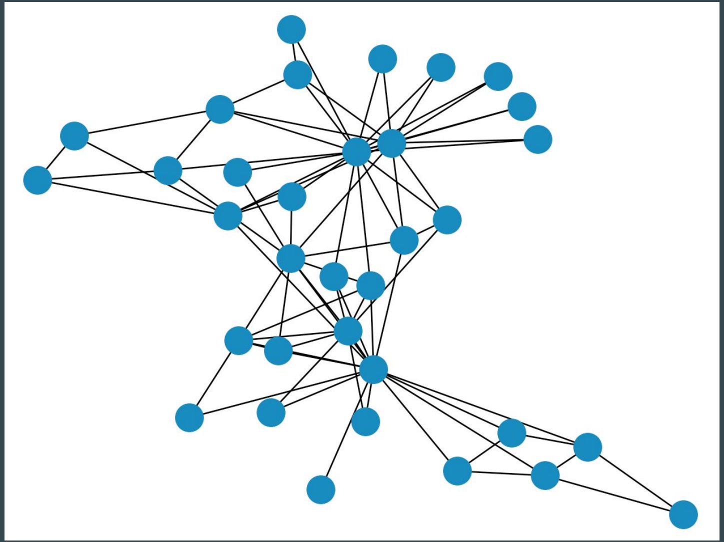
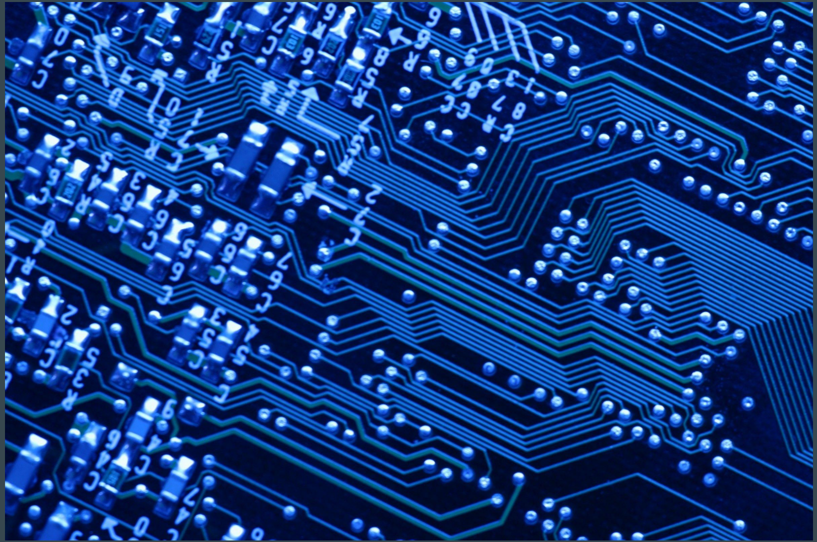
PyTorch Geometric to HLS

