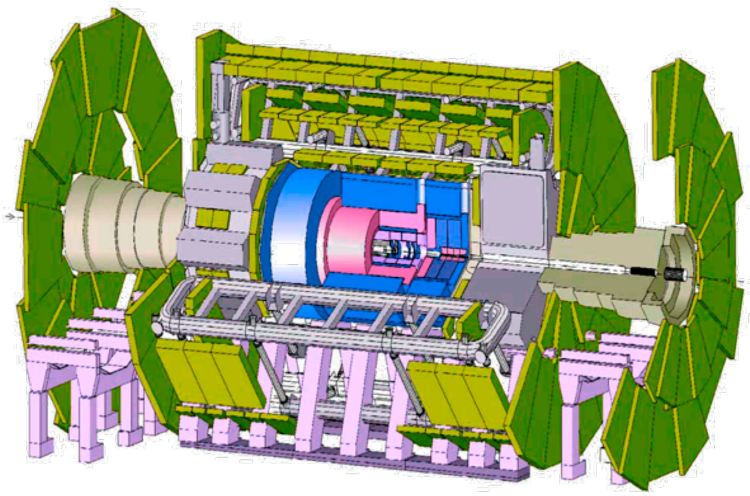
ML in Tracking

Alex Gekow[1], Antonio Boveia[1], JC Zeng[2], Viviana Cavaliere[3], Will Kalderon[3], Haider Abidi[3], Hao Xu, Elena Zhivun, Filiberto Bonini, Shinjae Yoo,
April 16, 2021
1 - Ohio State University
2 - UIUC
3 - BNL

Most of the slides made by Haider

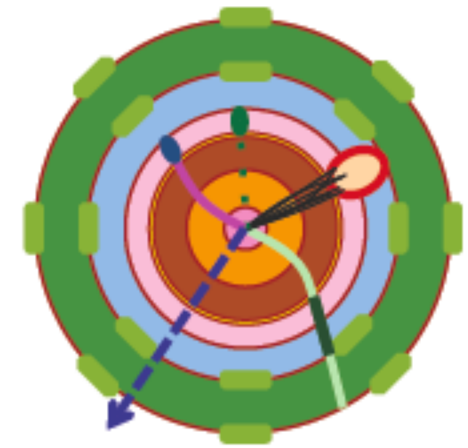
From the detector to the physics results

