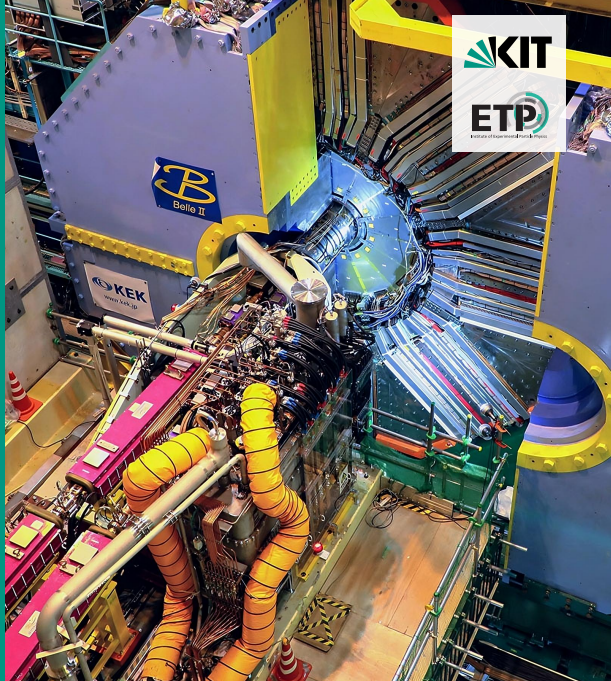
Challenges Of Hardware-Software Co-Design Of A Graph Neural Network For The ECL Trigger

Belle II Germany Meeting 2025

Frank Baptist* (frank.baptist@student.kit.edu), Isabel Haide*, Marc Neu+, Thomas Lobmaier*, Jürgen Becker+ Torben Ferber*

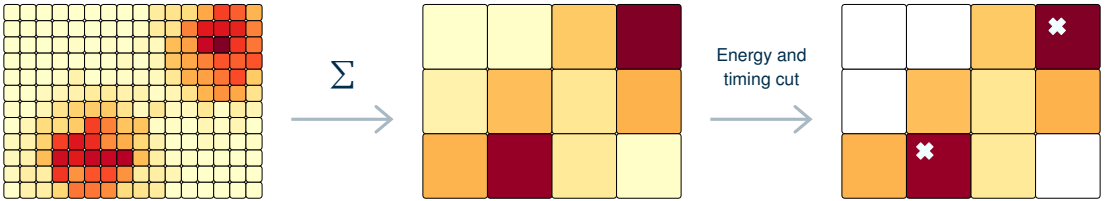
*Institute of Experimental Particle Physics (ETP)
+Institut fuer Technik der Informationsverarbeitung (ITIV)

September 9, 2025

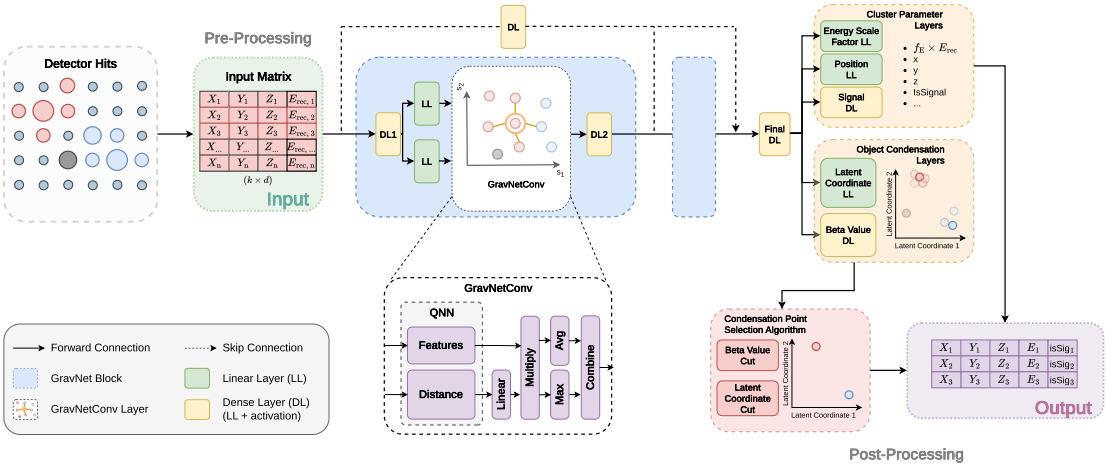


The GNN-ETM

- Proposed upgrade for the L1 trigger ECL clustering system
- One-shot algorithm for detecting and reconstructing clusters
- 8736 ECL crystals are grouped into 576 trigger cells (TC)
- After energy and timing cuts, up to 32 TCs are used as input
- Graph Neural Network (GNN) architecture because of unordered, very sparse input and irregular TC geometry
- GravNet: GNN architecture that dynamically builds a graph from the input data and connects closest neighbors