An hls4ml add-on for conversion of Graph Neural Networks

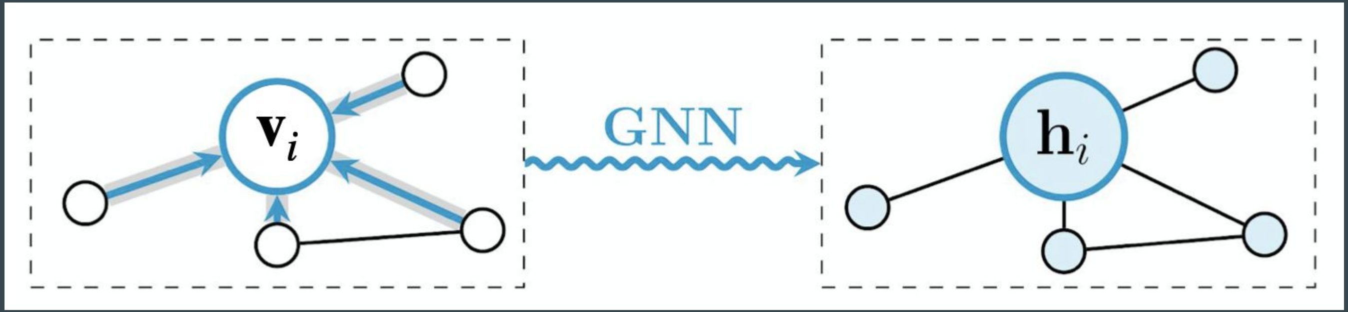




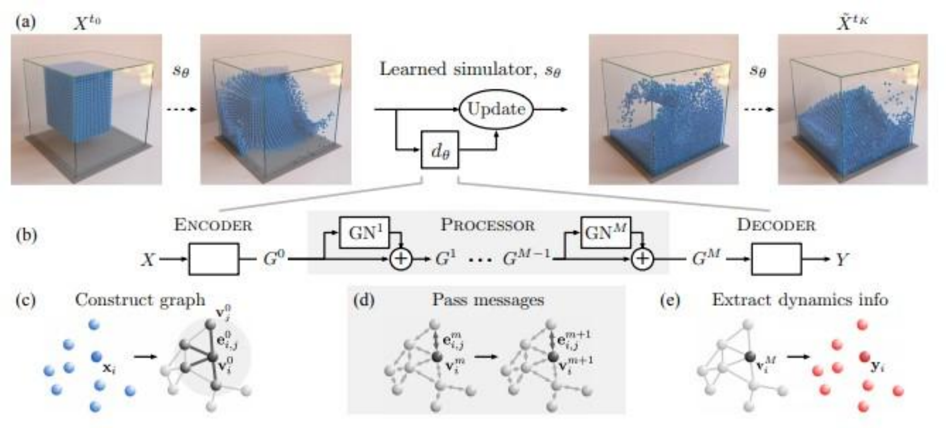
https://github.com/abdelabd/hls4ml/tree/pyg_to_hls_rebase

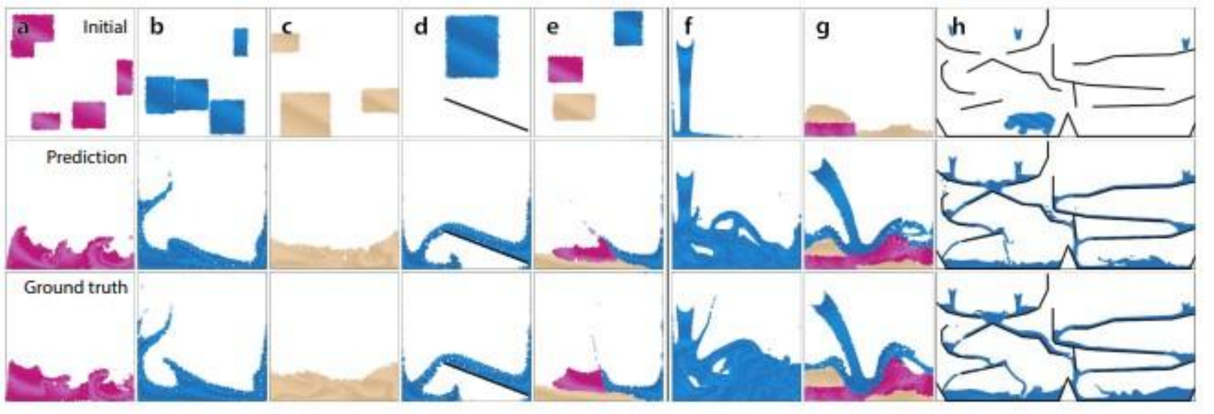
Graph Neural Networks

- G = (V, E, edge_index);
    - V[i,j] = jth attribute of ith vertex
    - E[i,j] = jth attribute of ith edge
    - edge_index[0,i] = the index of the sending node for the ith edge
    - edge_index[1,i] = the index of the receiving node for the ith edge
- We can represent arbitrary-dimensional objects as ‘nodes’
- We can represent arbitrary-dimensional relationships as ‘edges’
- Benefits: efficient encoding of context, generalizability, permutation invariance



