Track Reconstruction in the Upgraded LHCb VELO

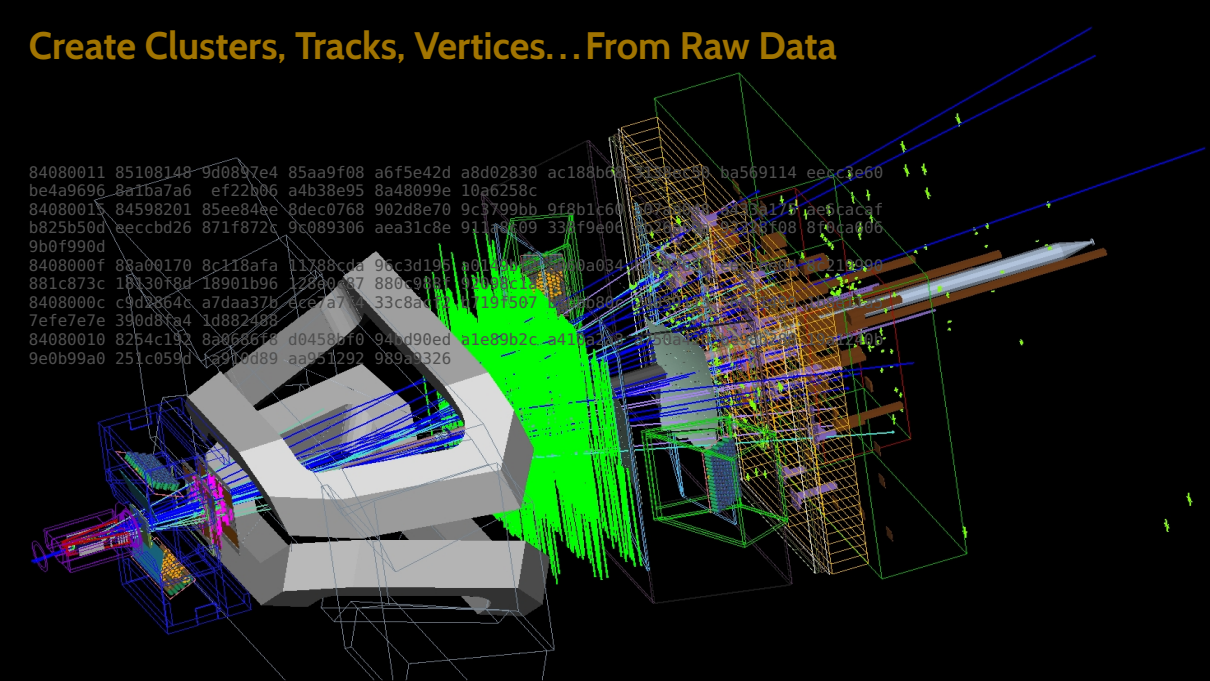
Novel Approach: An NN Classifier Connecting Detector Layers

Kurt Rinnert

UNIVERSITY OF LIVERPOOL

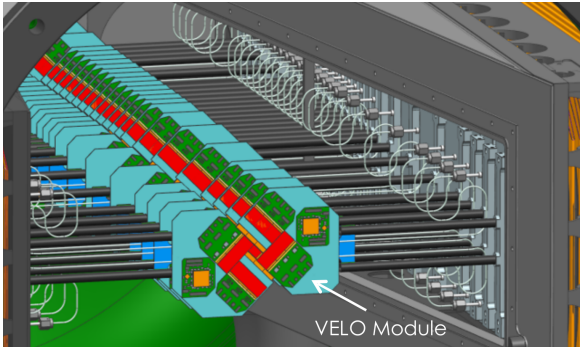
EDM Workshop
Liverpool/FNAL
04.10.2018

Create Clusters, Tracks, Vertices…From Raw Data



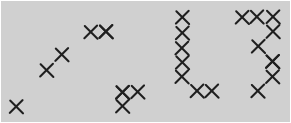
The VELO Upgrade

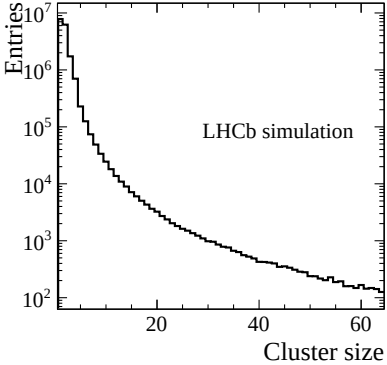
- Two movable halfs
- 52 modules with 4 pixel sensors each
- 768×256 pixels per sensor
- pixel size $55\mu m$
- closest pixel $\sim$ 5mm from beam



Higher luminosity and more interactions.