Implementation

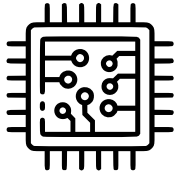


Talk Isabel Haide: https://indico.belle2.org/event/14576/contributions/98794/

Hardware Implementation

Network is developed and trained in Python using `QKeras`

Translation into High Level Synthesis (HLS) C++ code



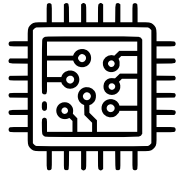
Implementation on FPGA hardware

Python Logo: python.org, C++ Logo: isocpp, FPGA Icon: embien.com

Hardware Implementation

Network is developed and trained in Python using `QKeras`

Translation into High Level Synthesis (HLS) C++ code



Implementation on FPGA hardware

- Cosmics data taking has shown that the hardware is not doing exactly what our python model does
- We have a simulation generated from the HLS C++ (C-SIM) that does exactly what the hardware does
- No interaction with the hardware is necessary from our side to test our models

Python Logo: python.org, C++ Logo: isocpp, FPGA Icon: embien.com

Fixed Point Calculation and Quantization

Python: Floating Point Arithmetics

Example: 16 bit floating point representation of 8.75_{10}

$$\text{value} = \text{Sign} \cdot \text{Mantissa} \cdot 2^{\text{Exponent}}$$

- Wide range of representable numbers
- Different precision in different ranges
- Not feasible for FPGA implementation due to resource constraints

Fixed Point Calculation and Quantization

Python: Floating Point Arithmetics

Example: 16 bit floating point representation of 8.75_{10}

$$\text{value} = \text{Sign} \cdot \text{Mantissa} \cdot 2^{\text{Exponent}}$$

- Wide range of representable numbers
- Different precision in different ranges
- Not feasible for FPGA implementation due to resource constraints

Hardware/C-SIM: Fixed Point Arithmetics

- Numbers are represented using two's complement

$8.75_{10} = 8 + 0.5 + 0.25 = 2^3 + 2^{-1} + 2^{-2}$

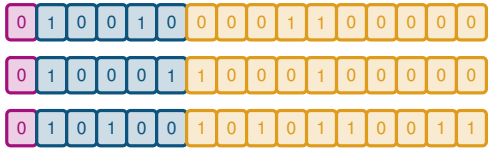
$=$ 

- Example: Q5.2 fixed point number
  - 7 bits total
  - 5 bits for integer part (including sign bit)
  - 2 bits for fractional part
- Fixed precision for the entire quantization range

Multiplication Example

$$8.75_{10} \times 6.125_{10} = 53.593\,75_{10}$$

Floating Point

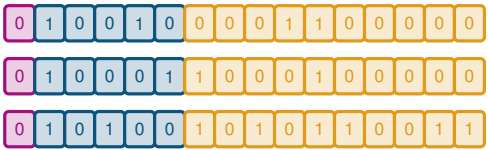


- Result of 2 floating point numbers is another floating point number (same bit width)
- Multiple operations needed on FPGA: multiply mantissas, add exponents, determine sign, normalize result
- Little to no loss of precision

Multiplication Example

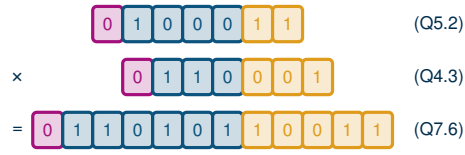
$$8.75_{10} \times 6.125_{10} = 53.593\,75_{10}$$

Floating Point



- Result of 2 floating point numbers is another floating point number (same bit width)
- Multiple operations needed on FPGA: multiply mantissas, add exponents, determine sign, normalize result
- Little to no loss of precision