Generalizable, Contextual

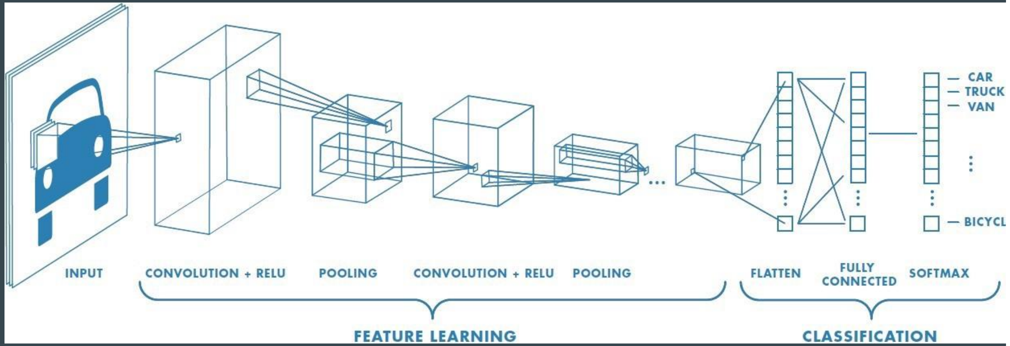


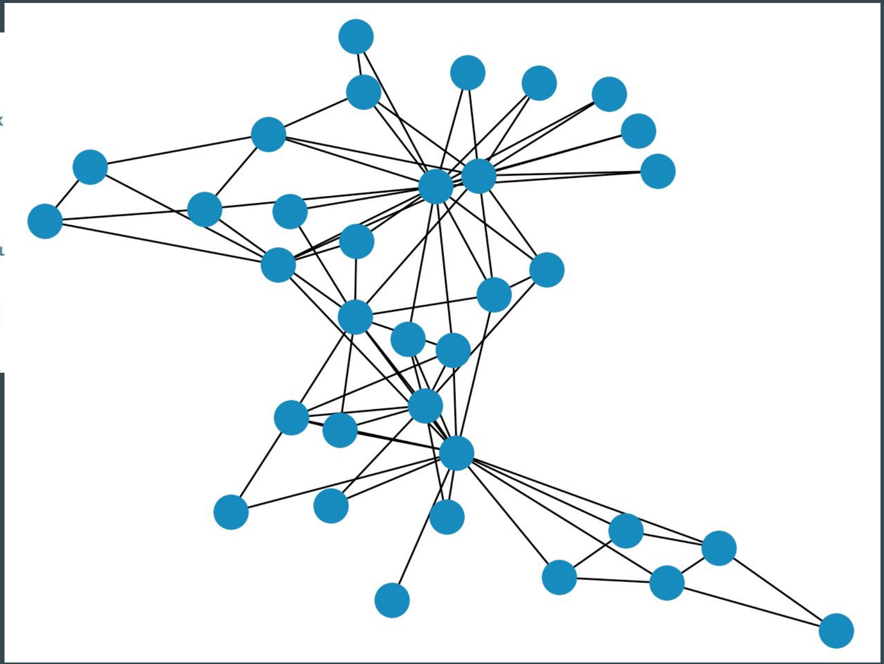


Learning to Simulate Complex Physics with Graph Networks, DeepMind 2020

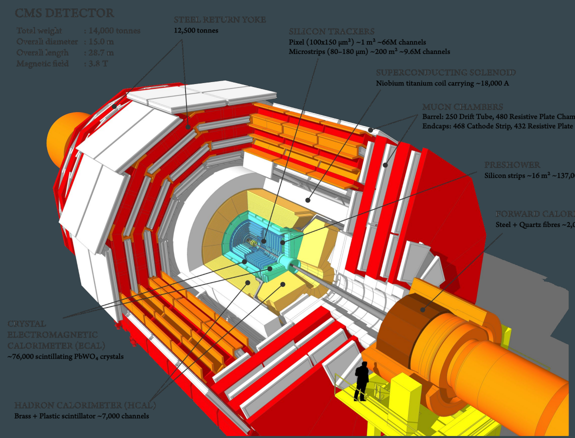
