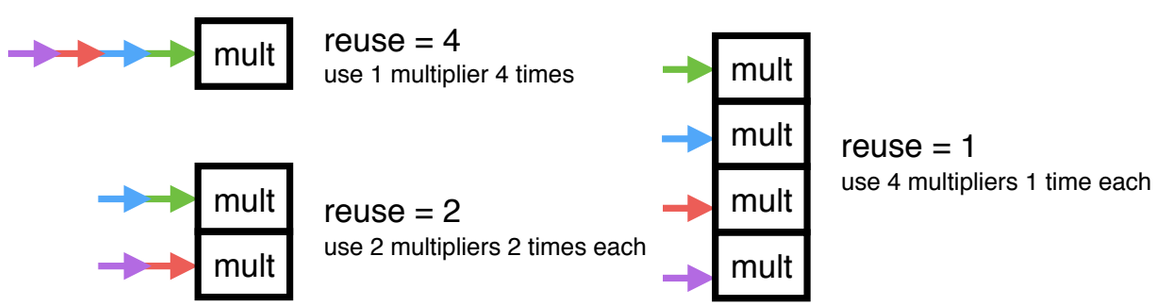
https://fastmachinelearning.org/hls4ml/

HLS4ML features

- Hyper-parameters (model config) and parameters (weights) are baked into the firmware at synthesis time
- Templates support pruned weights (compressed models)
- Weight quantization also supported
  - Represent weights by {-1, 1} (binary) or {-1, 0, 1} (ternary)
- Parallelization for each layer tuned by reuse factor
- Uses fixed-point numbers internally



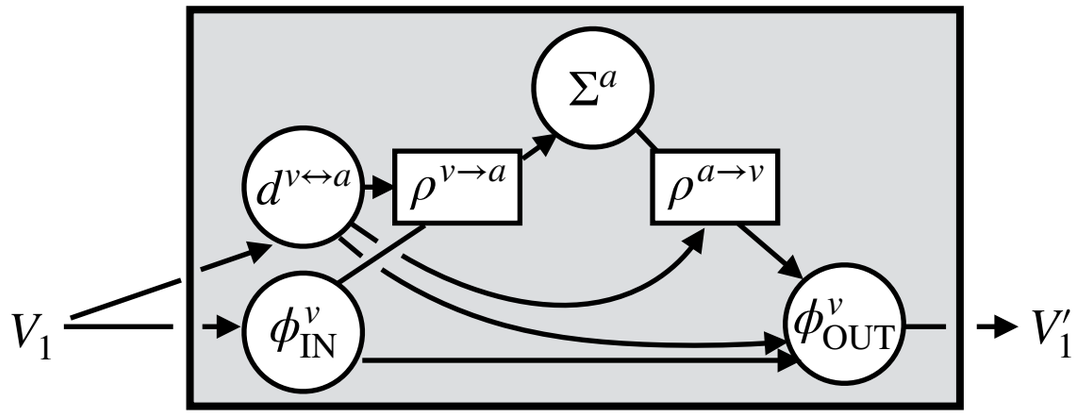
© O'Reilly



1804.06913

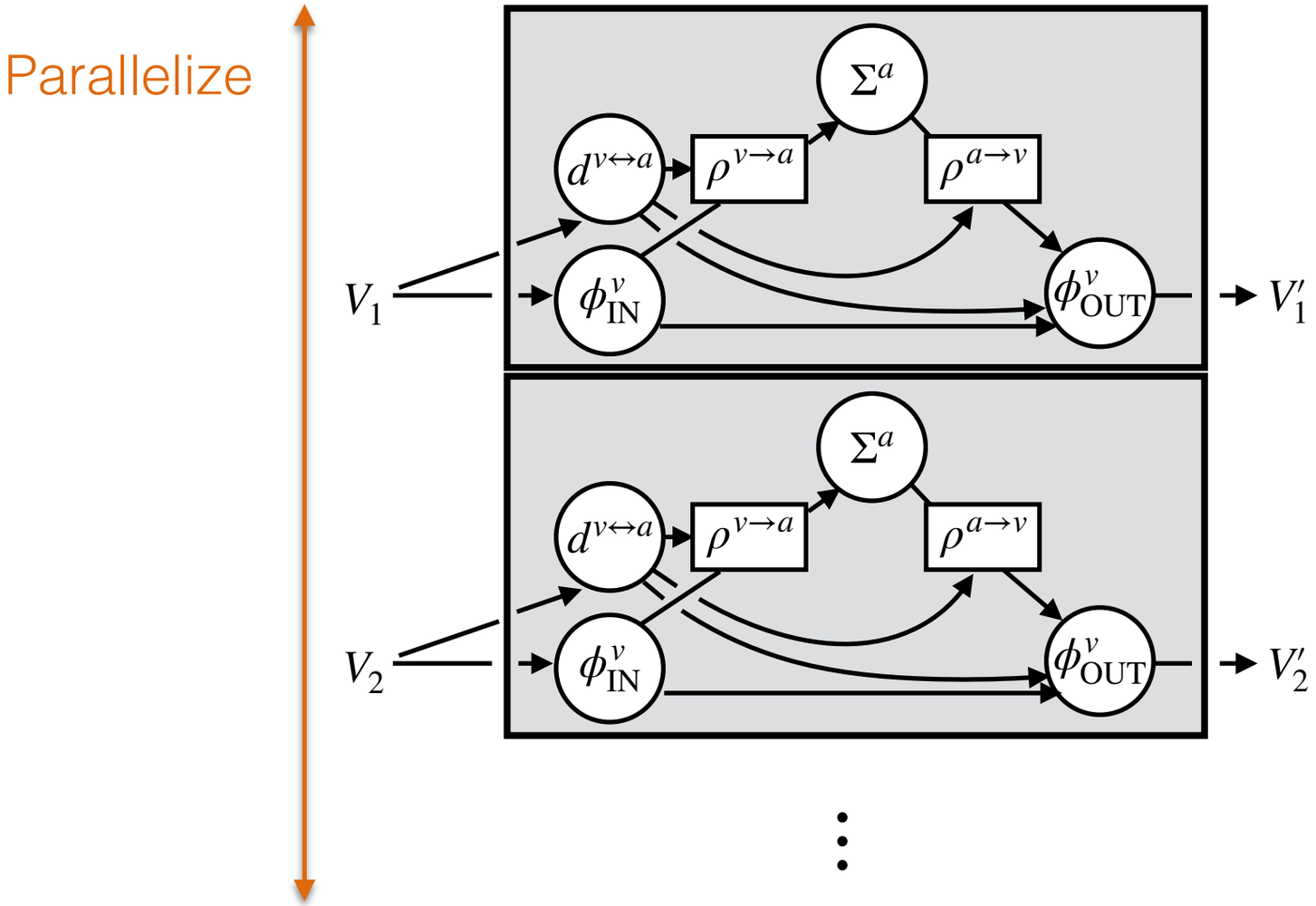
GarNet in HLS4ML: block modifications

HLS4ML will support GarNet in a modified form:



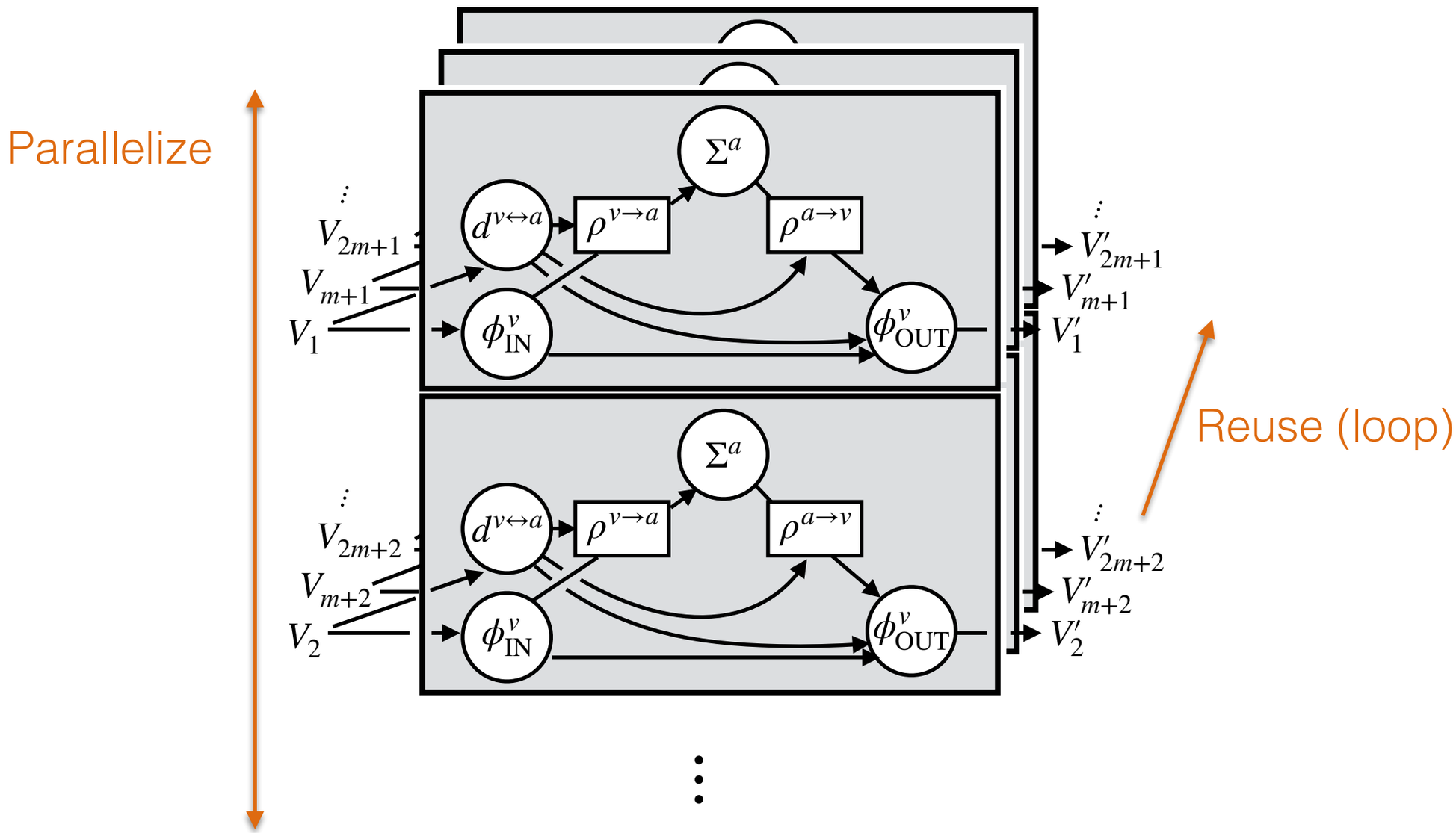
GarNet in HLS4ML: block modifications

HLS4ML will support GarNet in a modified form:



GarNet in HLS4ML: block modifications

HLS4ML will support GarNet in a modified form:

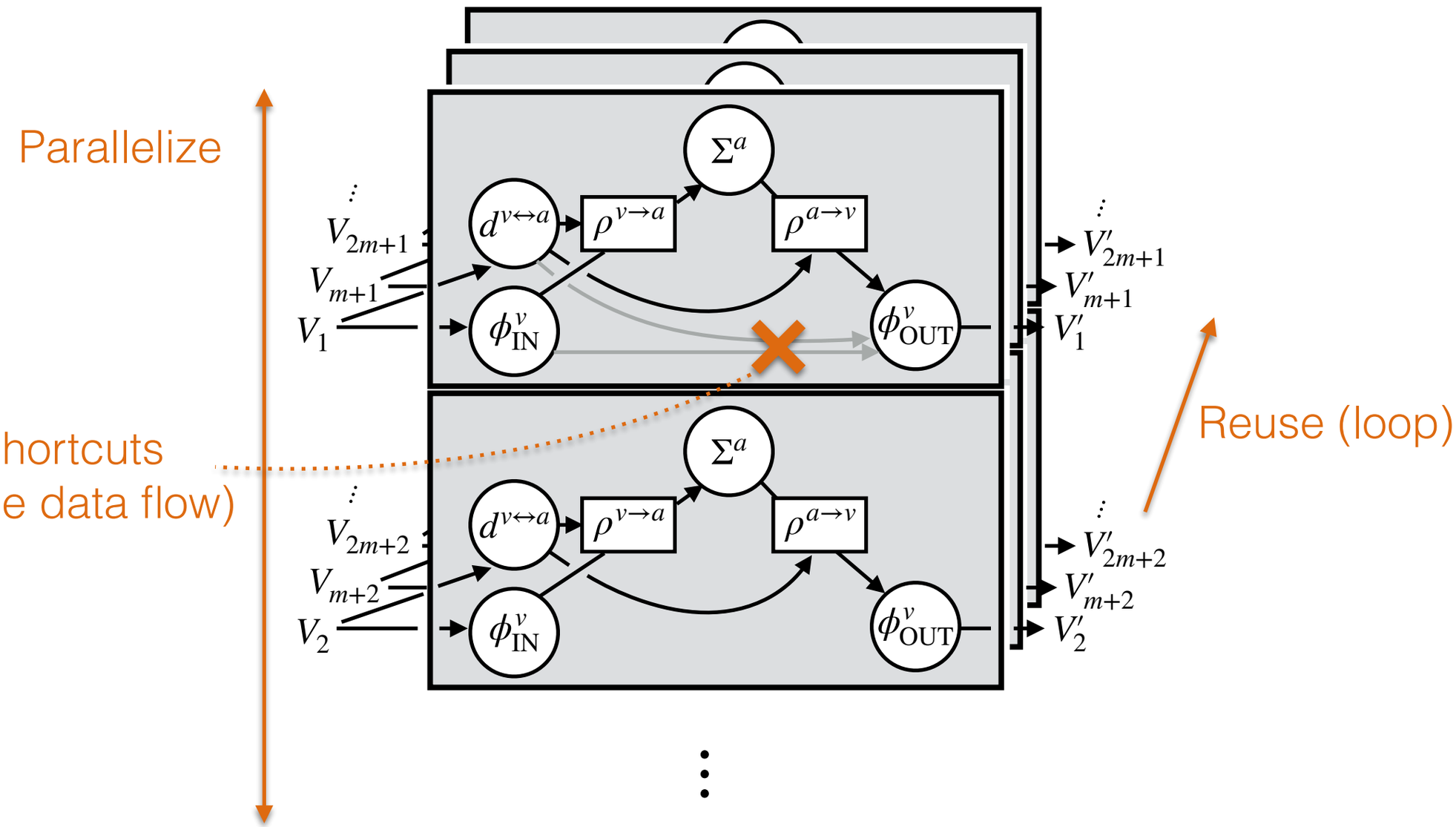


Reuse factor (length of loop) configurable

→ Users define the balance between latency and resource usage

GarNet in HLS4ML: block modifications

HLS4ML will support GarNet in a modified form:

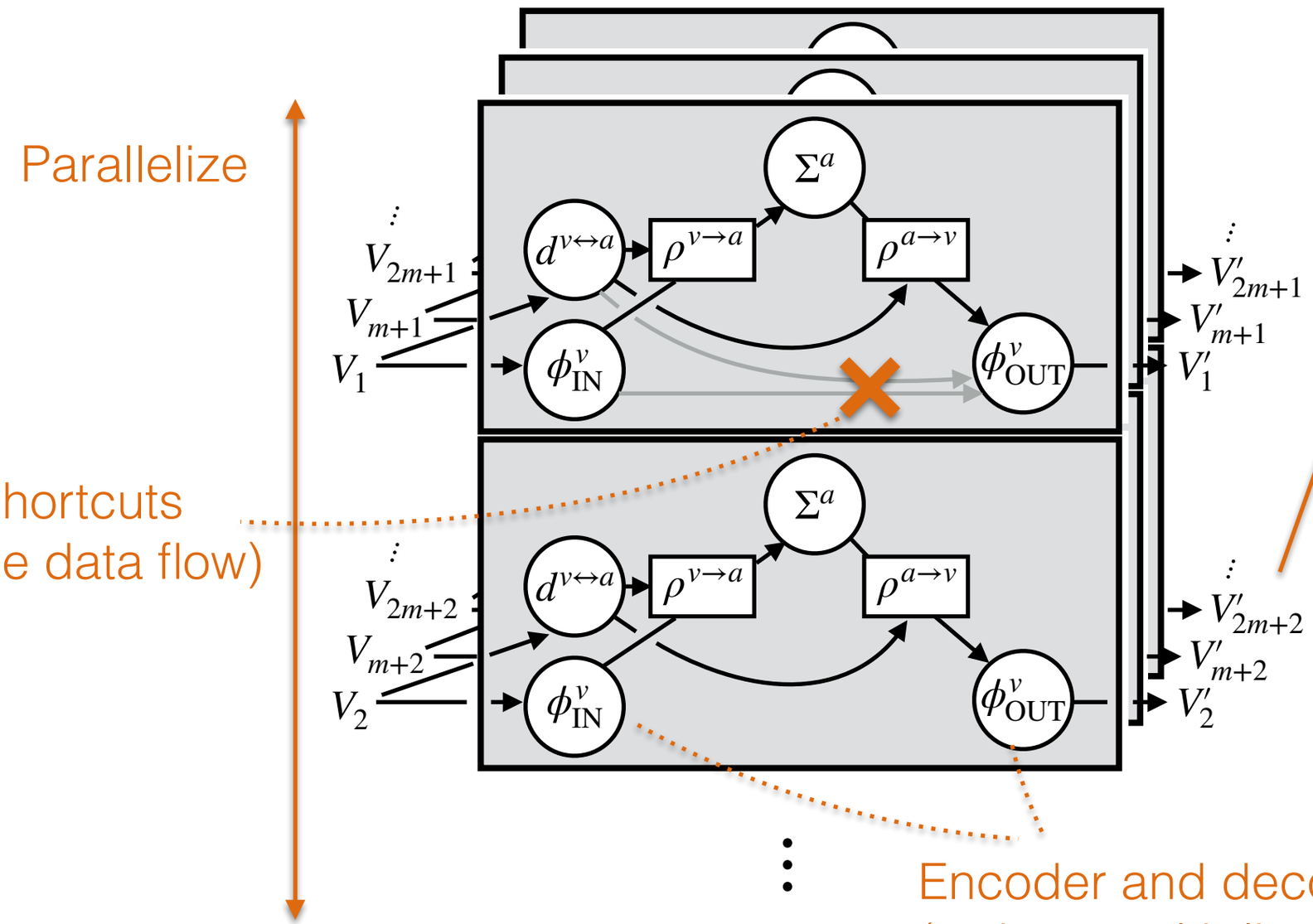


Reuse factor (length of loop) configurable

→ Users define the balance between latency and resource usage

GarNet in HLS4ML: block modifications

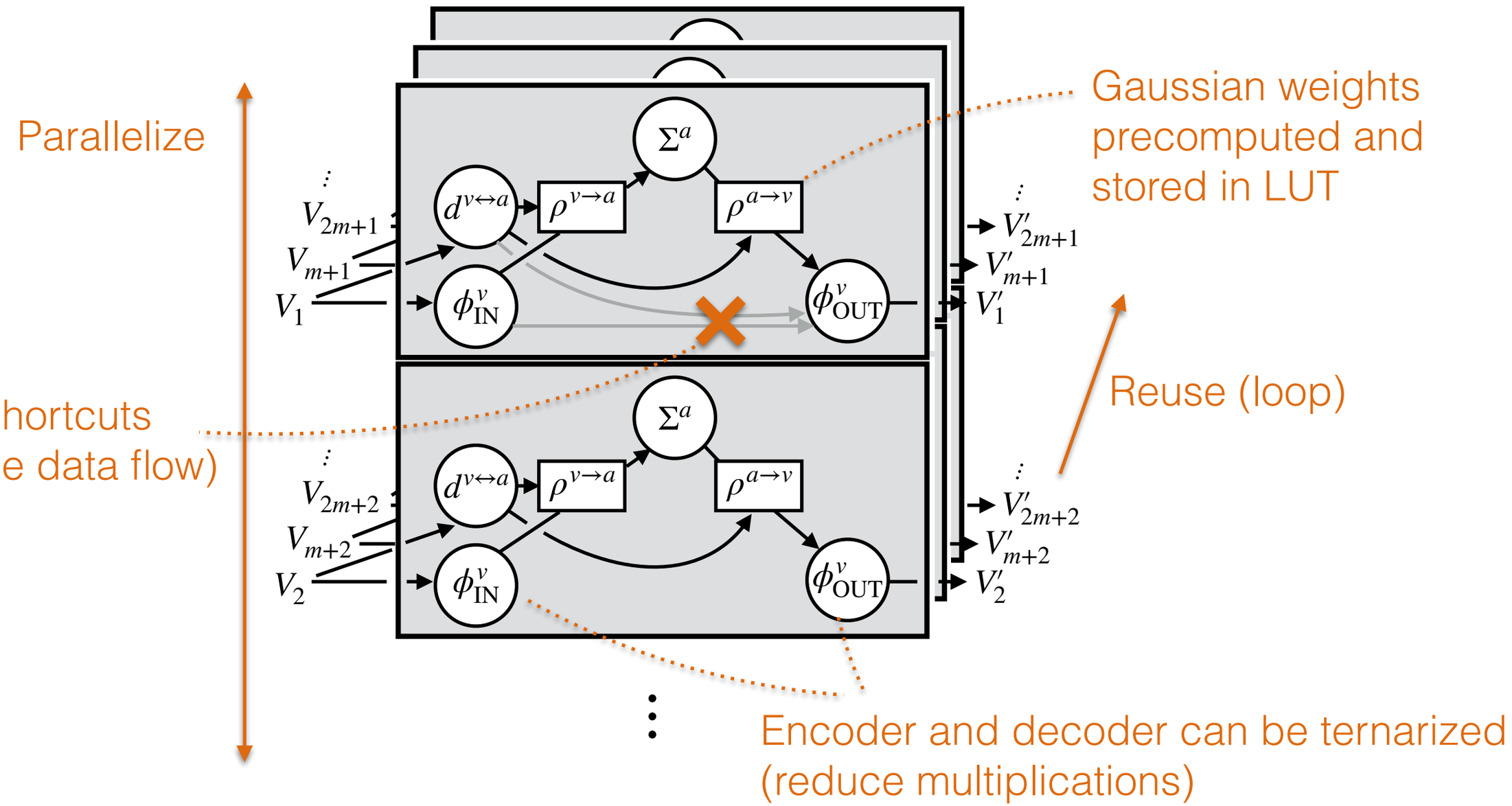
HLS4ML will support GarNet in a modified form:



Reuse factor (length of loop) configurable
→ Users define the balance between latency and resource usage

GarNet in HLS4ML: block modifications

HLS4ML will support GarNet in a modified form:



Reuse factor (length of loop) configurable

→ Users define the balance between latency and resource usage

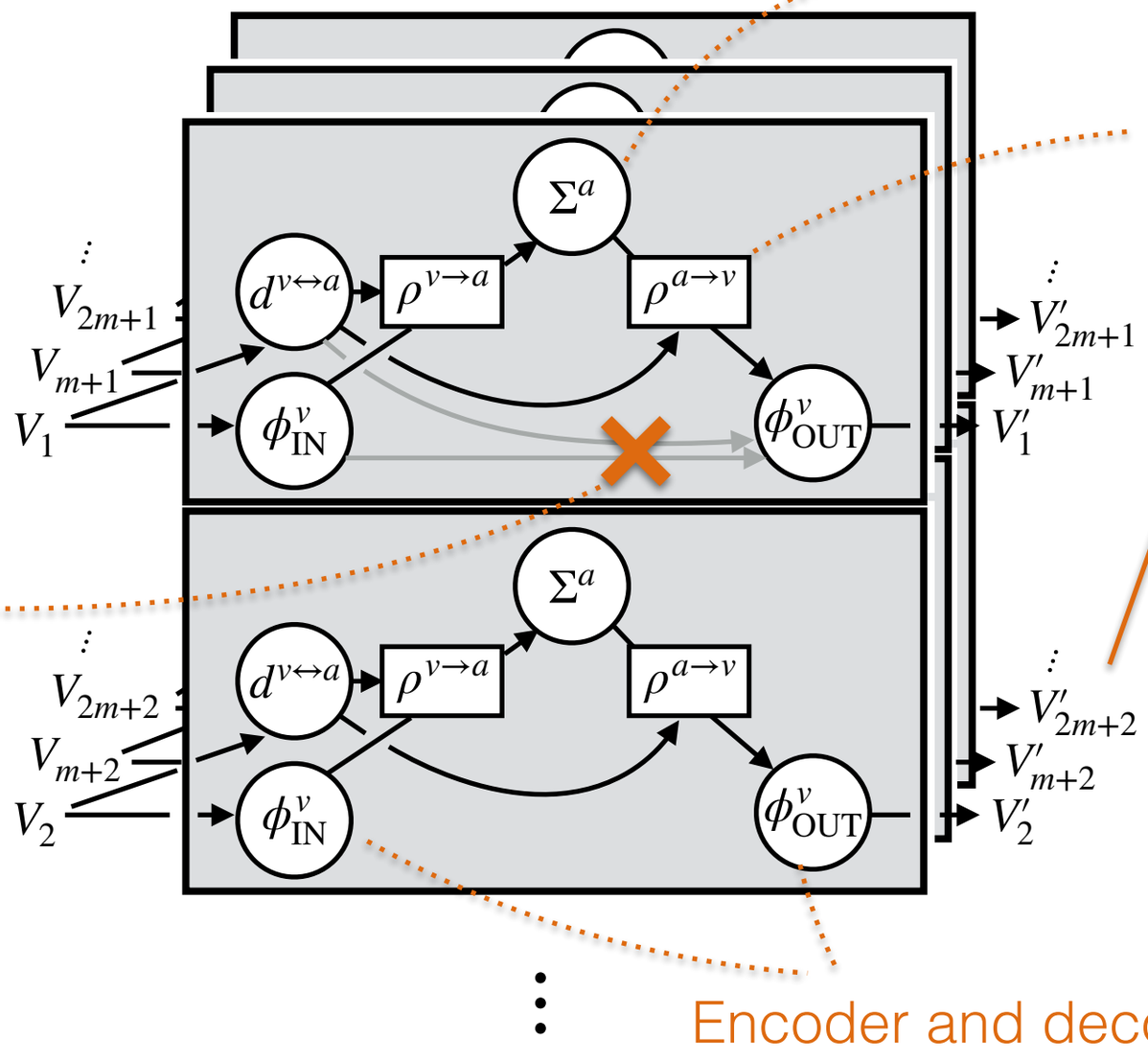
GarNet in HLS4ML: block modifications

HLS4ML will support GarNet in a modified form:

Only mean aggregation
(reduce operations)

Gaussian weights
precomputed and
stored in LUT

Parallelize



Eliminate shortcuts
(simplify the data flow)

Reuse (loop)

Encoder and decoder can be ternarized
(reduce multiplications)

Reuse factor (length of loop) configurable
→ Users define the balance between latency and resource usage

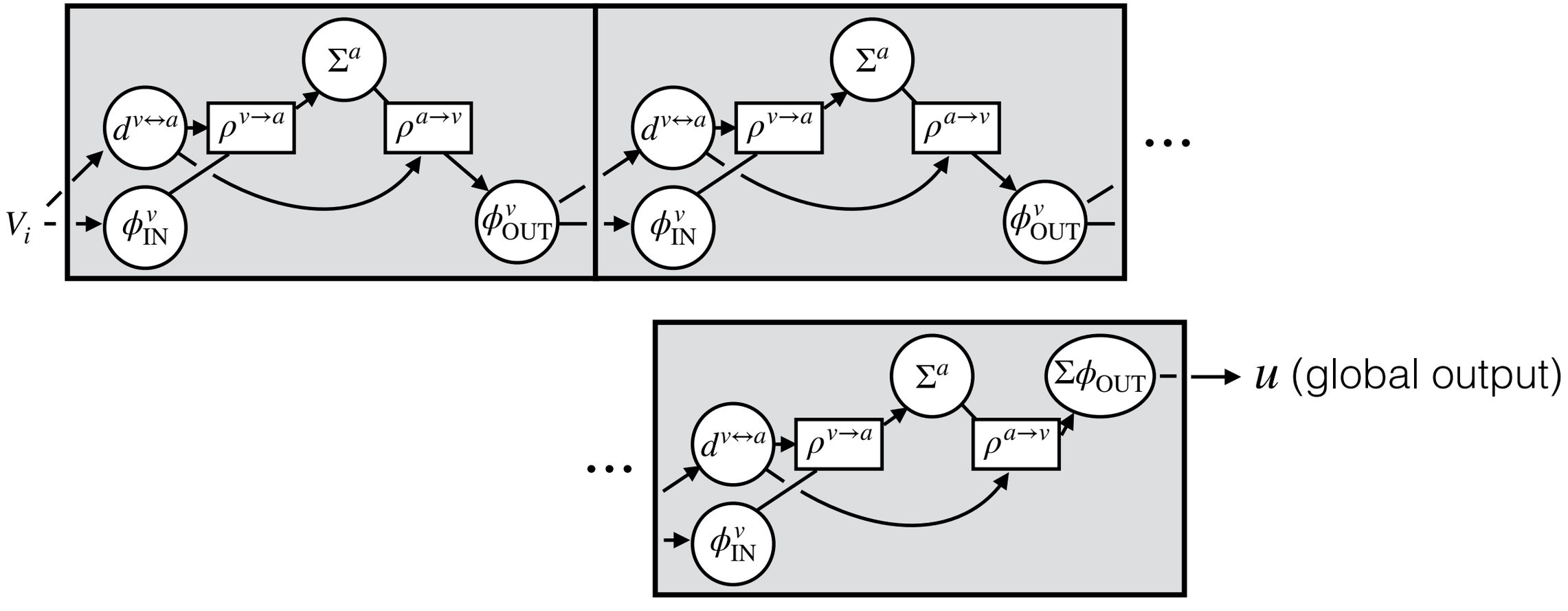
GarNet in HLS4ML: stacking optimization

Typical configuration:

- Stack multiple GarNet layers
- Output of final layer reduced over vertices

→ Build-in stacking and output aggregation

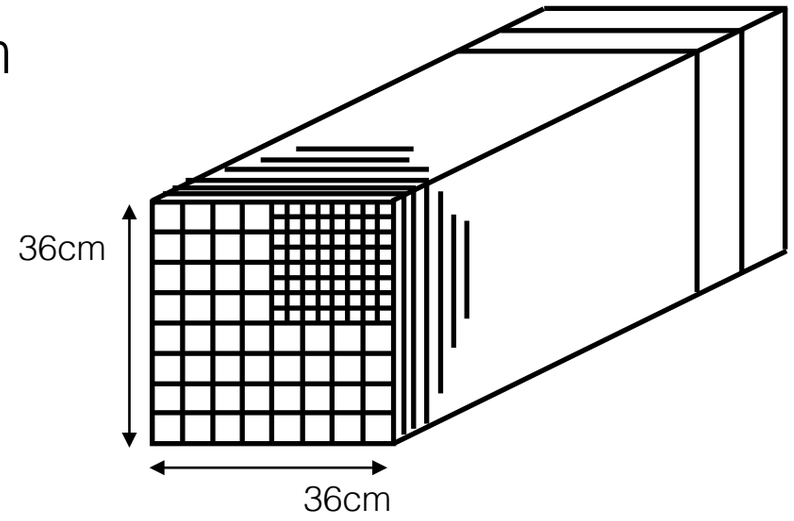
- Reduces BRAM usage (no intermediate array of vertices)
- Reduces latency (output of one layer + input of next in one iteration)



Case study:
PID and energy regression in a 3D calorimeter

Modified GarNet tested in a toy calorimeter simulation

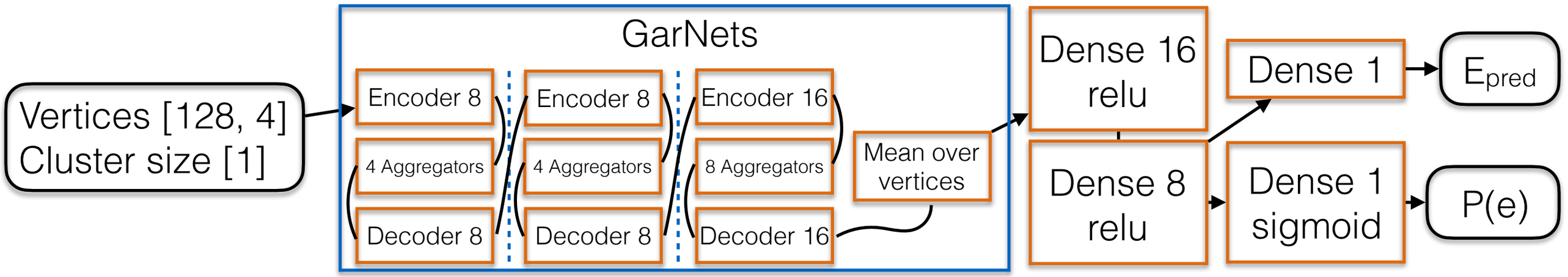
- Geometry: a cutout of a high-granularity 3D calorimeter
  - Varying cell size, minimum ~ 1cm × 1cm × 1cm
  - 50 longitudinal layers
  - 4375 cells in total