Fixed Point

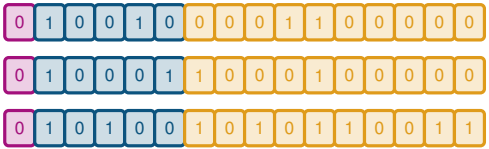


- After mutlipyling, additional bits are needed to keep precision
- Very efficient implementation on FPGA: numbers are treated as integers
- Saturation and rounding to avoid exploding bit widths
- Significant loss of precision

Multiplication Example

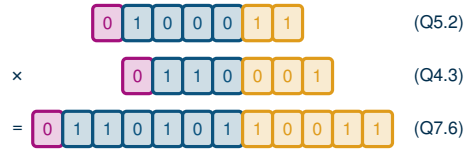
$$8.75_{10} \times 6.125_{10} = 53.593\,75_{10}$$

Floating Point



- Result of 2 floating point numbers is another floating point number (same bit width)
- Multiple operations needed on FPGA: multiply mantissas, add exponents, determine sign, normalize result
- Little to no loss of precision

Fixed Point

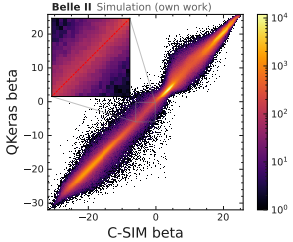


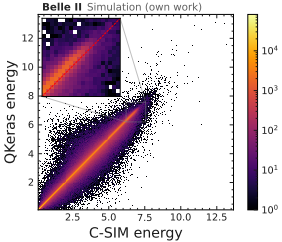
- After mutlipyling, additional bits are needed to keep precision
- Very efficient implementation on FPGA: numbers are treated as integers
- Saturation and rounding to avoid exploding bit widths
- Significant loss of precision

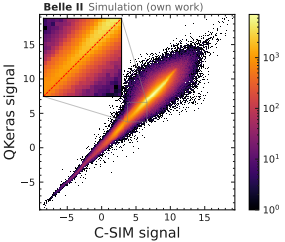
`QKeras`

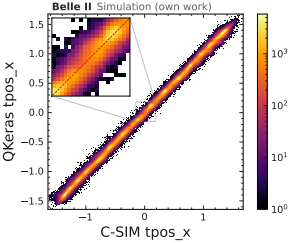
Quantization extension to Keras that is trying to mimic fixed point calculations by artificially quantizing the weights, inputs and outputs of each layer to fixed point numbers.

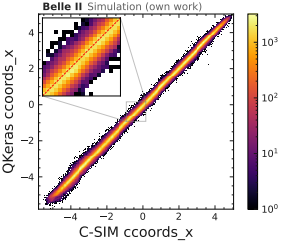
C-SIM Agreement











- Agreement is not good enough
- Why are some features so much worse than others?
  - Beta
  - Energy
  - Signal

Dense Layers And Linear Layers

Example: 5 inputs, 4 outputs

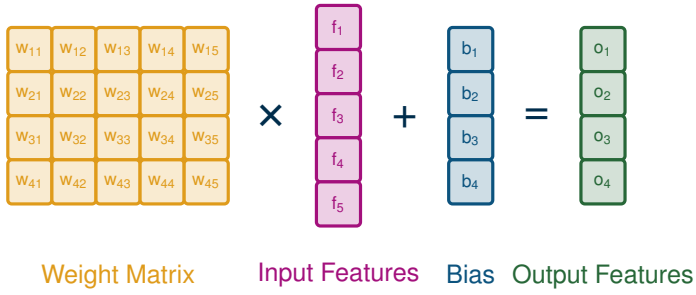
Linear layer (LL):

1. Input features are multiplied with a weight matrix.
2. Biases are added yielding the output features.

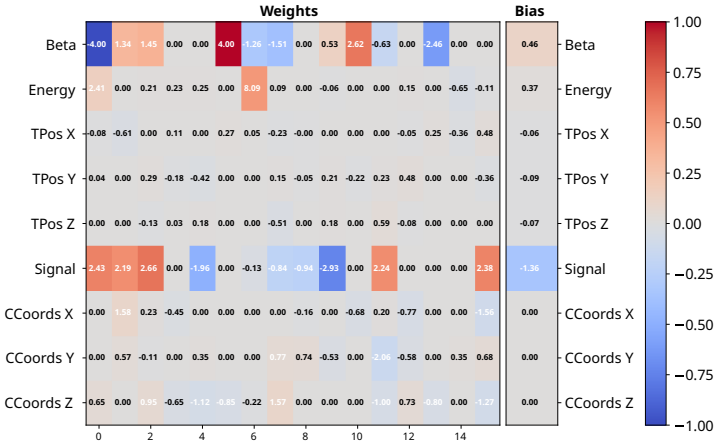
Dense layer (DL):

3. The output features are passed through an activation function to introduce non-linearity.

Weights and biases are learned during training.