https://arxiv.org/abs/2002.09405

Permutation Invariant

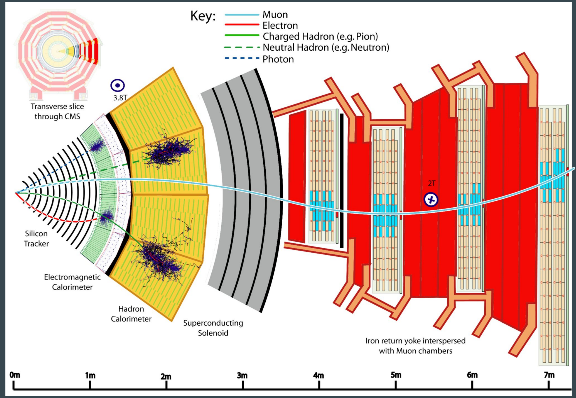


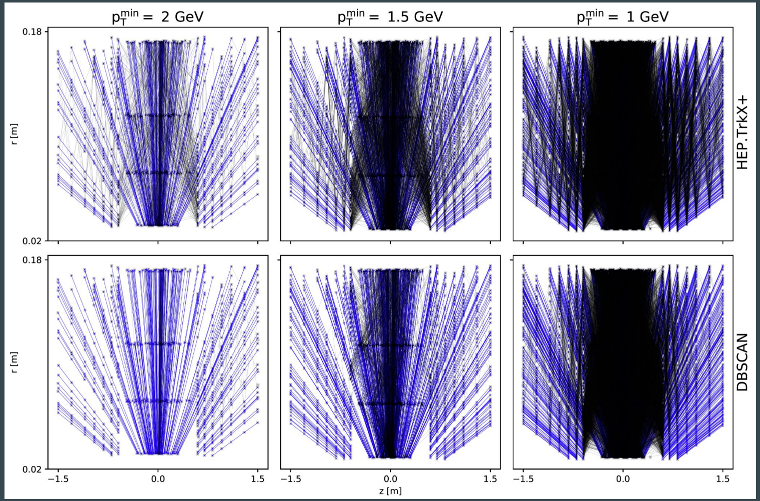


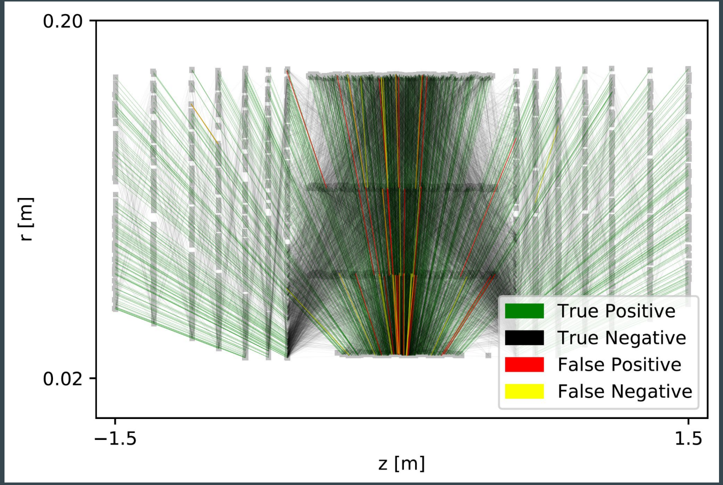
Institute for Research & Innovation in Software for High Energy Physics