Pixel Clustering



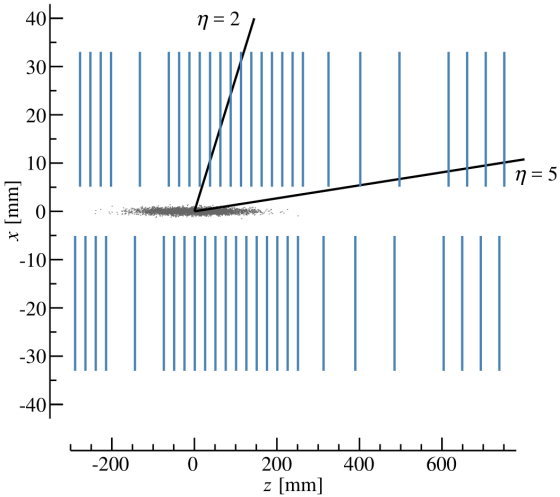


- Choose data format wisely
- Takes about 10% of the CPU time
- Hard to do on FPGA
- Cutting the tails not trivial

Do not underestimate resources for clustering.

Conventional Approach

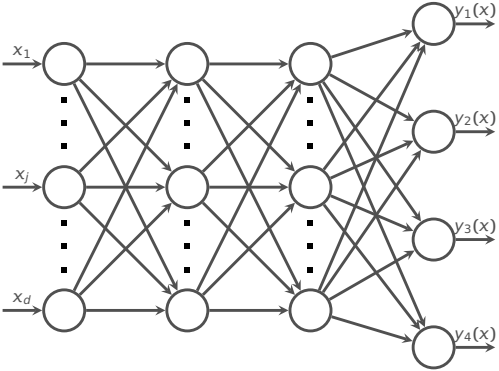
- Build track stubs from neighboring layers
- Require $|dr/dz| < 0.4$
- Pursue stub through layers
- Many configurable parameters (search windows etc.)
- Very fast, ~2ms/event



Good performace. Hard to define working point.

FCNN Classifier

- Inputs **x** are seed & target layer (r, ϕ) coodinates
- *One* seed (r, ϕ) pair and *several* target (r, ϕ) pairs
- Outputs are target index (class) probabilities
- Network topology & dimension of **y** not *a priory* ovious
- Trained on labeled data from *full* simulation
- Train one classifier for each pair of layers to be connected



Efficient data preparation an book-keeping required.

Dataset & Format

- Minimum bias data with $\nu = 7.6$
- One (r, ϕ, z) space point per cluster
- The z coordinate is not used (implicit)
- One file stream per module
- Vectors of interlaced r and ϕ values
- Padded to modulo **y** dimension
- Vector of truth labels (MC IDs)

Inputs: d seeds and n targets

$$
\begin{array}{ccccccccc}
\mathbf{r^s_0} & \phi^s_0 & r^t_0 & \phi^t_0 & r^t_1 & \phi^t_1 & \dots & r^t_3 & \phi^t_3 \\
\mathbf{r^s_0} & \phi^s_0 & r^t_4 & \phi^t_4 & r^t_5 & \phi^t_5 & \dots & r^t_7 & \phi^t_7 \\
\dots & & & & & & & & \\
\mathbf{r^s_0} & \phi^s_0 & r^t_{n-1} & \phi^t_{n-1} & r_p & \phi_p & \dots & r_p & \phi_p \\
\dots & & & & & & & & \\
\mathbf{r^s_{d-1}} & \phi^s_{d-1} & r^t_{n-1} & \phi^t_{n-1} & r_p & \phi_p & \dots & r_p & \phi_p
\end{array}
$$

Choose padding values wisely.

