High Granularity Quantization

Chang Sun, Zhiqiang Que, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba
Wayne Luk, Maria Spiropulu

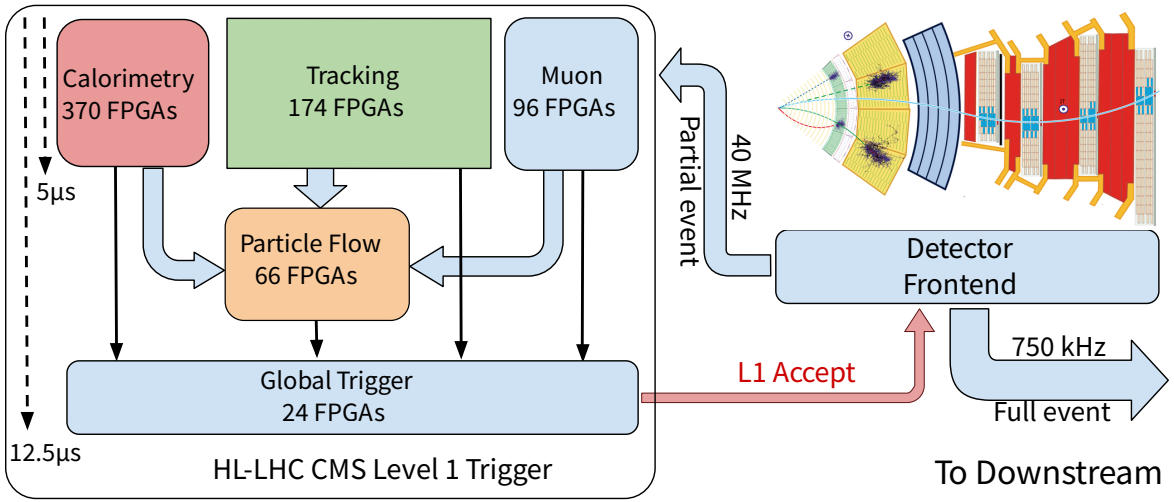
Caltech, PMA Division

22.02.2026

Some Background

The Level-1 Trigger System (L1T) at the CMS Experiment

- Deterministic latency
- <1 μs for individual algorithms
- ML algorithms are being explored and has been deployed
- Baseline methods: conventional quantization aware training, deployment via direct logic mapping (i.e., for loop unrolling) in `hls4ml` with Vitis HLS



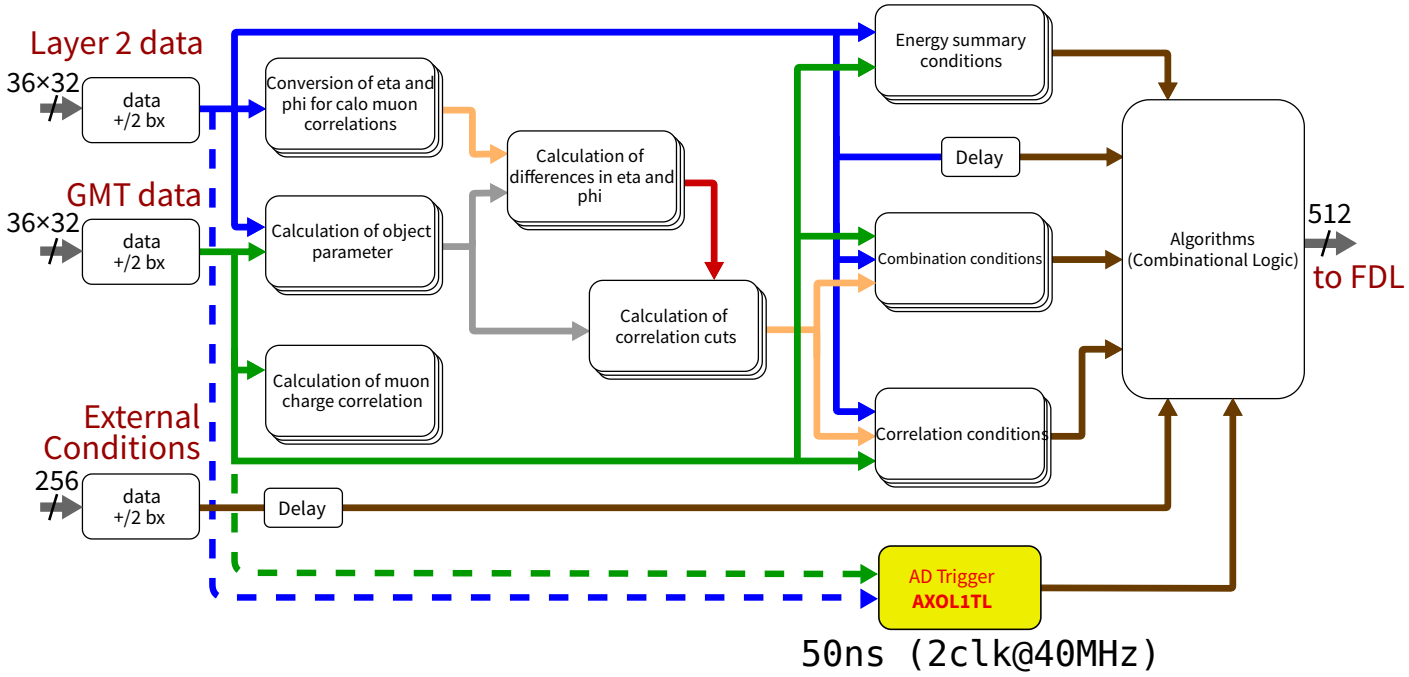
Schematic of the proposed CMS L1T for the future High-Luminosity LHC upgrade.

Example ML Trigger at CMS L1T

A VAE-based anomaly detection trigger at the CMS L1 Global Trigger

This trigger is deployed currently at the CMS L1T.