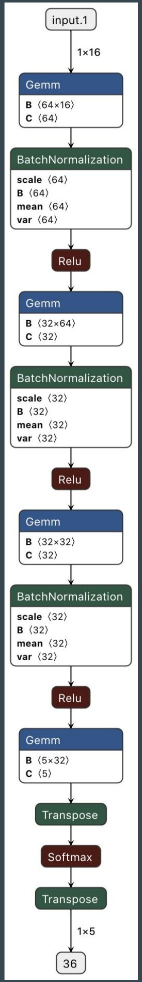
https://arxiv.org/abs/2103.16701

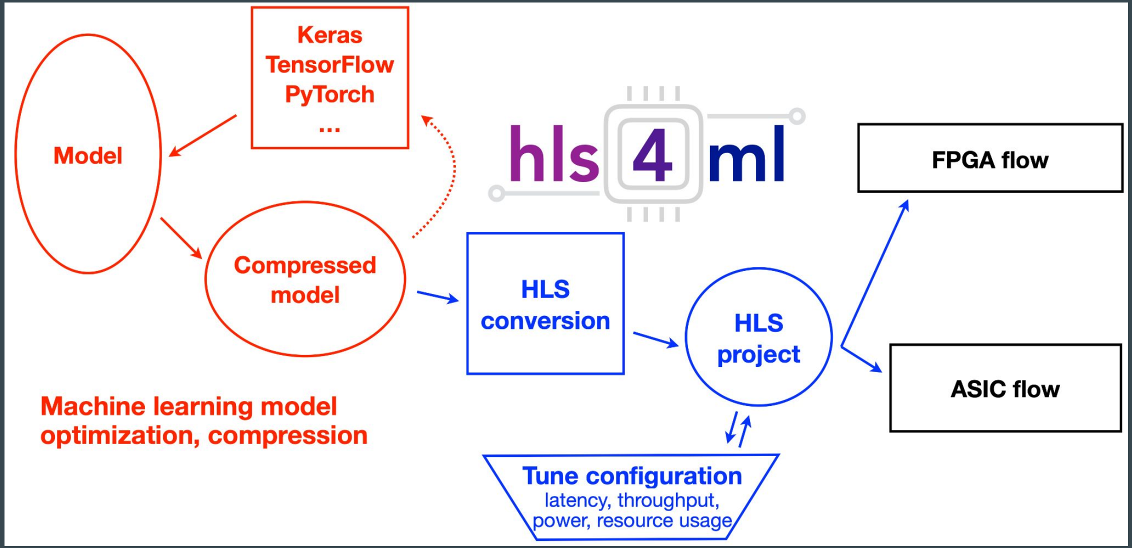






hls4ml

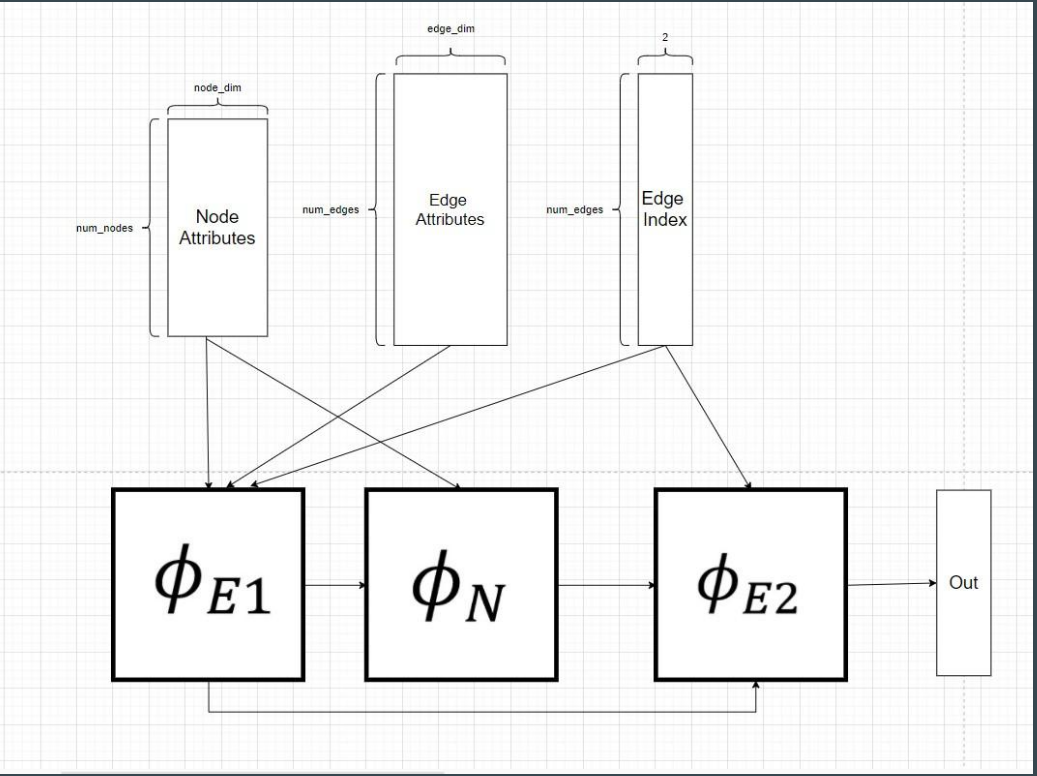





```
layer2_t layer2_out[N_LAYER_2];
#pragma HLS ARRAY_PARTITION variable=layer2_out complete dim=0
nnet::dense<input_t, layer2_t, config2>(input1, layer2_out, w2, b2); // fc1

layer4_t layer4_out[N_LAYER_2];
#pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
nnet::relu<layer2_t, layer4_t, ReLU_config4>(layer2_out, layer4_out); // act1

layer5_t layer5_out[N_LAYER_5];
#pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
nnet::dense<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // fc2

layer7_t layer7_out[N_LAYER_5];
#pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
nnet::relu<layer5_t, layer7_t, ReLU_config7>(layer5_out, layer7_out); // act2

layer8_t layer8_out[N_LAYER_8];
#pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
nnet::dense<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // fc3

layer10_t layer10_out[N_LAYER_8];
#pragma HLS ARRAY_PARTITION variable=layer10_out complete dim=0
nnet::relu<layer8_t, layer10_t, ReLU_config10>(layer8_out, layer10_out); // act3

layer11_t layer11_out[N_LAYER_11];
#pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
nnet::dense<layer10_t, layer11_t, config11>(layer10_out, layer11_out, w11, b11); // fc4

nnet::softmax<layer11_t, result_t, Softmax_config12>(layer11_out, layer12_out); // softmax
```