Detector



Trigger

Reconstruction

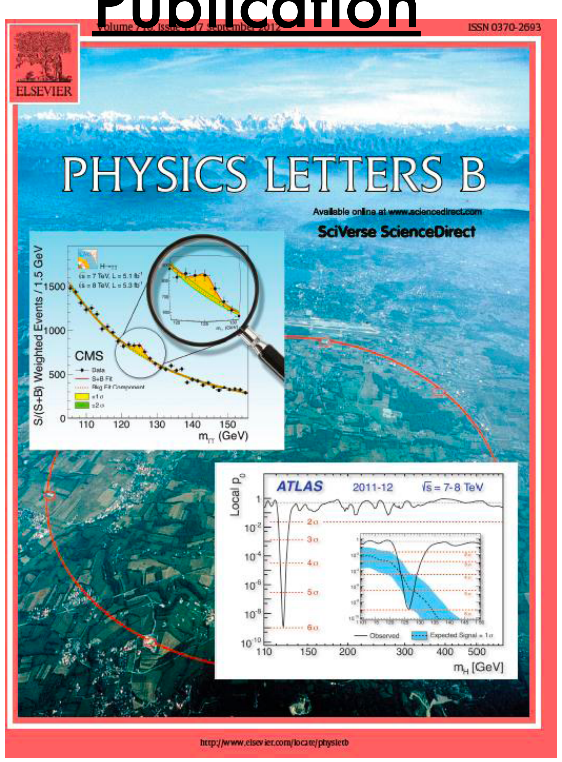


rate: 40 MHz
~75 Tb/s

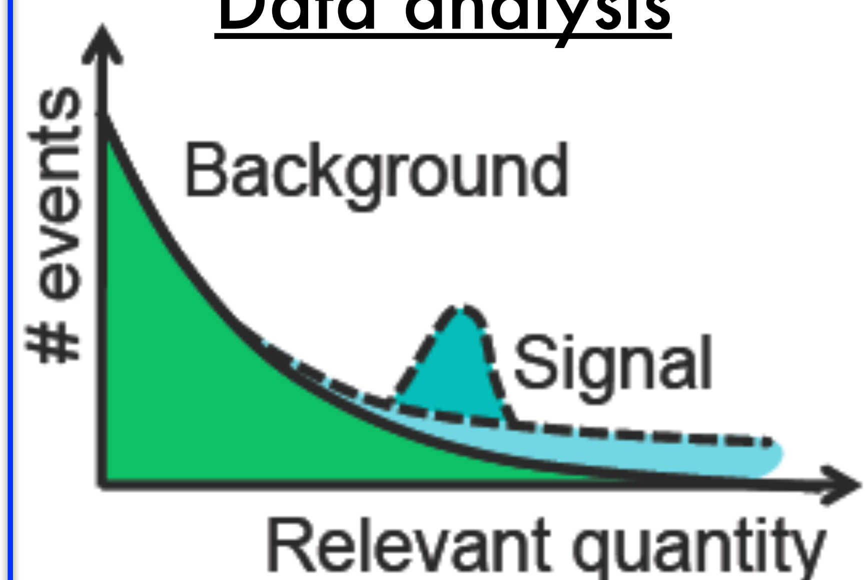
1KHz
Event size:1.6Mb

1Hz

Publication



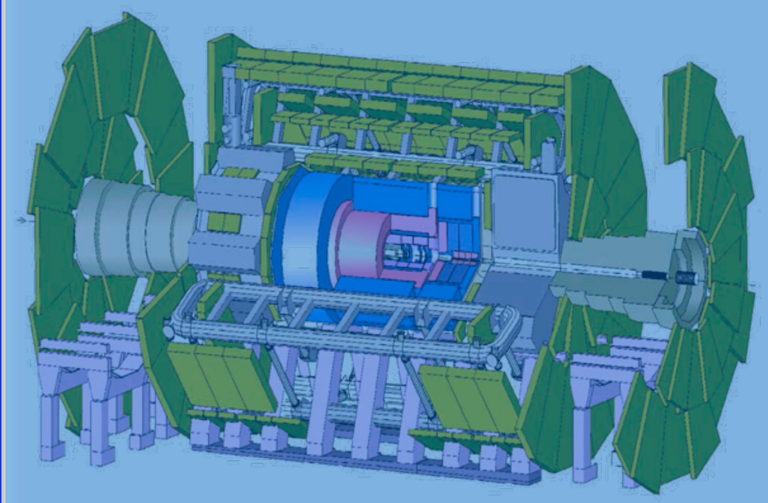
Data analysis



1 observed Higgs event in a trillion (10^{12}) pp collisions

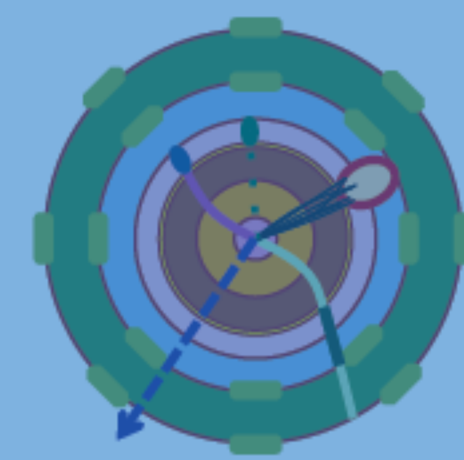
From the detector to the physics results