- 2 clocks @ 40 MHz
  - Sync with LHC clock
- part: `xc7vx690tffg1927-2`
- >30k LUTs utilization leads to timing violation in general
  - Other logic on same board



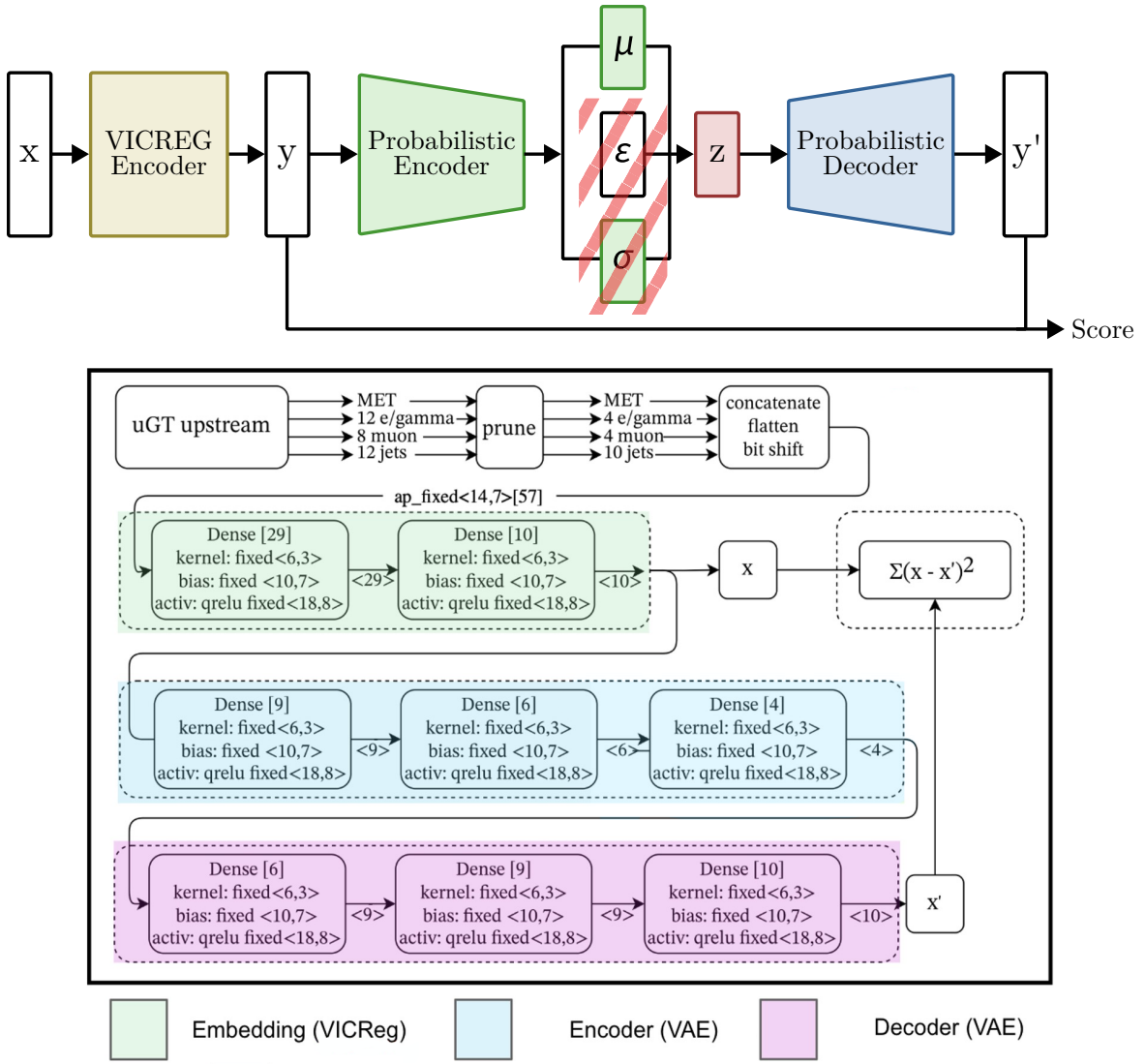
Some Background

Example Model in Production

8-layer MLP VAE with VICREG preprocessor

- Only two combinational blocks
  - All logics are unrolled
  - No register allowed
- DSPs are not useful here:
  - If all dense, this one needs ~2289 DSPs for all mul-accum ops

->We need specific ways to optimize for networks with similar requirements.



Key Motivation

For unrolled computation graph, uniform quantization is not optimal

- The operations no longer share the same PE

Challenge: The current methods support heterogeneous quantization only at tensor level.

- Not useful for unrolled models: the operations happen in parallel on different PEs.

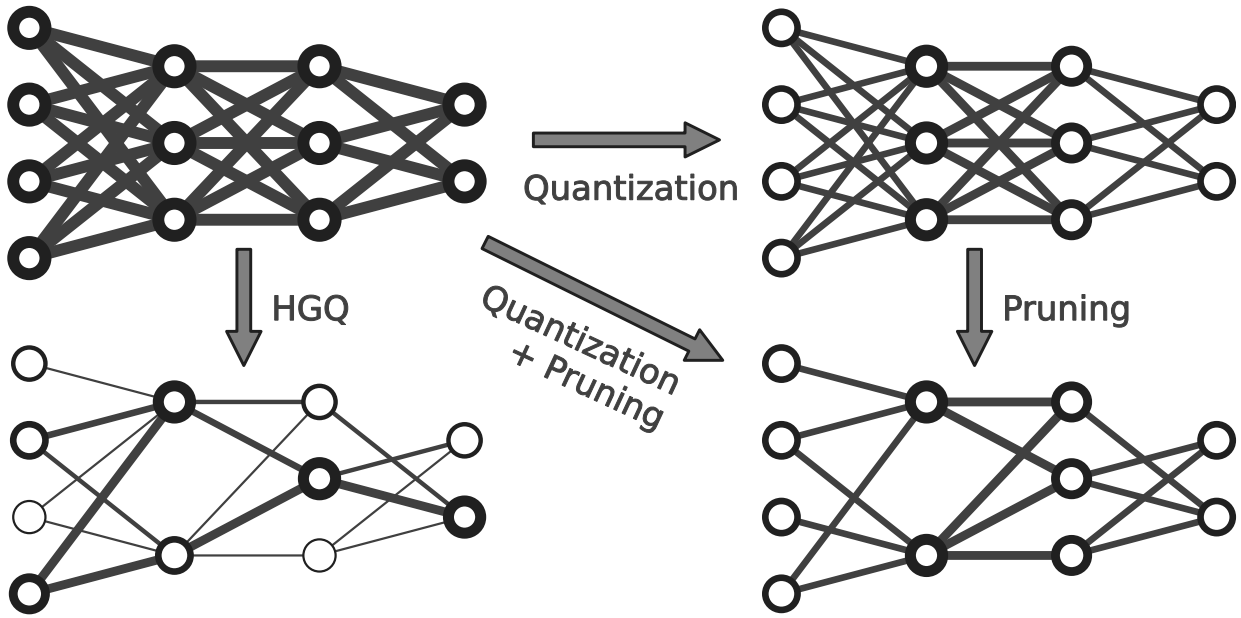
Solution: Allow different bitwidth for different weights/activations.