Implementation

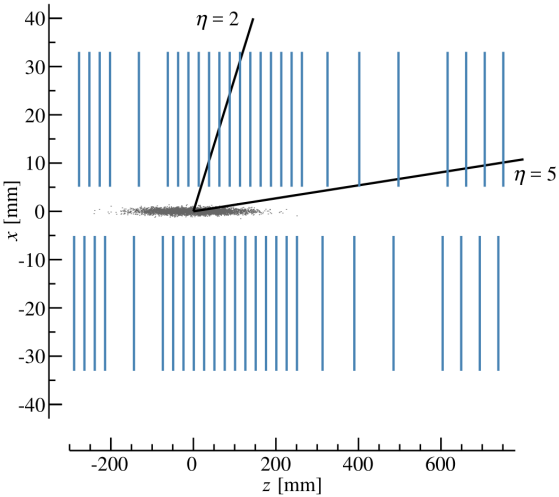
- Model building is done in PyTorch

```
class RPhiSeedClassifier(nn.Module):
    def __init__(self, npoints, nhidden):
        super().__init__()

        self.npoints = npoints  # 4
        self.nhidden = nhidden  # 129

        self.linear_1 = nn.Linear(2*(self.npoints + 1), self.nhidden)
        self.linear_2 = nn.Linear(self.nhidden, self.nhidden + 1)
        self.linear_3 = nn.Linear(self.nhidden+1, (self.nhidden + 1) // 2)
        self.linear_4 = nn.Linear((self.nhidden + 1) // 2, self.npoints)

    def forward(self, x):
        x = F.relu(self.linear_1(x))
        x = F.relu(self.linear_2(x))
        x = F.relu(self.linear_3(x))
        x = self.linear_4(x)
        return x
```

Other frameworks are more opaque and harder to control.

Track Reconstruction with FCNN Classifier

- For now only per-half reconstruction
- Start from most downstream module (module 50 for left half; upper in diagram)
- Connect to next upstream layer with classifier
- Allow to skip one layer
- Repeat until no connection is found
- List of cluster indicies → track



No additional assumptions!

Performance

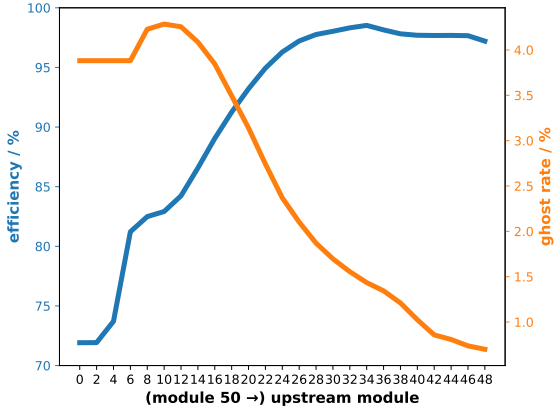
- Reconstructible: $\geq$ 3 hits
- Reconstructed: $\geq$ 70% of hits from one particle
- Clone: reconstructed twice or more
- Ghost: no association possible ($<$ 50% from same particle)

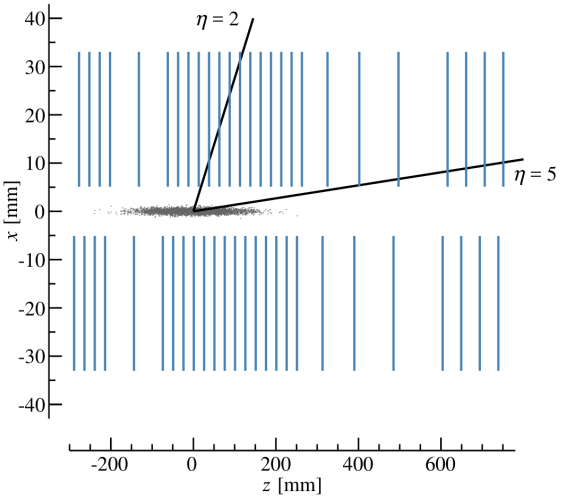
Performance comparison

algorithm	eff. %	ghost %	clone %
conventional, p > 5GeV	98.9	2.5	1.0
FCNN, module 50 $\rightarrow$ 30	98.0	1.6	0.8
FCNN, module 50 $\rightarrow$ 20	93.2	3.1	0.9
FCNN, module 50 $\rightarrow$ 0	71.1	3.9	0.7

Passing through the interaction region is a problem for the NN.

Understanding the FCNN Performance





Several options for improvement. (Use slopes, more capacity around IR…)

Collaboration With Industry

Microsoft Azure

- Azure cloud resources for model development & training
- Discussions with MS AI engineers
- Learn about available frameworks
- Understand deployment options

We work to broaden this initiative.

Summary

Achievements & things to come

- A simple layer-connecting NN classifier performs well (within reason)
- Numerous improvements just around the corner
- More sophisticated ideas will be tried (reinforcement learning, graphs…)
- Framework integration is crucial to understand all performance aspects
- Industry connection very helpful