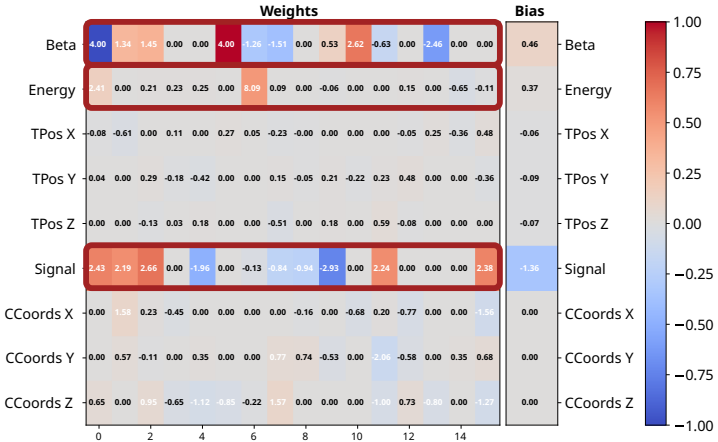


Weights Of Output Dense Layer



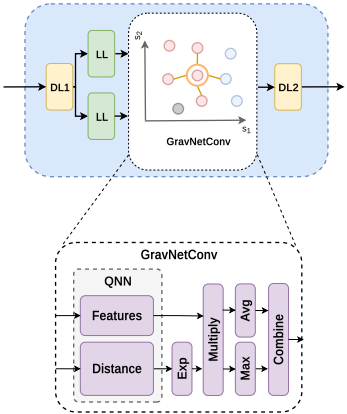
- The colors indicate how close the weights are to their quantization limits
- The output features with the worst agreement have weights that are very close to their quantization limits

Weights Of Output Dense Layer

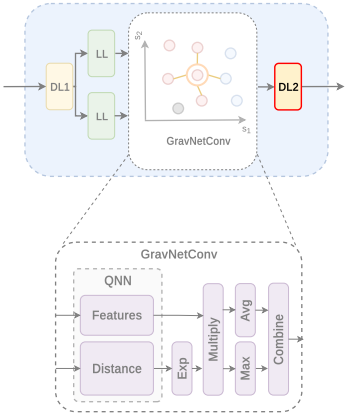


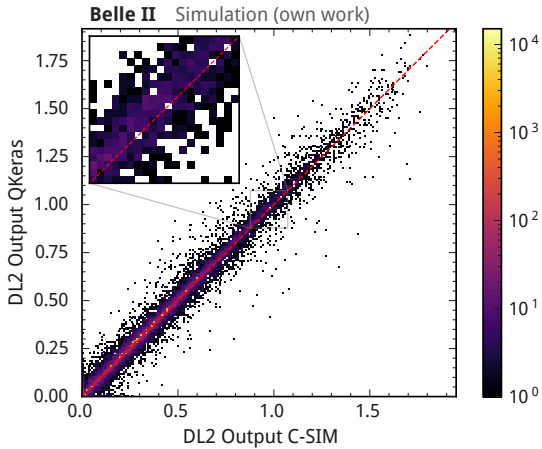
- The colors indicate how close the weights are to their quantization limits
- The output features with the worst agreement have weights that are very close to their quantization limits