- **New Challenge**: This creates a huge search space for bitwidths (same as $\mathcal{O}(\#\text{weights})$), and the current methods are not efficient enough to explore it.
- **Solution**: We propose a differentiable quantization method to optimize the bitwidth also via gradient descent along with the weights.

High Granularity Quantization

Proposed Method

We propose **High Granularity Quantization (HGQ)**, a method and codesign framework that assigns an independent learnable bitwidth to every weight and activation



Pruning is included naturally as 0-bit the case and unified with the quantization process.

Differentiable Quantization

Quantization error δ comes from two sources: **1. rounding** and **2.clamping/overflow**.

For 1. assuming

- Rounding-induced error δ_f is uniform in distribution: $\delta_f \sim \mathcal{U}(-2^{-f-1}, 2^{-f-1})$
- The induced loss increment is dominated by magnitude of δ_f: $\mathcal{L}(x, \delta_f) \approx \mathcal{L}(x, |\delta_f|)$

With finite difference and mean-field approximations, we have $\frac{\partial \delta_f}{\partial f} \approx \ln 2 \cdot \delta_f$.

For 2., depending on the overflow mode:

- Saturation-like: direct autodifferentiation on the clamping operation
- Wrap-around: allocate bits to prevent overflow statically with a calibration dataset

Differentiable Resource Surrogate

To enable the optimization of bitwidths, we need also a **differentiable** surrogate for the resource usage estimation on FPGAs.

- differentiable $\rightarrow$ trainable hardware cost

Bit-Operations (BOPs) is a common metric to estimate the resource usage. However, it is not accurate for unrolled models on FPGAs.

We proposed Effective Bit Operations (EBOPs) as a better surrogate for FPGA resource usage estimation:

$$\text{EBOPs} = \sum_{i,j \in \mathcal{M}} b_i \cdot b_j + \sum_{i,j \in \mathcal{A}} \max(b_i, b_j)$$

$$\mathcal{L} = \mathcal{L}_{\text{original}} + \beta \cdot \text{EBOPs} + \gamma \cdot \sum \text{bitwidths}$$

The second term is for regularization of parameters not involved in EBOPs, i.e., preventing them to grow unboundedly and cause numerical issues during training. In general, γ of $\sim 10^{-8}$ is sufficient.

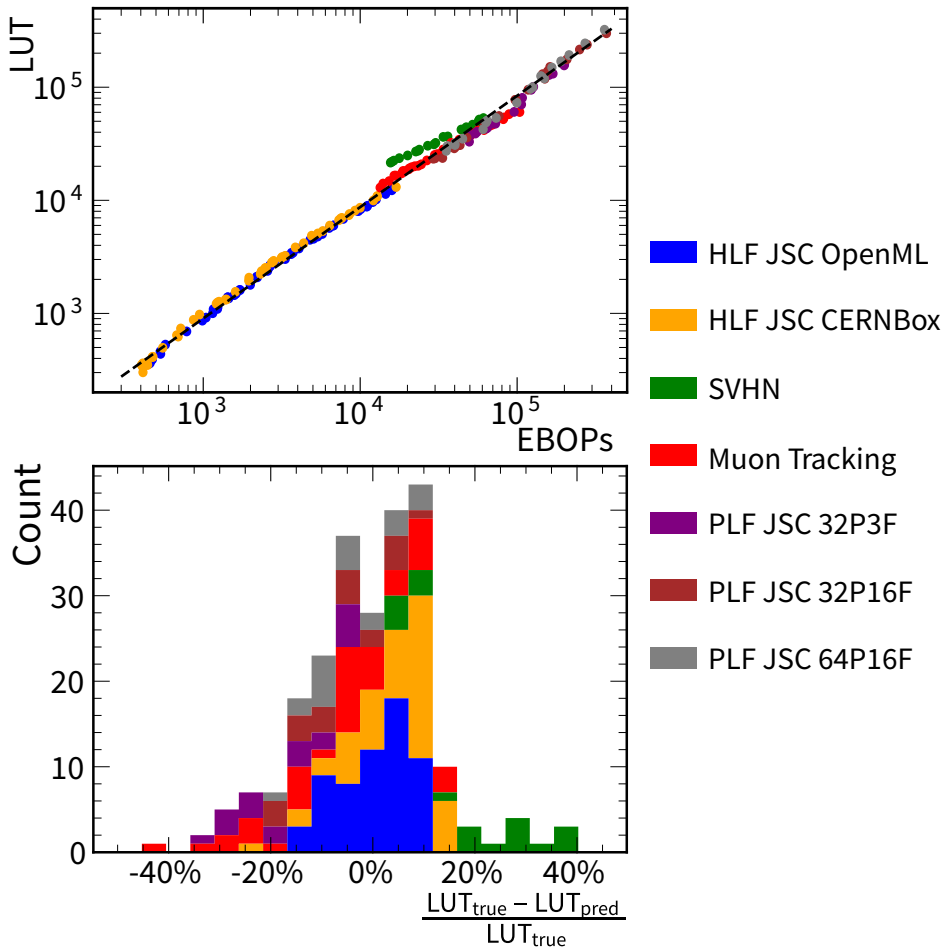
Effective Bit Operations as Resource Estimation Metric

Empirical formulas for Xilinx UltraScale+ FPGAs:

- $\mathrm{LUTs} \approx \exp(0.985 \cdot \log(\mathrm{EBOPs}))$
- The coefficients is expected to change for different FPGA families.

For the Constant-Matrix-Vector-Multiplication (CMVM) dominant models, EBOPs is also found to be a good resource estimation metric.