- Physics: primary $e^{\pm}/\pi^{\pm}$ + "pileup" $\pi^{\pm}/\gamma$
  - Primary particle energy flat in [10, 100] GeV
  - Pileup energy and flux equivalent to HL-LHC PU200 scenario at $|\eta| = 2$
- Sample: "cluster" around the highest-energy hit per event
  - Cluster = all hits in a cylinder (r = 6.4cm)
  - Each hit = graph vertex has 4 features [x, y, z, E]
- Dataset: 500k total events (250k each for e and π)
- Task: primary particle classification + energy prediction

Model and training

- Model



- Loss function =

$$0.99\left((E_{\text{pred}} - E_{\text{truth}})/E_{\text{truth}}\right)^2 + 0.01\,\text{BCE}\left(P(e)\right)$$

BCE: binary cross-entropy

- Training: 400k samples, ~500 epochs (early stopping)
- Two versions trained separately
  - "Continuous": encoder and decoder weights in fixed-point numbers with 10 binary fractional digits (`ap_fixed<N+10, N>`)
  - "Quantized": encoder and decoder ternarized

Classification performance

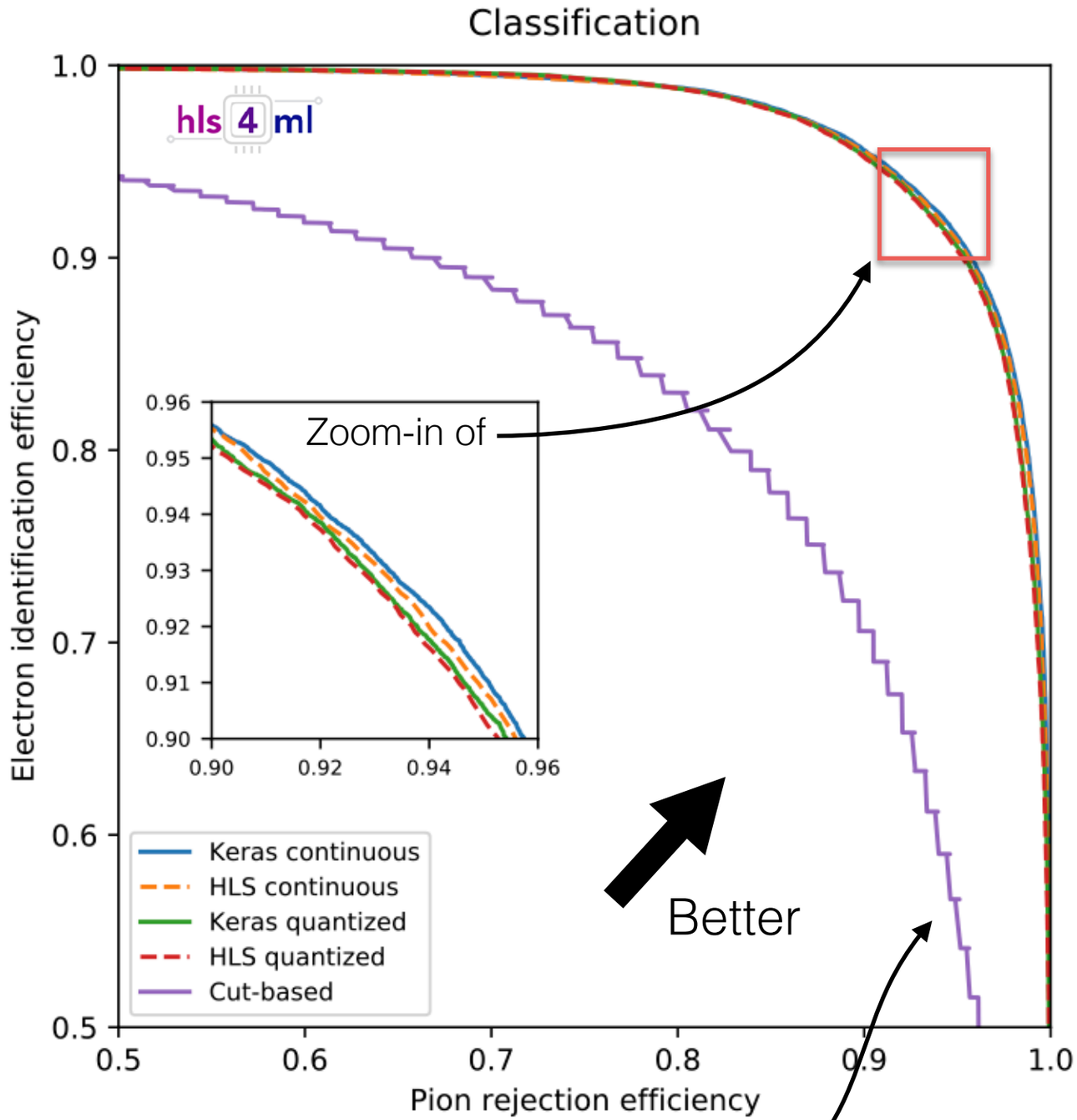
e± id – π± rejection ROC

Comparing

- Keras implementation (floating point)
- FPGA simulation of synthesized logic

Near-identical performance from all models

Small difference order as expected



Reference cut-based PID using

- cluster center-of-mass z
- cluster z spread

Regression performance

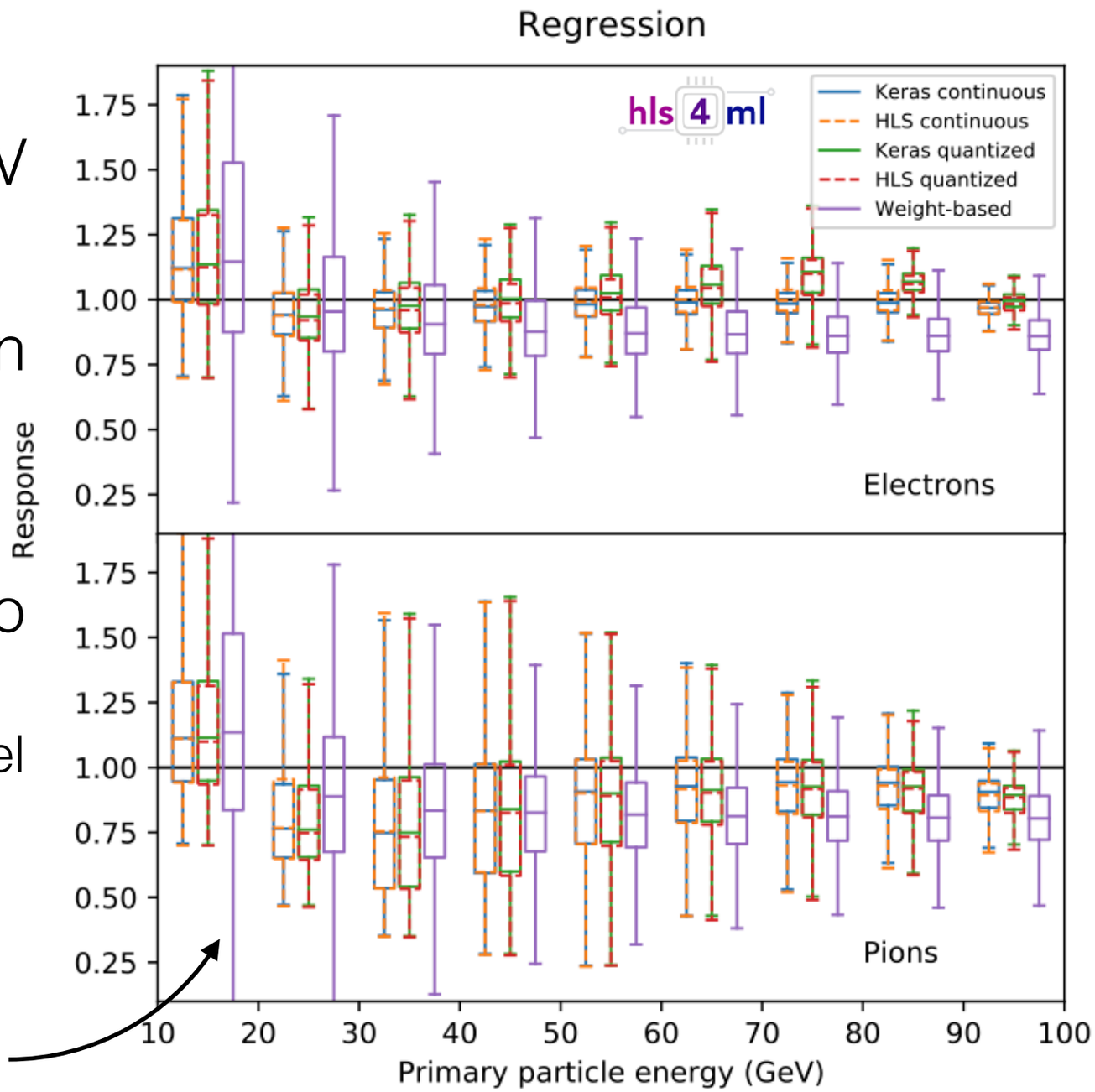
Response = $E_{predicted}$ / E_{truth}

Median and spread in 10 GeV E_{truth} bins

Negligible difference between Keras and HLS

Result not production grade, but shows that GarNet can do regression

(Lots of room for improving the model and loss function)



Reference weight-based regression:

$E = \sum W(z_{hit}) \times [E_{hit} + b(z_{hit})]$

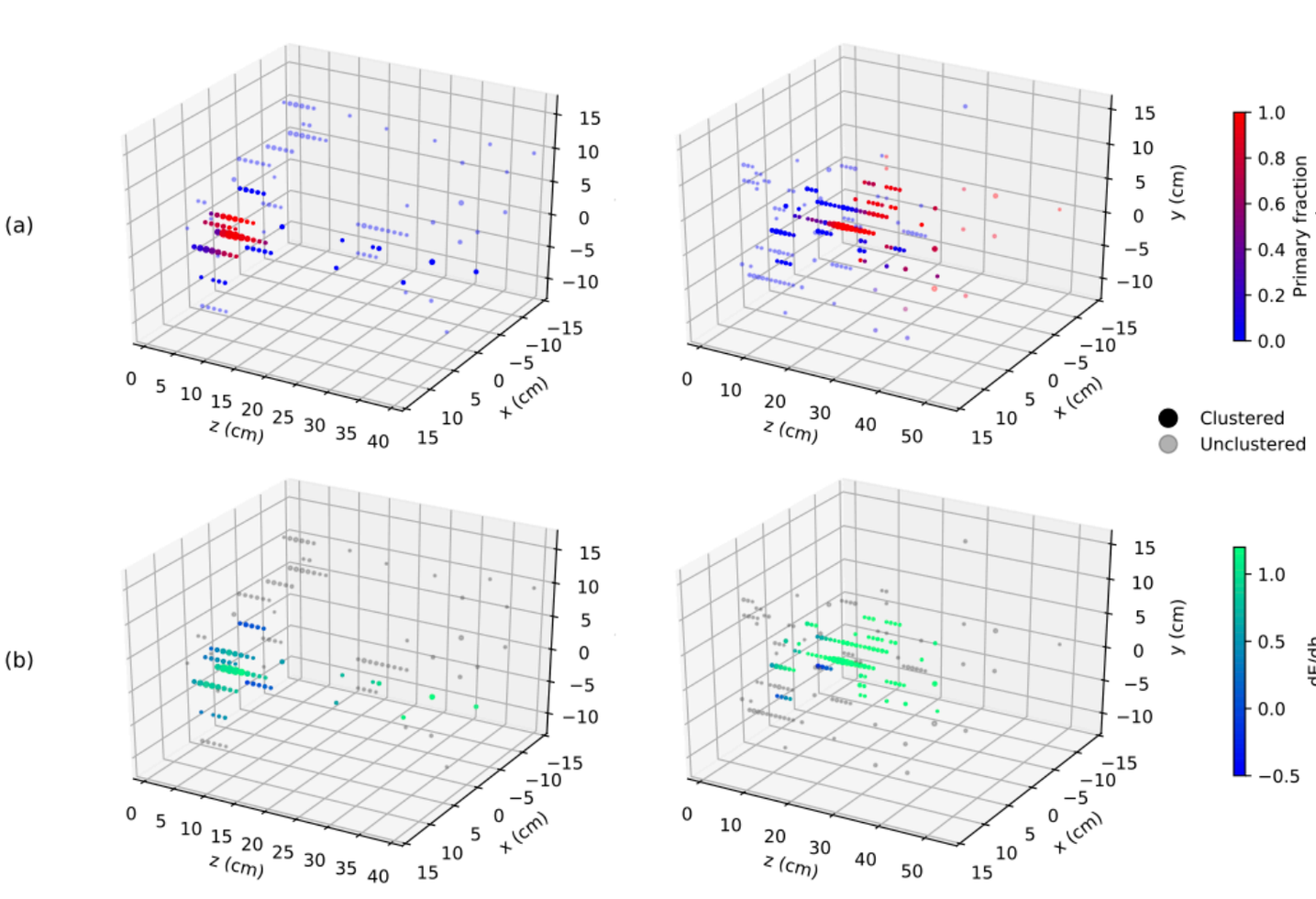
W(z) and b(z) parameters optimized by minimizing the regression loss for the training dataset

(Connecting the) dots?