Detector



Trigger

Reconstruction

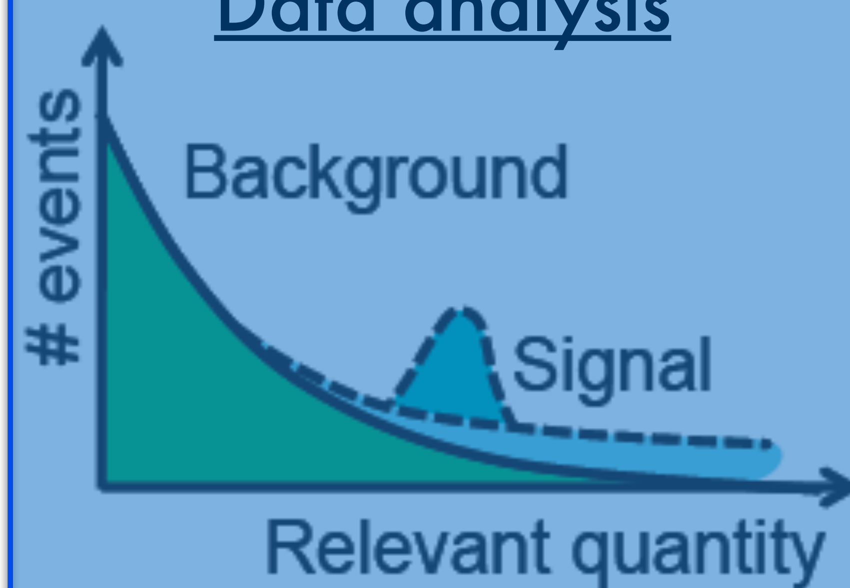


rate: 40 MHz
~75 Tb/s

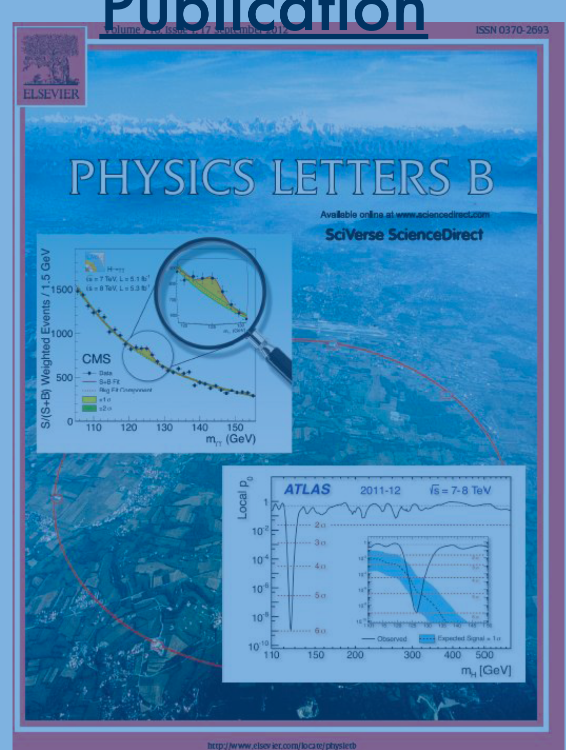
1KHz
Event size:1.6Mb

1Hz

Data analysis



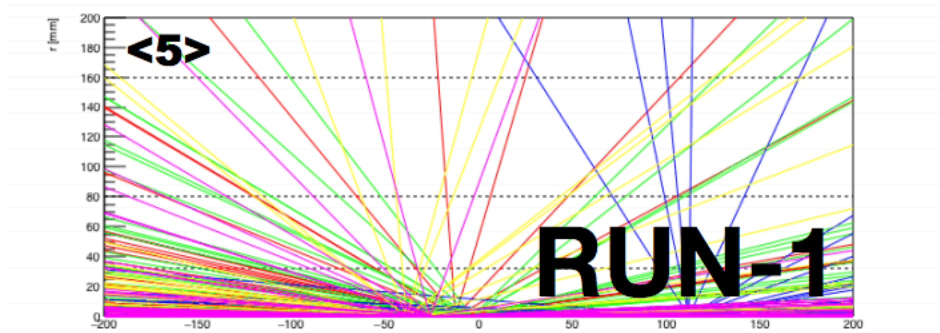
Publication

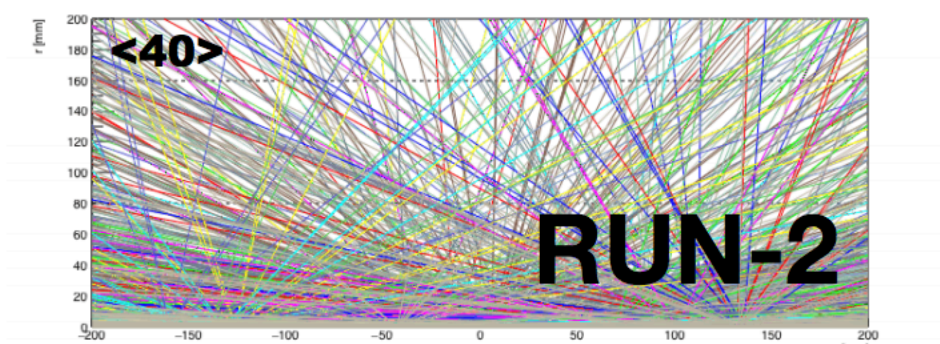


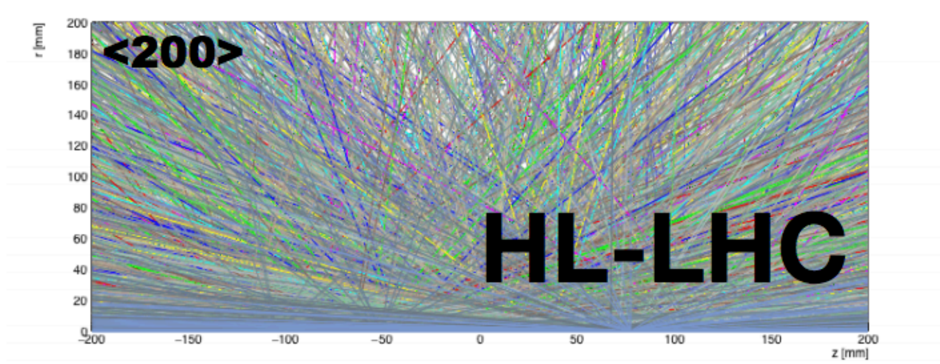
1 observed Higgs event in a trillion (10^{12}) pp collisions