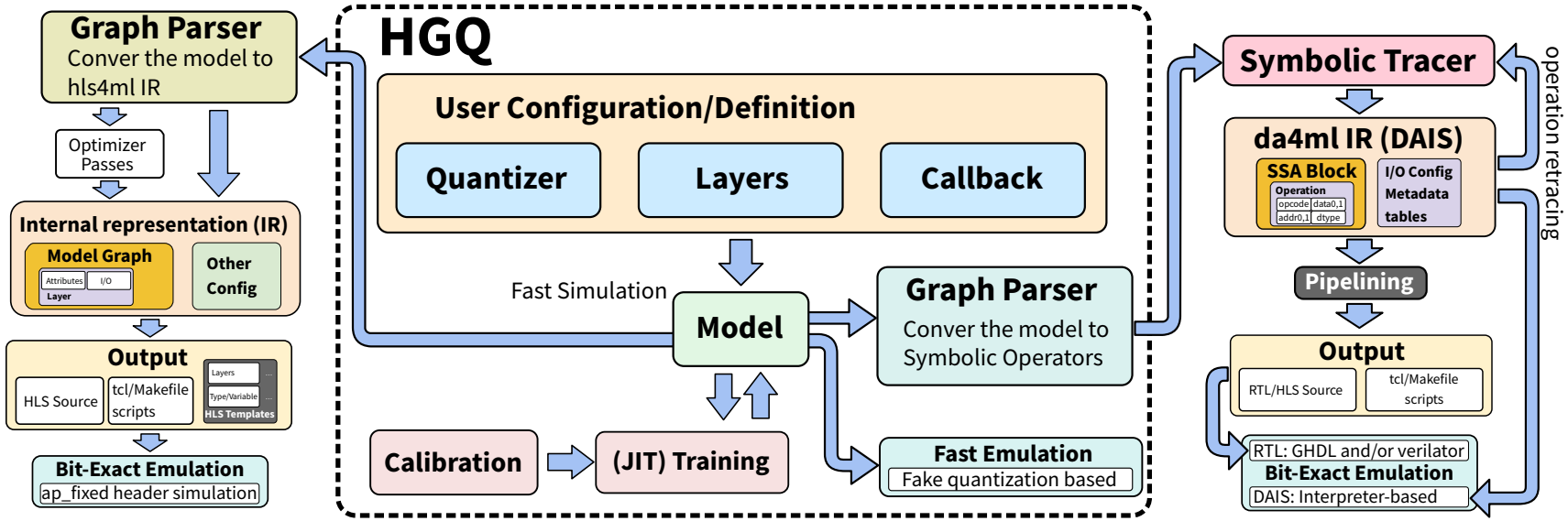
EBOPs can be extended for other operations, such as table-lookups for LUT-based models.



The SVHN samples used includes FIFOs uncounted in EBOPs, which explains the outliers.

Software Implementation

- HGQ is a full hardware co-design frontend instead of just quantization provider
  - Implemented as a Keras v3 based library `HGQ2`, (code [LGPL-3], docs)
- All common operations are supported
  - From `Dense/Conv` to `MultiHeadAttention`, most of the Keras layers
  - For example, one can implement static GNNs [1] or transformers [2] with it



[1] JEDI-Linear: Fast and Efficient Graph Neural Networks for Jet Tagging on FPGAs [FPT'25]

[2] Sub-microsecond Transformers for Jet Tagging on FPGAs [ML4PS'25]

Software Implementation

- Full jit-compilation support for `jax/tensorflow` backend
  - 50%-90% training speed compared to native implementation
  - Kernel launch overhead bounded or memory bounded in general
  - Less overhead for larger models as computation becomes more dominant

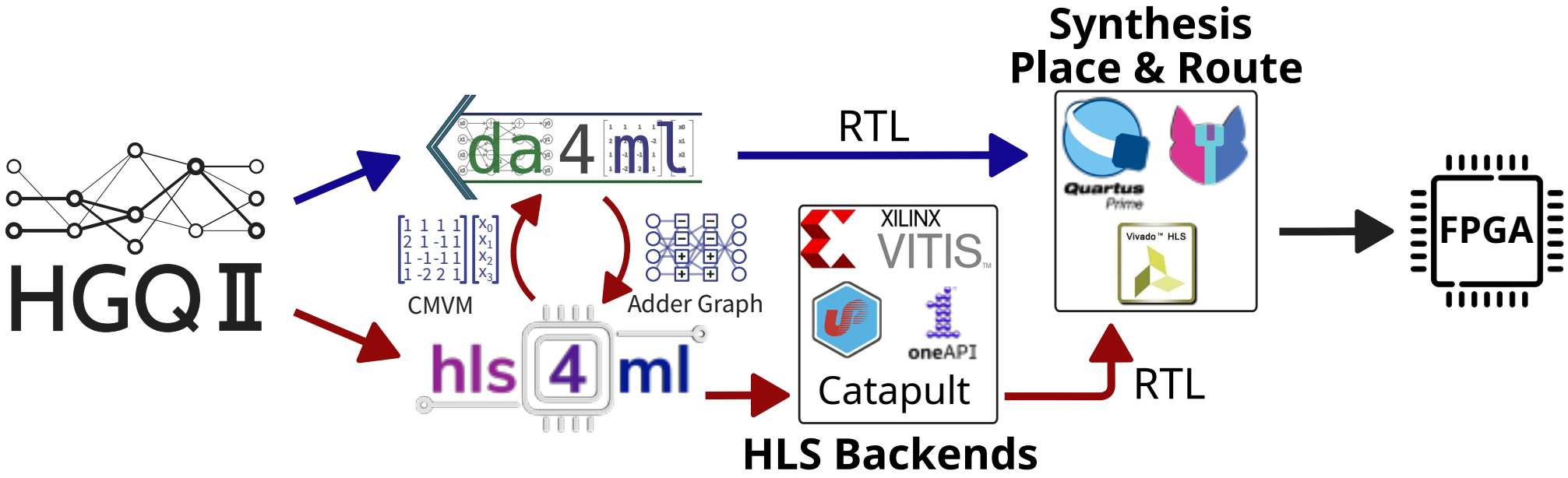
Training speed comparison

Task	Batch size	HGQ	Keras	NLA
HLF JSC	16600	0.645	0.414	164.
JSC PLF (32 particles, 16 features)	2790	1.85	1.25	-
JSC PLF (64 particles, 16 features)	2790	4.10	2.76	-
TGC Muon Tracking	51200	2.30	1.68	-
SVHN Classifier	2048	5.03	3.49	-

Training time in ms/batch for different methods on an NVIDIA RTX4090. NLA is NeuraLUT-Assemble.