Case study was about predicting graph-global properties
→ Can FPGA-size GNN infer edge and vertex properties (and do tracking)?

Note: GarNet is a largely vertex-local algorithm
→ Global inference must be happening through inherent local inference

Qualitative confirmation of the claim:
Extract per-vertex PU labeling from the PID+regression model



Fraction of hit energy from primary particle

How E_{pred} changes under hit energy perturbation

→ Network assigns higher importance to hits from primary particle

Synthesis results

HLS project synthesized for Xilinx Kintex Ultrascale 115

- Clock frequency 200 MHz (5 ns / cycle)
- Vivado HLS version 2019.2

GarNet fits on 1 FPGA with sub-μs latency

Need shorter initiation interval to be used in real L1T

	Continuous	Quantized
Latency	155 clk (0.83 μs)	148 clk (0.80 μs)
Initiation interval	55 clk (0.28 μs)	50 clk (0.25 μs)
LUT	57k (8.6%)	70k (11%)
FF	39k (3.0%)	41k (3.1%)
DSP	3.1k (57%)	1.6k (28%)
BRAM	1.8 Mb (2.3%)	1.9 Mb (2.4%)

Percentage: fraction of resource available on Ultrascale 115

Conclusion

Conclusion

- GNNs show promising performance in HEP tasks
  - Implementations for firmware triggers are desired

Conclusion

- GNNs show promising performance in HEP tasks
  - Implementations for firmware triggers are desired
- Identified key challenges for GNNs in FPGAs
  - Number of operations
  - Memory capacity and access

Conclusion

- GNNs show promising performance in HEP tasks
  - Implementations for firmware triggers are desired
- Identified key challenges for GNNs in FPGAs
  - Number of operations
  - Memory capacity and access
- Presented GarNet, which addresses these issues
  - No vertex-to-vertex edges → low RAM usage, regular access
  - Nonlinearity in edge weights → encoder / decoder can be shallow
  - Will be integrated into HLS4ML soon
    → Makes a general-purpose GNN layer on FPGA available to public

Conclusion

- GNNs show promising performance in HEP tasks
  - Implementations for firmware triggers are desired
- Identified key challenges for GNNs in FPGAs
  - Number of operations
  - Memory capacity and access
- Presented GarNet, which addresses these issues
  - No vertex-to-vertex edges → low RAM usage, regular access
  - Nonlinearity in edge weights → encoder / decoder can be shallow
  - Will be integrated into HLS4ML soon
    → Makes a general-purpose GNN layer on FPGA available to public
- Discussed algorithm simplifications / optimizations to fit a GNN in FPGA

Conclusion

- GNNs show promising performance in HEP tasks
  - Implementations for firmware triggers are desired
- Identified key challenges for GNNs in FPGAs
  - Number of operations
  - Memory capacity and access
- Presented GarNet, which addresses these issues
  - No vertex-to-vertex edges → low RAM usage, regular access
  - Nonlinearity in edge weights → encoder / decoder can be shallow
  - Will be integrated into HLS4ML soon
    → Makes a general-purpose GNN layer on FPGA available to public
- Discussed algorithm simplifications / optimizations to fit a GNN in FPGA
- Demonstrated a sub-μs latency GNN circuit for a L1T-like physics task

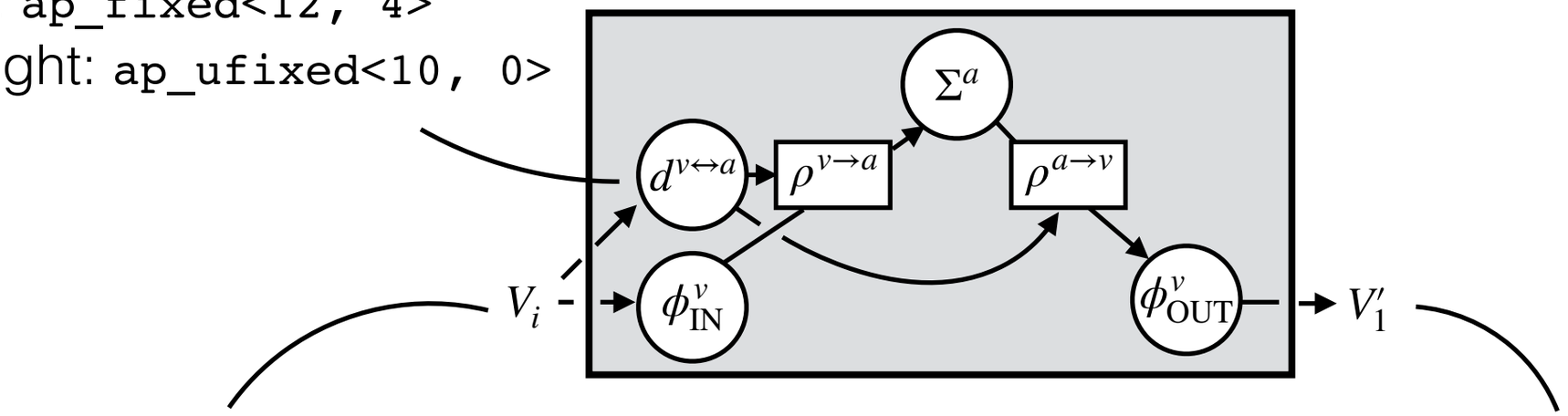
Backup

How to put a GNN on an FPGA

- Reuse: loop over $\mathscr{V}$ and $\mathscr{E}$
  - Unfeasible to parallelize thousands of edge and node blocks
  - But parallelize as much as possible → partial unrolling
- Reduce: simplify the logic and data flow
  - FPGA logic performs better when data flow is simple
  - Avoid shortcut connections (input reuse)
  - Use shallower, narrower MLPs
- Regularize array access
  - Use parallelizable data representation
  - Straightforward: input edges as V×V adjacency (edge) matrix
    → But this increases memory usage

HLS GarNet numerical precision

Distance: `ap_fixed<12, 4>`
Edge weight: `ap_ufixed<10, 0>`



`ap_fixed<14, 5>`

`ap_fixed<16, 6>`

Internally:

- Edge weight accumulation: `ap_ufixed<15, 5>`
- Feature accumulation: `ap_fixed<18, 8>`

* `ap_fixed<N, I>` = fixed precision number with N total and I integral bits