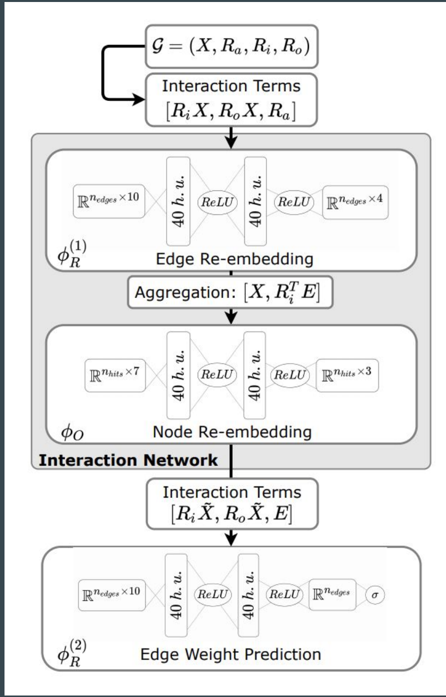
See https://github.com/jmduarte/pytorch_dev_hls4ml/tree/commit_hls

Graph Neural Networks: non-feedforward data flow



← Interaction Network

- https://arxiv.org/abs/1612.00222
- https://arxiv.org/abs/2103.16701
- https://github.com/GageDeZoort/interaction_network_paper