Software Implementation

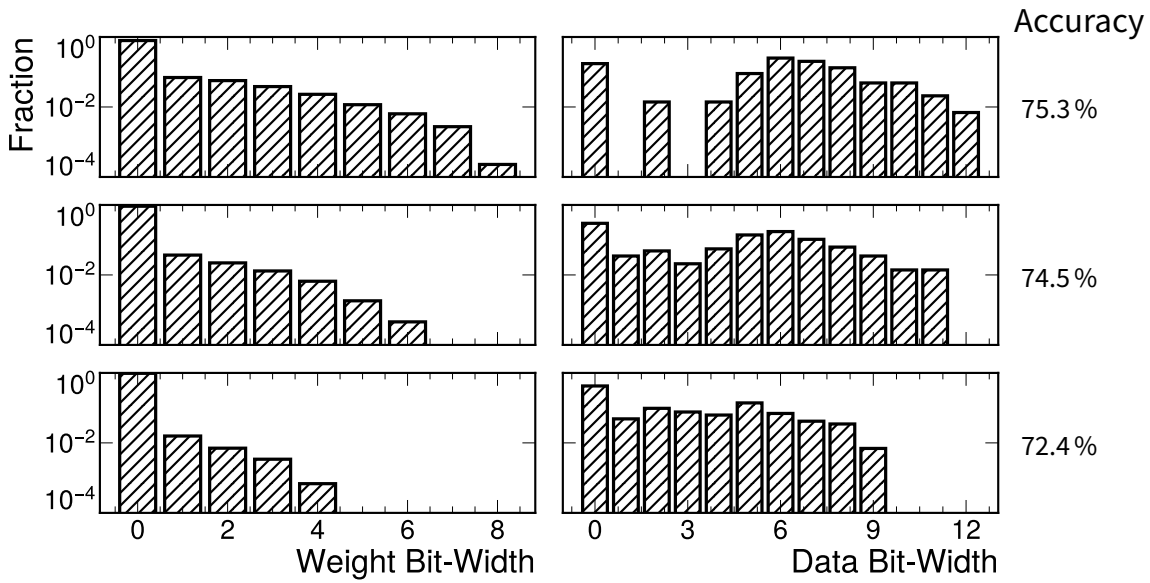
- Support for all operations in `da4ml` and most operations in `hls4ml`
  - Configuration-free (float-point-limit) bit-exact HDL/HLS code generation

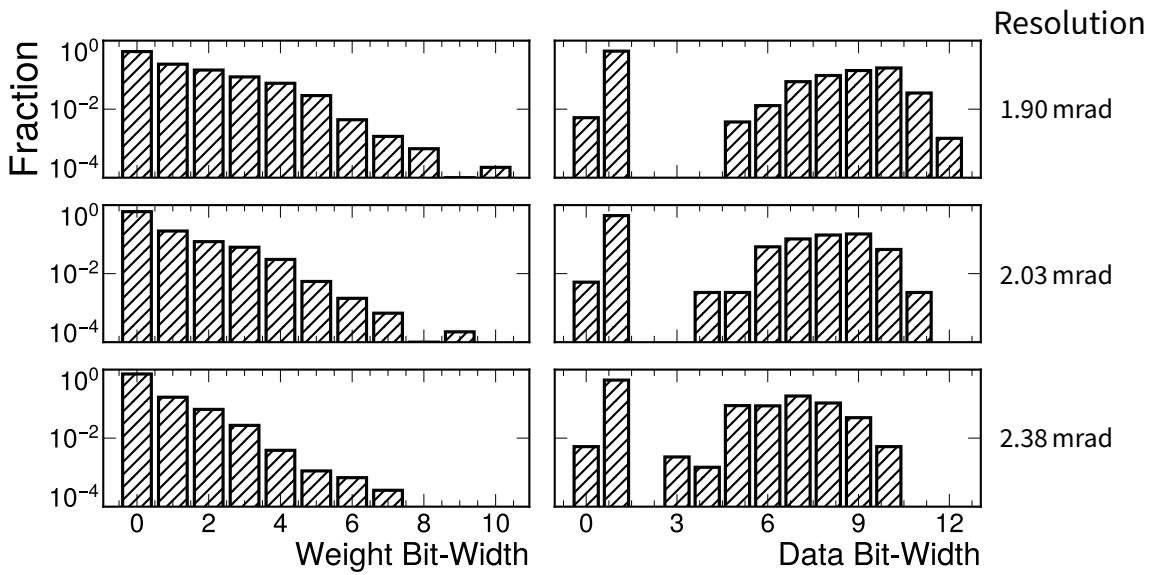


The CMVM based models are implemented with optimized adder trees; heterogeneous bitwidths can be fully utilized for resource saving.

Bit-width obtained

When applying HGQ at per-parameter level, the resultant bitwidth is usually a very sparse and peaked at 0/1 bit (pruned/powers-of-two):