Debugging GravNet Block Layerwise

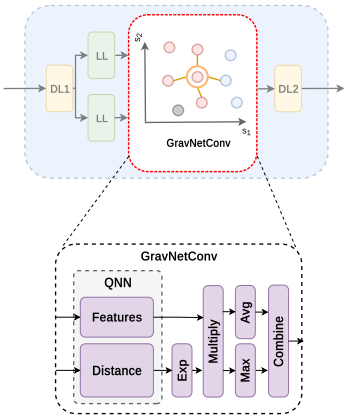


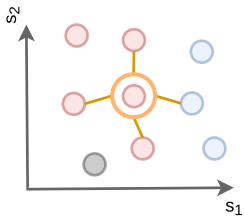
Agreement After The GravNetConv Layer





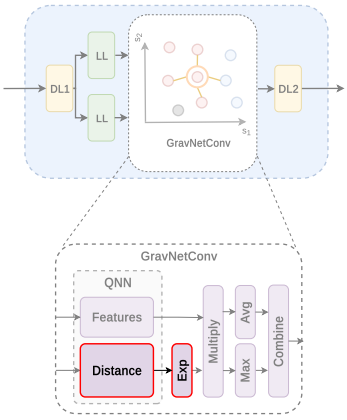
Inside GravNetConv Layer

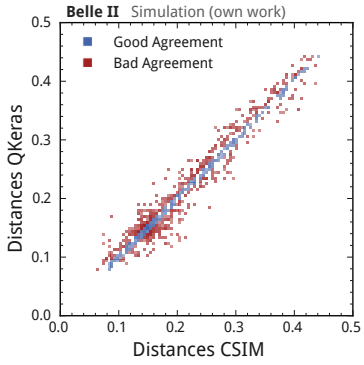




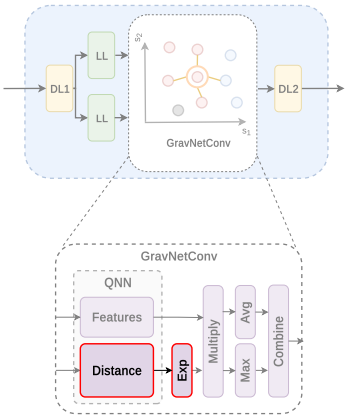
- Graph is dynamically built by dense and linear layers
- Neighboring nodes are connected
- Their distances are weighted by an inverse exponential function.