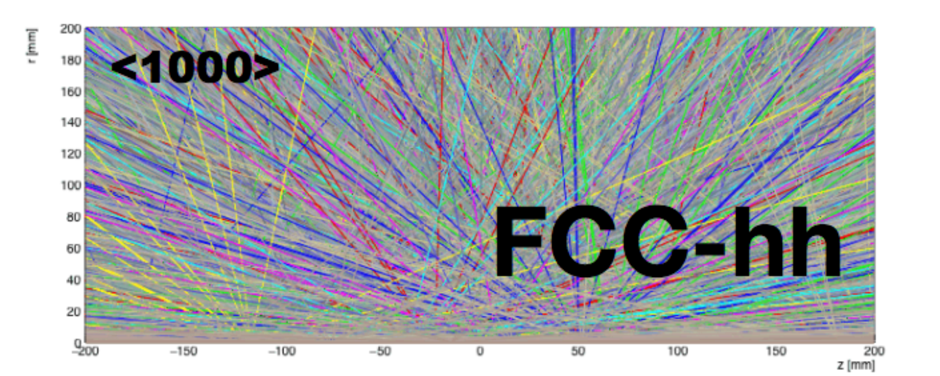
Environment in sPhenix, LHC, HL-LHC and FCC-hh

mu= # of simultaneous pp collision









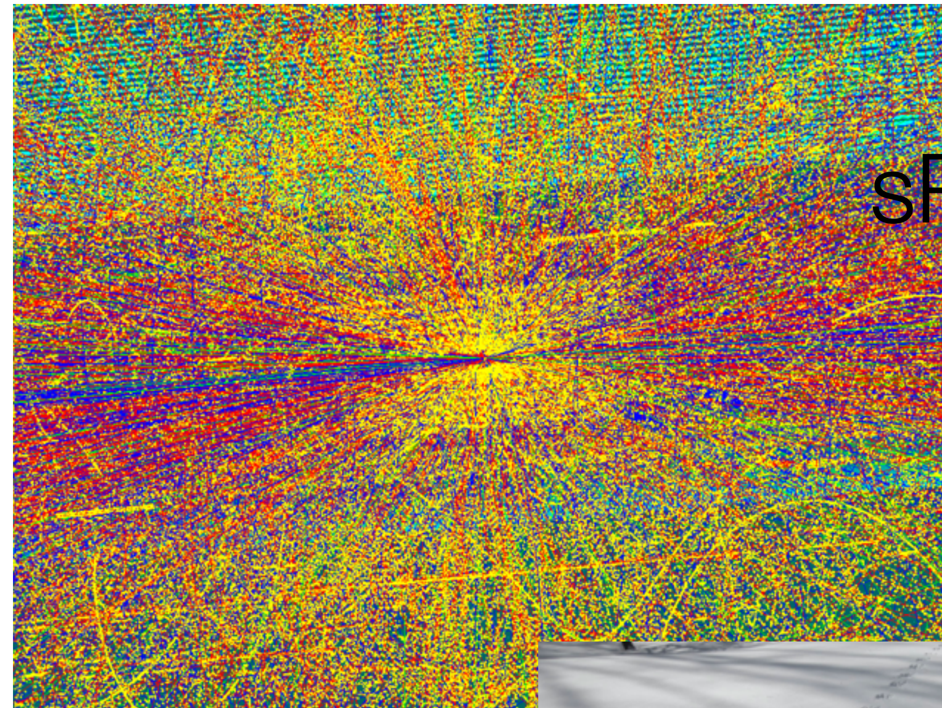


Fig. 1

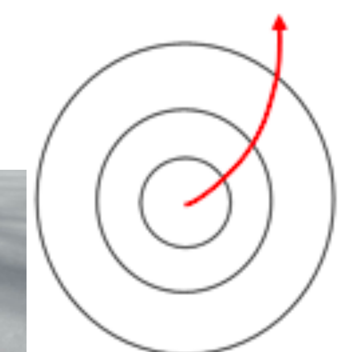


Fig. 2

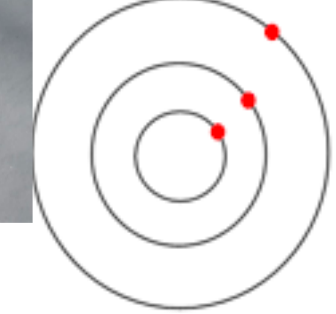


Fig. 3

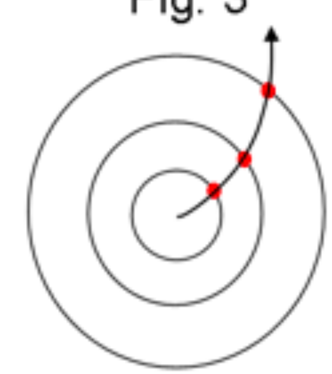


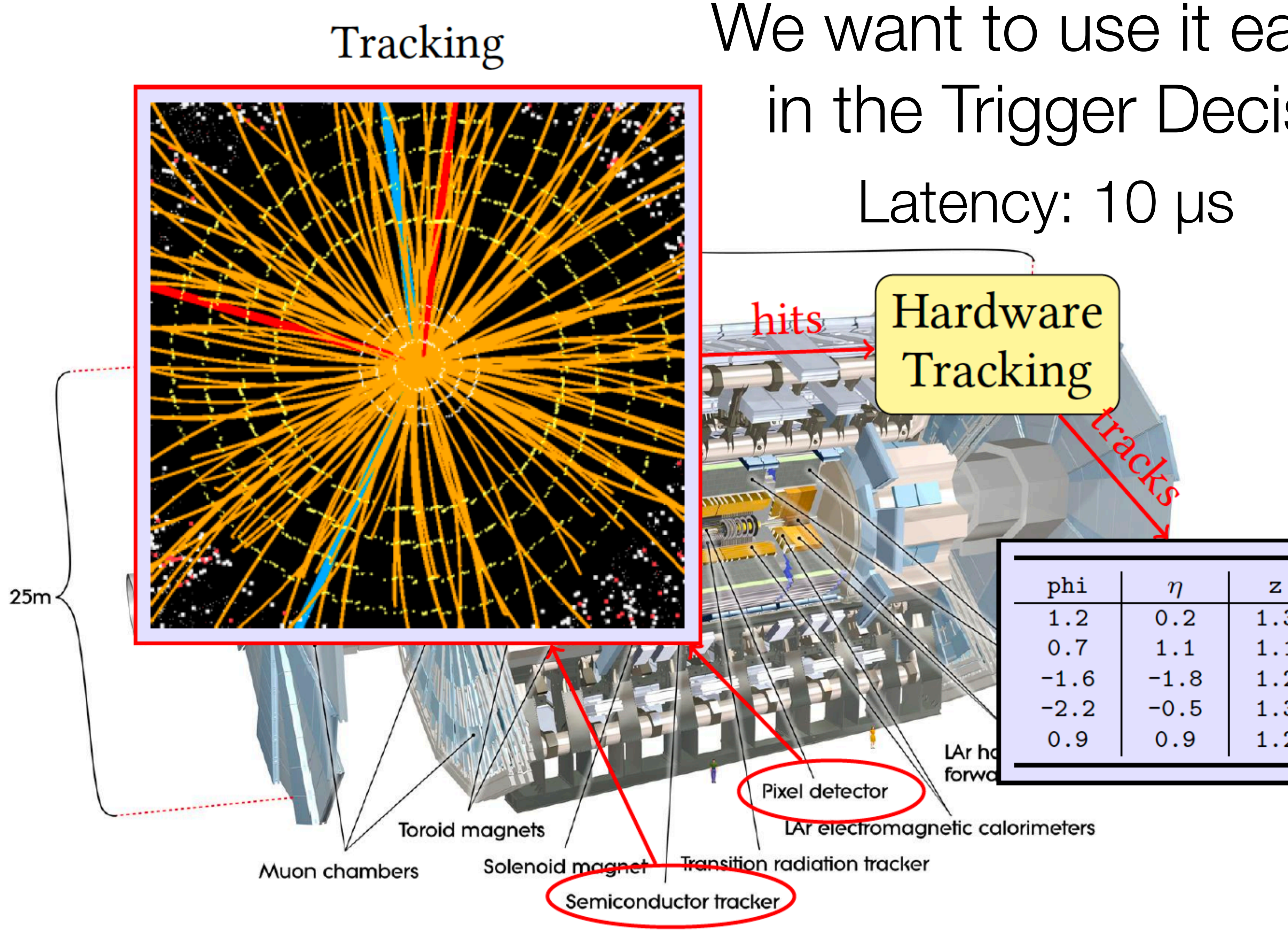
Fig. 4

- Overcome the pileup problem by
  - tracing all the paths left by the particles back to the center of the detector, pinpointing all the collisions points (called vertices) that occurred at a proton-bunch crossing
  - decide which particles originated from which vertex
- **==> Need Tracking for early background rejection @ Trigger level**