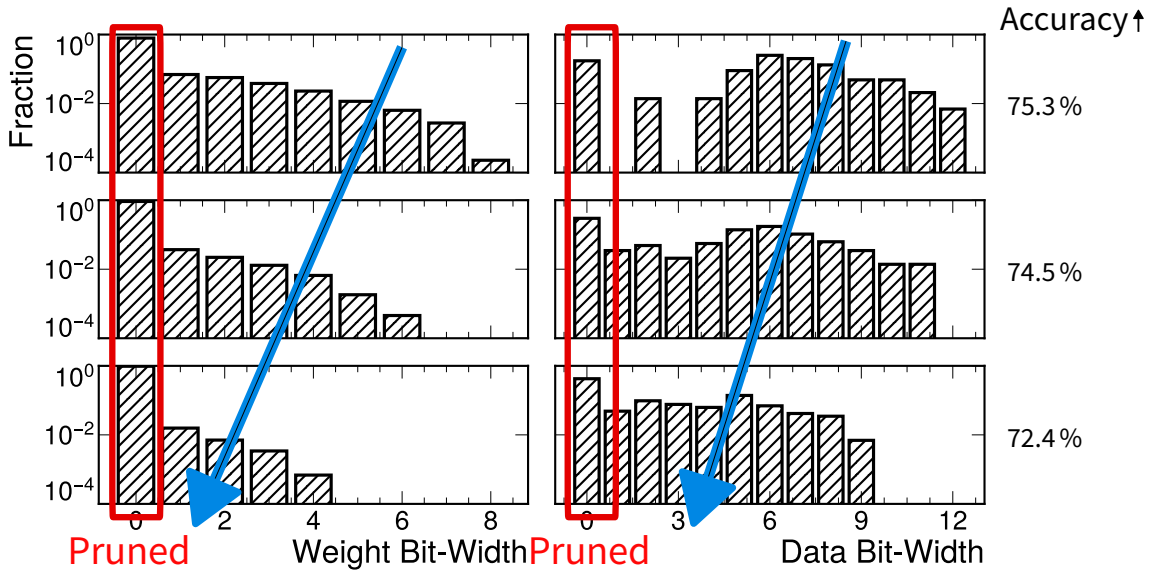


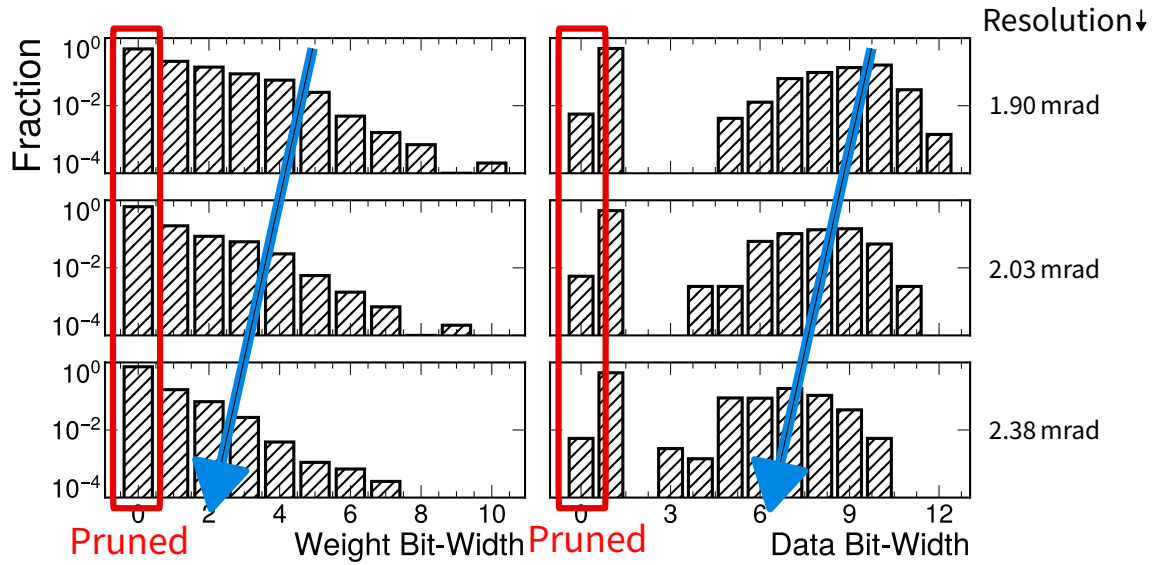


Higher bitwidths are associated with better performance in general

Bit-width obtained

When applying HGQ at per-parameter level, the resultant bitwidth is usually a very sparse and peaked at 0/1 bit (pruned/powers-of-two):





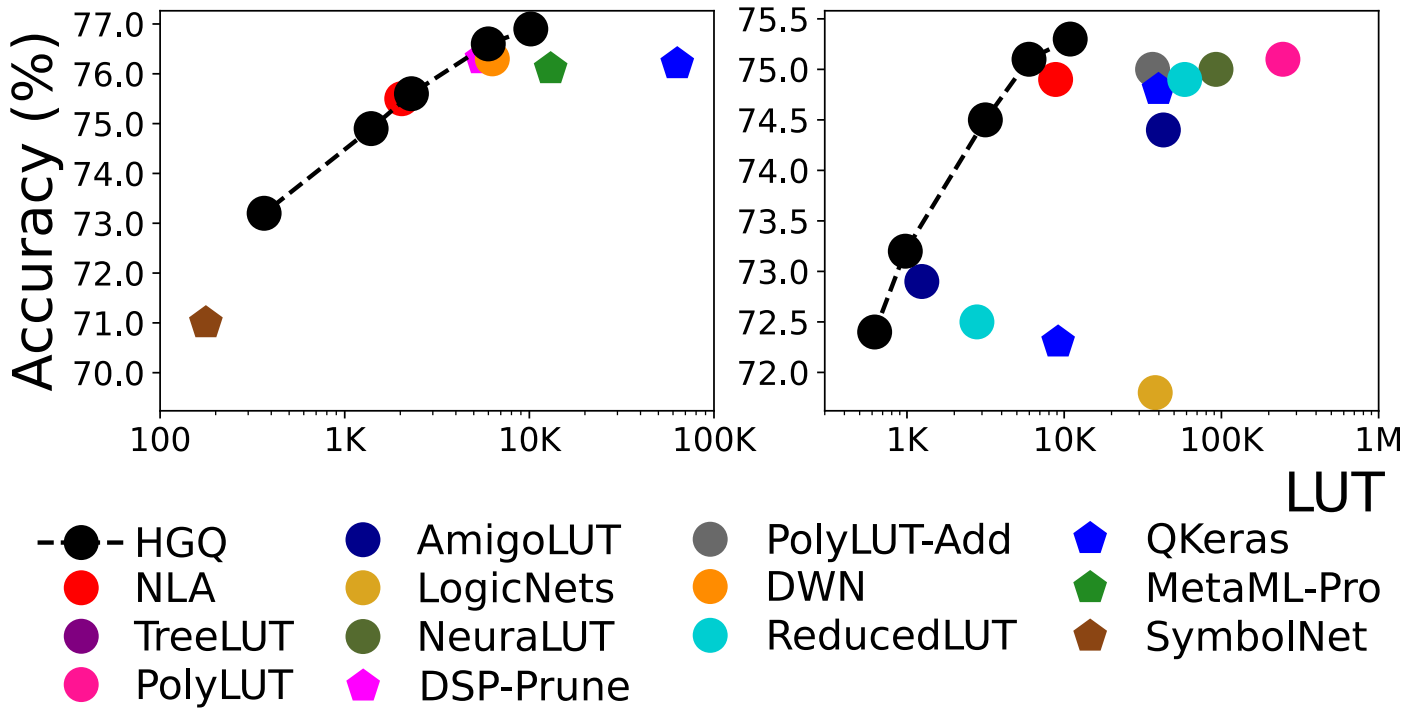
Higher bitwidths are associated with better performance in general

Performance Comparison (JSC HLF)

Comparasion of LUT-accuracy tradeoff for different methods on

- task: Jet-Substructure-Tagging
  - inp: 16 high-level features
  - out: 5 classes
  - 3-layer MLP

HGQ achieves SOTA LUT-accuracy tradeoff on these tasks



- More than 10x reduction compared to uniform quantization
  - Device: AMD UltraScale+ family; xcvu13p or xcvu9p

* NeuraLUT-Assemble (NLA) results are reproduced after addressing the issues listed in the backup section.