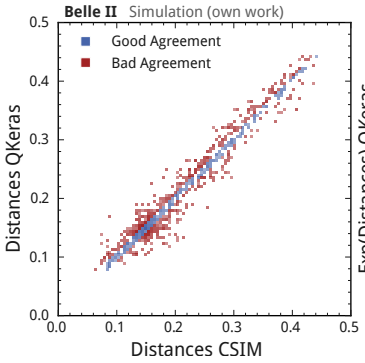
GravNetConv Distances Before Weighting

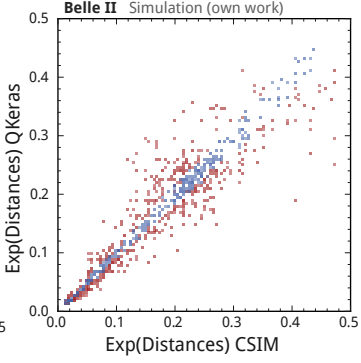




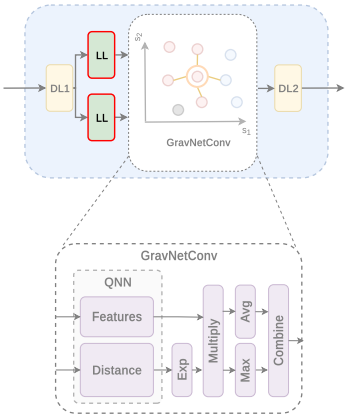
GravNetConv Distances After Weighting

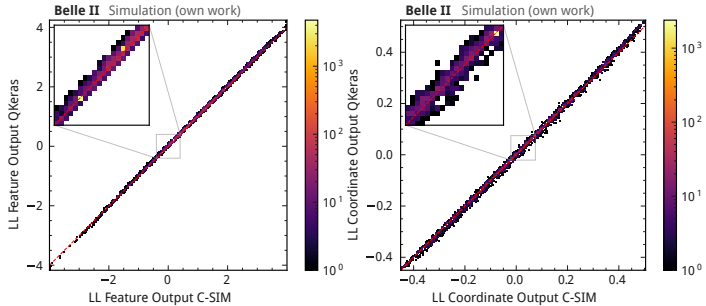




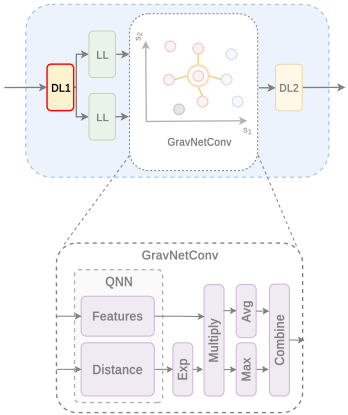


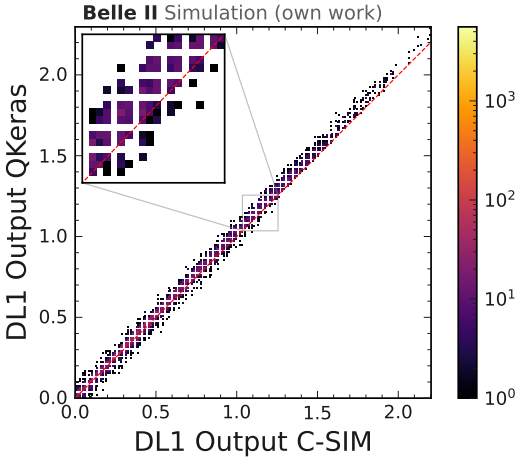
Agreement Before GravNetConv Layer



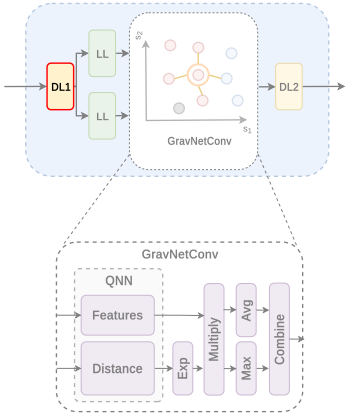


Agreement In First Dense Layer





Reweighting In C-SIM

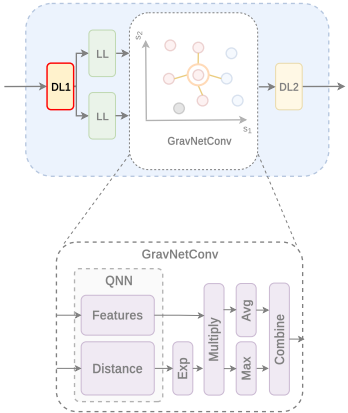


- In `QKeras` we normalize the input features to the range [–1, 1].
- In C-SIM we do not want to add a normalization layer, since it would add overhead.
- Instead we modify the weights of the first dense layer to account for the normalization.

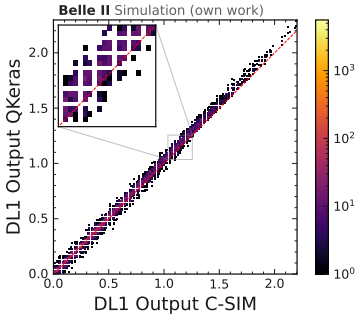
$$F_{out} = W \cdot \frac{F_{in} - \text{offset}}{\text{scale}} + B \quad \rightarrow \quad F_{out} = \frac{W}{\text{scale}} \cdot F_{in} + (\text{scale} \cdot B + \text{offset})$$

- Problem: The input features are quantized to 16 bit, but the weights of the first dense layer are only quantized to 8 bit.
  → Much more rounding errors in the first dense layer in C-SIM than in `QKeras`.
- We have to prescale the inputs in C-SIM as well