torch_geometric.nn.conv.MessagePassing

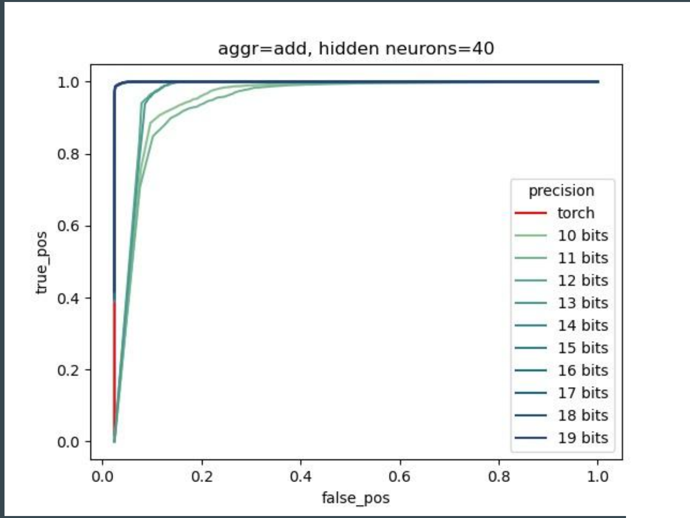
- Message → Aggregate → Update

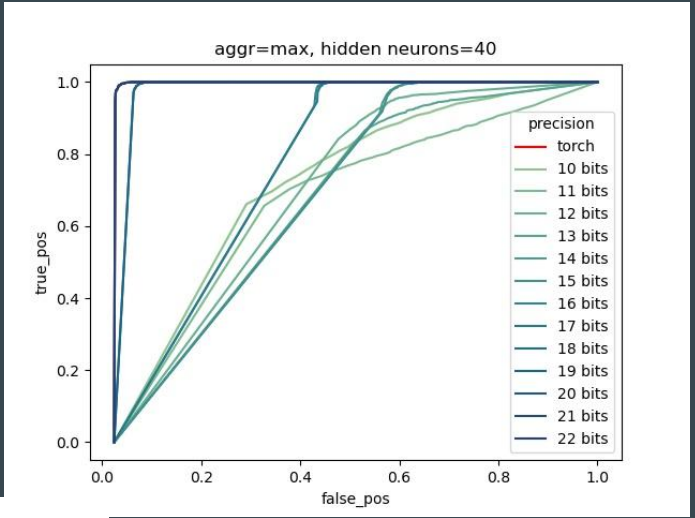
```
def forward(self, data):#x, edge_index, edge_attr):
        x = data.x
        edge_index = data.edge_index
        edge_attr = data.edge_attr

        # Message
        x_i, x_j = x[edge_index[1]], x[edge_index[0]]
        msg_out = self.message(x_i, x_j, edge_attr) # self.message(edge_index, edge_attr)

        # Aggregate
        index = edge_index[1,:]
        ptr = None
        dim_size = x.shape[0]
        aggr_out = self.aggregate(msg_out, index, ptr, dim_size)

        # Update
        update_out = self.update(aggr_out, x)
        x_tilde = update_out

        #return update_out
        m2 = torch.cat([x_tilde[edge_index[1]],
                        x_tilde[edge_index[0]],
                        self.E], dim=1)
        return torch.sigmoid(self.R2(m2))
```

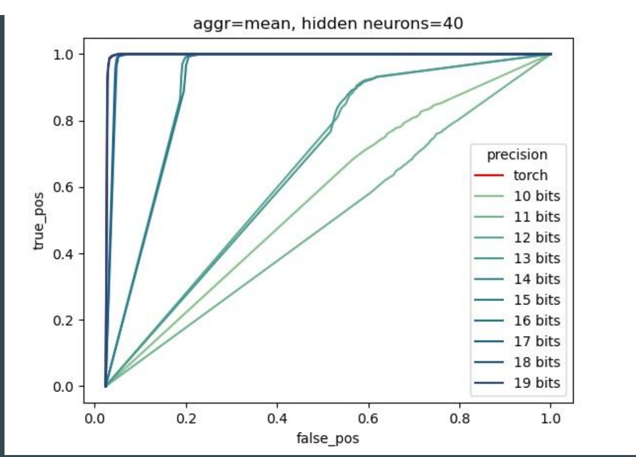
pyg_to_hls: EdgeBlock, NodeBlock, Aggregate

1. class GraphBlock(hls4ml.model.hls_layers.Layer): packages a torch *module* into a form that will be accepted as an HLS *layer*
2. class EdgeBlock(GraphBlock):
    - i. Inputs
        1. Node attributes
        2. Edge attributes
        3. Edge Index
    - ii. Function:
        1. For each edge:
            - a. <edge attributes, receiver-node attributes, sender-node attributes> → Neural Network → Edge predictions
            - b. Edge predictions → permutation-invariant aggregation → Aggregate edge predictions
    - iii. Outputs
        1. Edge predictions
        2. Aggregate edge predictions
2. class NodeBlock(GraphBlock):
    - i. Inputs
        1. Node attributes
        2. Aggregate edge attributes (or aggregate messages)
    - ii. Function:
        1. For each node:
            - a. <node attributes, aggregate attributes> → Neural Network → Node predictions
    - iii. Outputs
        1. Node predictions
3. class Aggregate(hls4ml.model.hls_layers.Layer):
    - a. Edge attributes → permutation-invariant aggregation → Aggregate edge attributes