Performance Comparison

HGQ with classical neural network on **AMD FPGAs**:

- SOTA resource efficiency
- Longer latency compared to some LUT-based models

Latency/resource profile can be different on different devices families/vendors:

HLF JSC (CERNBox)

Implementation	Accuracy ↑	Latency [cycles]	LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ	75.3%	18 (31.1 ns)	10,921	0	11,183	578.4	1
HGQ	75.1%	15 (24.6 ns)	5,974	0	5,775	609.8	1
HGQ	74.5%	13 (20.4 ns)	3,152	0	2,941	636.9	1
HGQ	73.2%	11 (14.6 ns)	976	0	904	754.1	1
HGQ	72.4%	9 (9.9 ns)	623	0	642	905.8	1
PolyLUT [TC'25]	75.1%	5 (24.6 ns)	246,071	0	12,384	203.	1
PolyLUT-Add [FPL'24]	75.%	5 (15.9 ns)	36,484	0	1,209	315.	1
NLA [FCCM'25]	74.9%	7 (10.3 ns)	8,819	0	2,770	679.8	1
ReducedLUT [FPGA'25]	74.9%	N/A	58,409	0	N/A	302.8	N/A
ReducedLUT [FPGA'25]	72.5%	N/A	2,786	0	N/A	408.5	N/A
QKeras [NMI'21]	74.8%	11 (55 ns)	39,782	124	8,128	~ 200	1
QKeras [NMI'21]	72.3%	11 (55 ns)	9,149	66	1,781	~ 200	1
AmigoLUT [FPGA'25]	74.4%	5 (9.6 ns)	42,742	0	4,717	520.	1
AmigoLUT [FPGA'25]	72.9%	5 (5.0 ns)	1,243	0	1,240	1,008.	1
LogicNets [FPL'20]	71.8%	5 (11.7 ns)	37,931	0	810	427.	1

- Likely due to different CLB layout/adder/carrier configurations
  - e.g., switching board can change latencies of two models from [20/9 ns] to [18/13 ns]

* NeuraLUT-Assemble (NLA) results are reproduced after addressing the issues listed in the backup section.

Conclusion & Takeaways

- We proposed HGQ, a method to efficiently optimize the bitwidth of NN at per-parameter level for unrolled models on FPGAs.
  - Differentiable quantization and accurate area estimation (EBOPs)
- SOTA LUT-accuracy tradeoff, $> 10\times$ LUT reduction to uniform quantization.
- Close to native speed with jit support; $> 200\times$ faster than the SOTA LUT-based model optimization method NLA.
- Will explore LUT-based models with HGQ in the future
  - LUT-based models has better latency than CMVM-based ones for some vendors

End of presentation

Acknowledgements

We would like to thank Maurizio Pierini for insightful discussions. Partial support from the United States DoE (grant numbers DE-SC0011925, DE-FOA-0002705), NSF (grant numbers PHY240298, PHY2117997), United Kingdom EPSRC (grant numbers UKRI256, EP/V028251/1, EP/N031768/1, EP/S030069/1, and EP/X036006/1), Swiss NSF Grant No.~PZ00P2_201594, Schmidt Futures (Grant G-23-64934), KIAT, Intel, and AMD is acknowledged. We acknowledge the Caltech Danny Koh graduate student scholarship and the ETH/Guenther Dissertori partial support for this project.

Full Performance Table - JSC HLF - OpenML

HLF JSC (OpenML)

Implementation	Accuracy ↑	Latency [cycles]	LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ	76.9%	20 (36.3 ns)	10,182	0	10,480	551.6	1
HGQ	76.6%	16 (26.2 ns)	5,991	0	5,890	611.2	1
HGQ	75.6%	12 (18.6 ns)	2,298	0	2,217	645.2	1
HGQ	74.9%	10 (14.5 ns)	1,390	0	1,316	691.6	1
HGQ	73.2%	6 (9.1 ns)	366	0	363	662.7	1
QKeras [ICFPT'23]	76.3%	15 (105.0 ns)	5,504	175	3,036	> 142.9	2
DWN [ICLR'24]	76.3%	10 (14.4 ns)	6,302	0	4,128	695.	1
QKeras [CoRR'21]	76.2%	9 (45)	63,251	38,	4,394	~ 200	1
MetaML-Pro [TRETS'26]	76.1%	10 (50 ns)	13,042	70	N/A	~ 200	1
TreeLUT [FPGA'25]	75.6%	2 (2.7 ns)	2,234	0	347	735.	1
NLA [FCCM'25]	75.5%	2 (3.6 ns)	2,036	0	420	558.0	1
SymbolNet [MLST'25]	71.%	2 (10 ns)	177	3	109	~ 200	1

Full Performance Table - JSC HLF - CERNBox

HLF JSC (CERNBox)

Implementation	Accuracy ↑	Latency [cycles]	LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ	75.3%	18 (31.1 ns)	10,921	0	11,183	578.4	1
HGQ	75.1%	15 (24.6 ns)	5,974	0	5,775	609.8	1
HGQ	74.5%	13 (20.4 ns)	3,152	0	2,941	636.9	1
HGQ	73.2%	11 (14.6 ns)	976	0	904	754.1	1
HGQ	72.4%	9 (9.9 ns)	623	0	642	905.8	1
PolyLUT [TC'25]	75.1%	5 (24.6 ns)	246,071	0	12,384	203.	1
PolyLUT-Add [FPL'24]	75.%	5 (15.9 ns)	36,484	0	1,209	315.	1
NLA [FCCM'25]	74.9%	7 (10.3 ns)	8,819	0	2,770	679.8	1
ReducedLUT [FPGA'25]	74.9%	N/A	58,409	0	N/A	302.8	N/A
ReducedLUT [FPGA'25]	72.5%	N/A	2,786	0	N/A	408.5	N/A
QKeras [NMI'21]	74.8%	11 (55 ns)	39,782	124	8,128	~ 200	1
QKeras [NMI'21]	72.3%	11 (55 ns)	9,149	66	1,781	~ 200	1
AmigoLUT [FPGA'25]	74.4%	5 (9.6 ns)	42,742	0	4,717	520.	1
AmigoLUT [FPGA'25]	72.9%	5 (5.0 ns)	1,243	0	1,240	1,008.	1
LogicNets [FPL'20]	71.8%	5 (11.7 ns)	37,931	0	810	427.	1