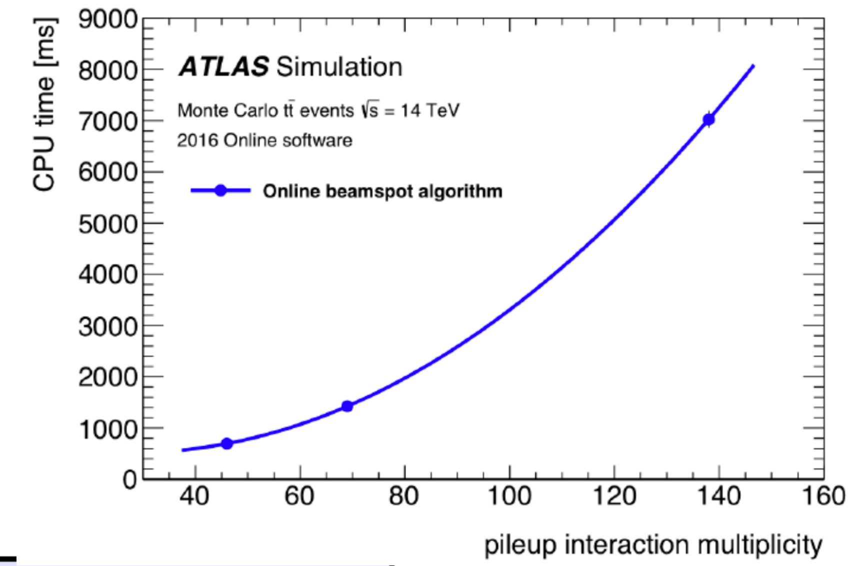
Real Time Tracking

We want to use it early on in the Trigger Decision

Latency: 10 μs



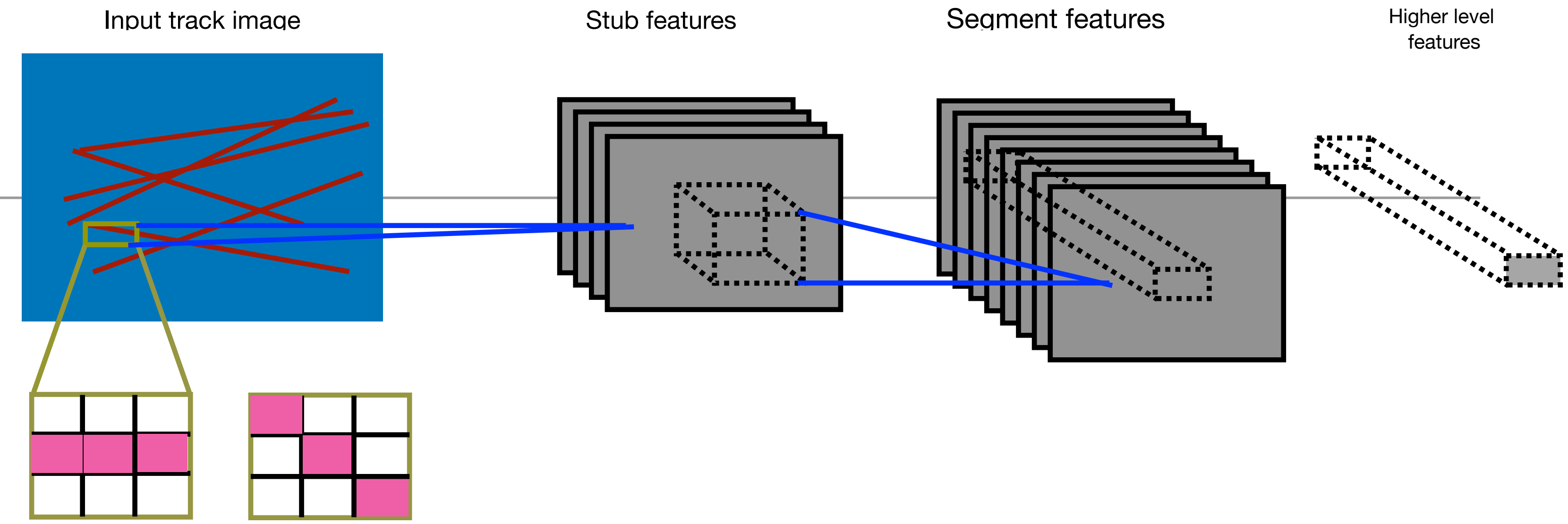
phi	η	z	p_T	d
1.2	0.2	1.3	10.5	0.03
0.7	1.1	1.1	4.3	0.14
-1.6	-1.8	1.2	6.6	0.32
-2.2	-0.5	1.3	12.2	0.28
0.9	0.9	1.2	1.9	0.83