Interaction Network Benchmarks







User-inputs

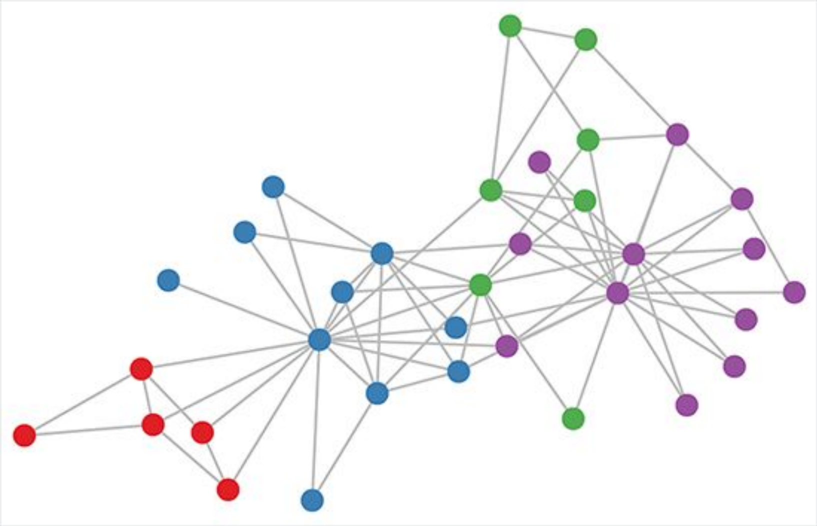
```
def pyg_to_hls(model, forward_dict, graph_dims,
               activate_final = None,
               fixed_precision_bits=16,
               fixed_precision_int_bits=6,
               int_precision_bits=16,
               int_precision_signed=False,
               output_dir = None):
```

```
# model.forward() dictionary
forward_dict = OrderedDict()
forward_dict["R1"] = "EdgeBlock"
forward_dict["O"] = "NodeBlock"
forward_dict["R2"] = "EdgeBlock"
```

Graph padding and truncation

Hardware-implementation requires: n_nodes_max, n_edges_max

1. Truncation: removes true nodes, true edges, or both
    a. Always bad
        i. Fixes: Look for disconnected nodes, remove the least connected nodes first
            1. Compute
2. Padding: adds dummy nodes, dummy edges, or both
    a. sometimes bad
        i. n_nodes >= n_nodes_max, n_edges < n_edges_max
            1. Dummy edges must connect true nodes
    b. sometimes alright
        i. n_nodes<n_nodes_max, n_edges <= n_edges_max
            1. Dummy nodes disconnected, or dummy nodes connected with dummy edges



Support yet to come

- Non-linear layers
- Arbitrary # of layers per block
  - Currently: 1-->4 layers
- Higher-degree neighbors for “Message” and “Aggregation”
  - Currently only first-degree neighbors
- Different “Message” schemes
  - Currently: NN_input = concat(receiver attributes, sender attributes, edge attributes)

About the author

← Abdelrahman Elabd

- Contact: aelabd@sas.upenn.edu, abdelabd777@gmail.com
- B.A.: Physics, Economics @ U.Penn.
  - Graduation: August 2021
- Research Interests:
  - HEP, Astronomy, Applied ML for experimental Physics
  - Information theory, entropy and thermodynamics
  - Signal processing and hardware design
  - Graph Neural Networks and Graphical Causality
  - Reinforcement learning, Game Theoretic ML
- Hobbies:
  - Mountain biking, hiking, fishing
  - Basketball
  - Electric Guitar (noob)
- Applying to Physics and CS PhD programs!