Full Performance Table - JSC PLF

PLF JSC (3 features)

Implementation	Particles	Accuracy ↑	Latency [cycles]	LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ (GNN)	32	79.2%	24 (108.8 ns)	159,238	0	91,027	220.5	1
HGQ (GNN)	32	78.5%	20 (72.3 ns)	80,618	0	42,101	276.5	1
`QKeras, DS` [MLST'24]	32	$\leq$ 75.9%	26 (130 ns)	903,284	434	358,754	$\sim$ 200	2
`QKeras, GNN` [MLST'24]	32	$\leq$ 75.8%	32 (160 ns)	1,162,104	2,120	761,061	$\sim$ 200	3

PLF JSC (16 features)

Implementation	Particles	Accuracy ↑	Latency [cycles]	(A)LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ (GNN)	64	82.4%	26 (122.5 ns)	244,515	0	112,993	212.2	1
HGQ (GNN)	64	80.7%	9 (45.0 ns)	53,546	0	13,629	199.9	1
HGQ (GNN)	32	81.5%	26 (119.8 ns)	238,255	0	116,039	217.1	1
HGQ (GNN)	32	80.2%	9 (45.5 ns)	48,343	0	10,012	197.7	1
GNN U4 (TECS'24)	50	80.9%	130 (650 ns)	855k	8,945	201k	$\sim$ 200	100
GNN U5 (TECS'24)	50	81.2%	181 (905 ns)	815k	8,986	189k	$\sim$ 200	150
GNN J4 (TECS'24)	30	78.4%	58 (290 ns)	865k	8,776	138k	$\sim$ 200	30
GNN J5 (TECS'24)	30	79.9%	181 (905 ns)	911k	9,833	158k	$\sim$ 200	150
GNN (FPL'22)	50	80.4%	2132 (10660 ns)	1515k	12,284	533k	$\sim$ 200	650
GNN (FPL'22)	30	78.7%	382 (1910 ns)	1158k	11,504	246k	$\sim$ 200	400

Full Performance Table - TGC and SVHN

Muon tracking

Implementation	Resolution ↓	Latency [cycles]	LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ	1.90 mrad	8 (47.4 ns)	41,830	0	10061	168.9	1
HGQ	2.03 mrad	6 (35.2 ns)	25,716	0	3455	170.3	1
HGQ	2.38 mrad	5 (28.7 ns)	14,789	0	3091	174.1	1
QKeras [NIMA'23]	1.95 mrad	17 (106.3 ns)	37,867	1,762	8,443	> 160	1
QKeras [NIMA'23]	2.04 mrad	13 (81.3 ns)	54,638	324	6,525	> 160	1
QKeras [NIMA'23]	2.45 mrad	10 (62.5 ns)	28,526	24	2,954	> 160	1

SVHN classification

Implementation	Accuracy ↑	Latency [cycles]	LUT	DSP	FF	F_{max} [MHz]	II [cycles]
HGQ (hls4ml)	93.8%	1,048 (5319.6 ns)	66,056	52	24,207	197.0	1,029
HGQ (hls4ml)	92.0%	1,060 (5272.4 ns)	41,733	20	18,774	201.0	1,030
QKeras [MLST'21]	94.%	1,035 (5,175 ns)	111,152	174	32,554	~ 200	1,030
QKeras [MLST'21]	88.%	1,059 (5,295 ns)	38,795	70	14,802	~ 200	1,029
DSP-Prune [ICFPT'23]	92.4%	5,447 (43,576 ns)	59,279	1,215	46,584	> 125	N/A

Differentiable quantization (simplified)

Let x_q be a fixed point number with sign `s`, `i` integer bits (ex. sign), and `f` fractional bits: $x_q = \mathrm{quant}(x, s, i, f) = x + \delta$.

$\frac{\partial \mathcal{L}}{\partial \delta}$ is available from backpropagation where $\mathcal{L}$ is the loss, we need only $\frac{\partial \delta}{\partial f}$ and $\frac{\partial \delta}{\partial i}$.

The quantization error δ come from two sources: **1. rounding** and **2.clamping/overflow**.

For rounding, **assume** the error δ_f is uniform: $\delta_f \sim \mathcal{U}(-2^{-f-1}, 2^{-f-1})$ **and** loss is insensitive to the direction of the error $\mathcal{L}(x, \delta_f) \approx \mathcal{L}(x, |\delta_f|)$.

Using an alternative form of the finite difference approximation $\frac{\partial |\delta_f|}{\partial f} \approx \ln \frac{|\delta_{f+1}|}{|\delta_f|} \cdot |\delta_f|$.
With mean-field approximation, we replace the first term by its expectation $\ln 2$, and we have $\frac{\partial \delta_f}{\partial f} \approx \ln 2 \cdot \delta_f$.

Differentiable quantization (simplified)

For the overflow part, the overflow mode matters:

- For "saturation" (clamp-like) quantization, we direct consider the value of the clamping point as the value of x when overflow happens.
  - i.e., direct autodifferentiation on $\mathrm{clamp}(x, -2^i, 2^i - 2^{-f})$
- For "wrap-around" (dropping MSBs) quantization, the actual quantization error is too noisy to be useful. We instead trace the number of bits needed to represent the any seen value during training to **prevent overflow statically**.

Together, we have the surrogate gradient for the bitwidths directly derived from the loss.

Reproduction of NeuraLUT-Assemble (NLA) Results

We found 4 issues in the NeuraLUT-Assemble's official codebase that are expected to lead to biased evaluations. They are fixed for our reproduction of the results.

- Use of test set for model selection during training (task data leakage)
  - train.py#L201-L203, train.py#L202-L204
- Contamination of training and testing set for the JSC HLF OpenML task due to unseeded random split (task data leakage)
  - neq2lut.py#L1-L245, README.md#L30-L46, demo.ipynb
- Skip of clamping operation on hardware for the inputs of the models (missing logic in the design)
  - models.py#L60-L76, models.py#L297, outputs of demo.ipynb
- The missing output register for timing analysis in out of context place and route (missing paths for timing analysis)
  - verilog.py#L48-L65, outputs of demo.ipynb