Software tracking —> order of milliseconds to perform tracking
Move to hardware —> FPGAs much faster

Huge combinatorial problem, need optimized algorithms

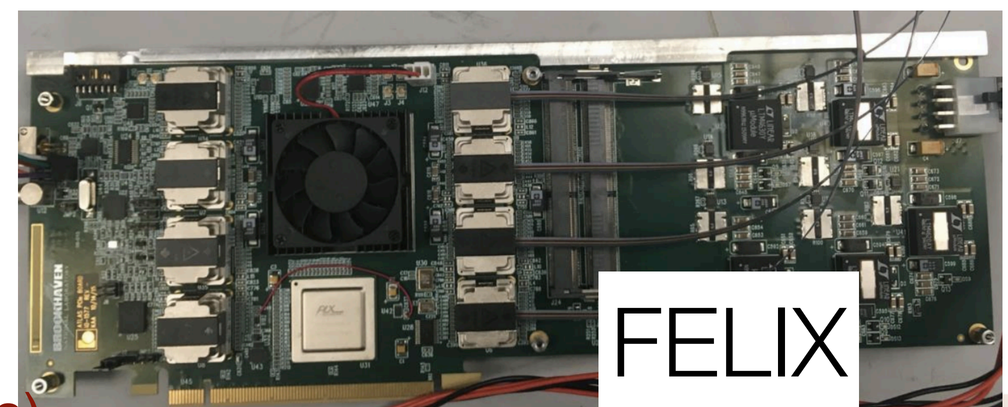
How?



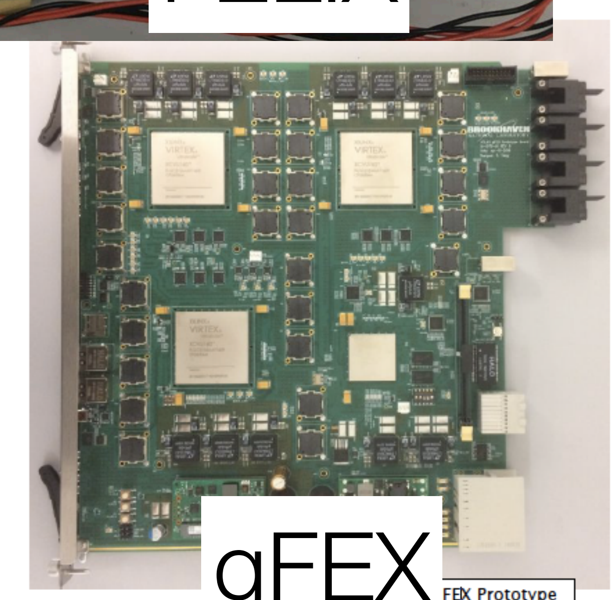
To meet the future experimental needs in term of latency need optimized algorithms ==> **Machine learning**

- Convolutional NN particular suitable, why?
  - Tracking data exhibits some of the traits of natural images: translational symmetries, locality etc
  - The initial layer of a CNN for tracking would identify stubs of compatible hits in adjacent layers.
  - Later layers of the network would then connect stub features together to form track segments, and so o model of an entire track or set of tracks is constructed.

- Problem: Deeper networks greatly increase the number of parameters and model sizes ==> increase the computational, memory bandwidth, and storage demands.
- Solution: Adopt more *compact low precision data types*