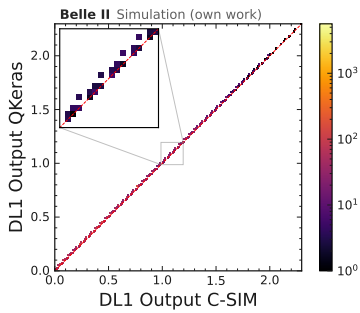
Reweighting Fix



Before Reweighting Fix

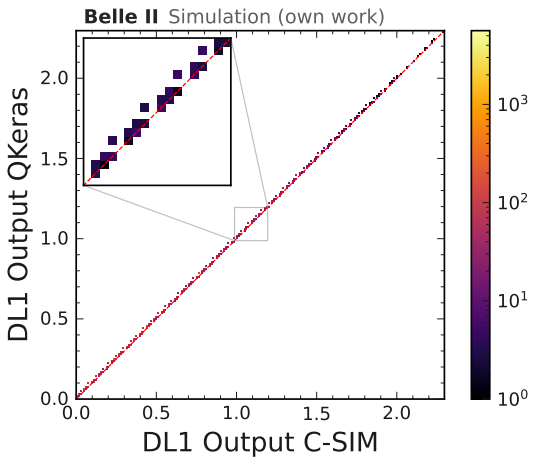


After Reweighting Fix

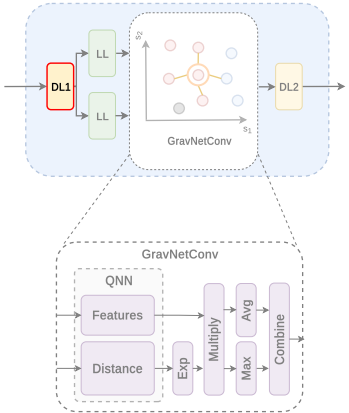


After Reweighting Fix

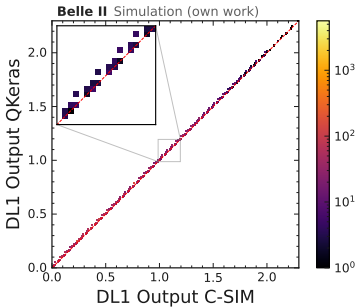
- Still, `QKeras` is constantly higher than C-SIM
- `QKeras` internally uses floating point numbers and artificially quantizes the output by rounding it to the closest legal value
- Flooring instead of rounding would describe the C-SIM behavior better



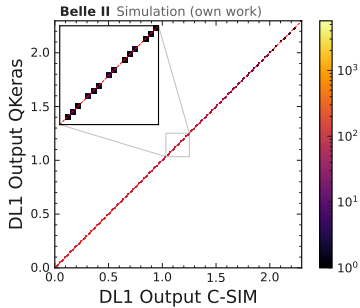
Rounding Fix



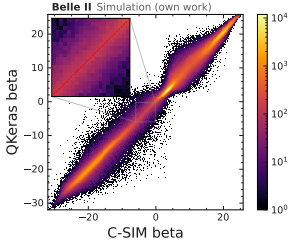
Before Rounding Fix

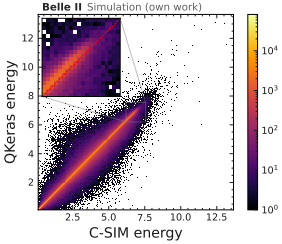


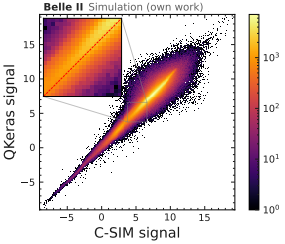
After Rounding Fix

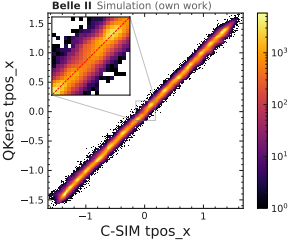


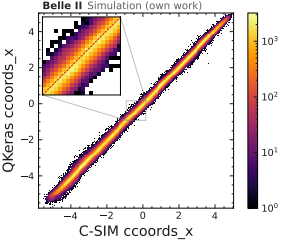
Original C-SIM Agreement



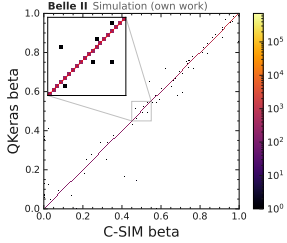


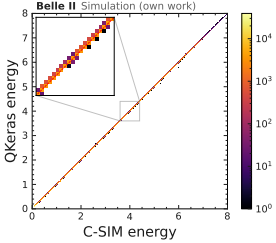


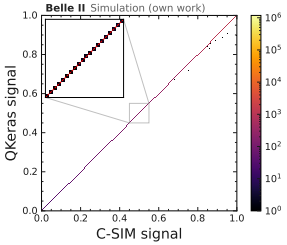


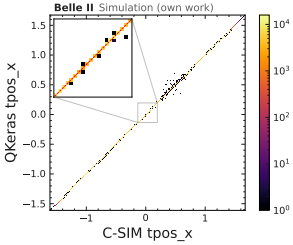


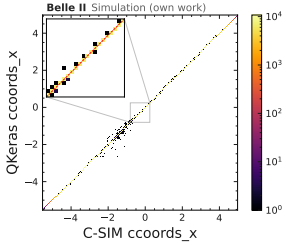
New C-SIM Agreement











Summary And Outlook

- We have a working GNN-ETM that can find clusters in the ECL.
- The FPGA hardware and C Simulation are doing what our `QKeras` model is doing.
- We plan to have a faster final model ready for the upcoming data taking in october.