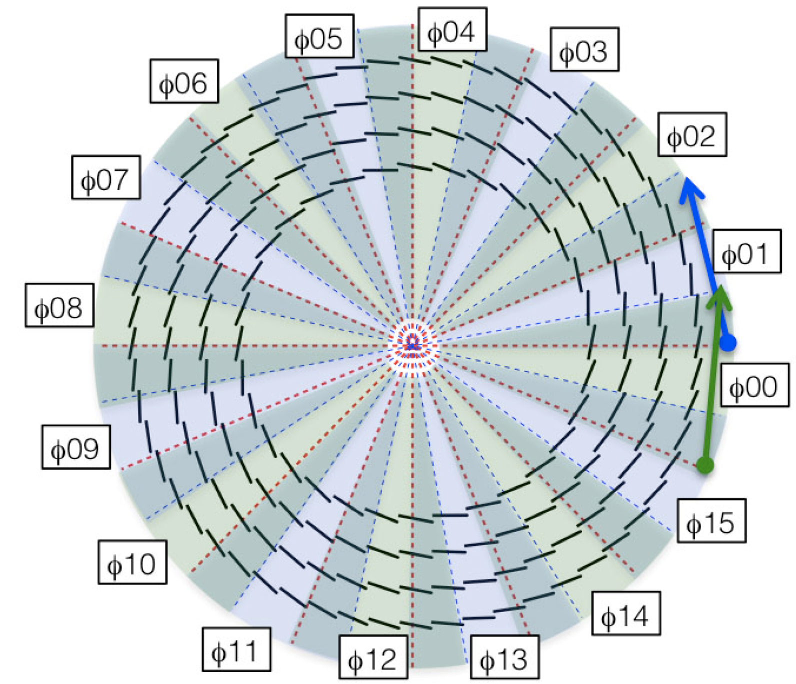
- Problem: GPU still too slow for the required latency O(ms)
- Solution: Use FPGAs: have distributed on-chip memory as well as large degrees of pipeline parallelism, which fit naturally with the feed-forward nature of deep learning inference methods
- **Test the system using HL-LHC pseudo-data and two boards developed at BNL**:
  - FELIX, data routing system , gFEX with four FPGAs with high speed interconnects

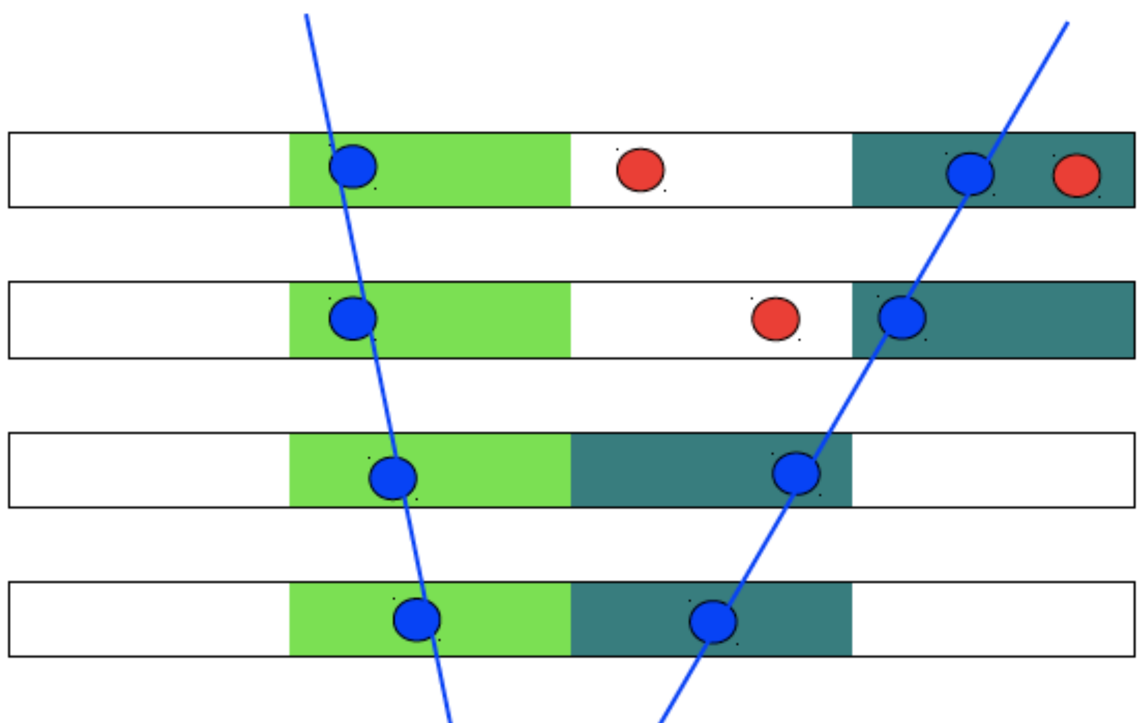


Track Trigger Challenges

Tracking has to be very fast

- Process in parallel: decompose detector data into independent regions (64 Towers)
- Data reduction: each cluster of adjacent pixels/strips defines one “hit”
- Perform tracking in two steps:
  ★ Find track candidates: Roads
  ★ Perform full-resolution track fitting inside Roads
    ★Combinatorics reduced
- Each of the pieces of the full chain can be performed using Machine Learning



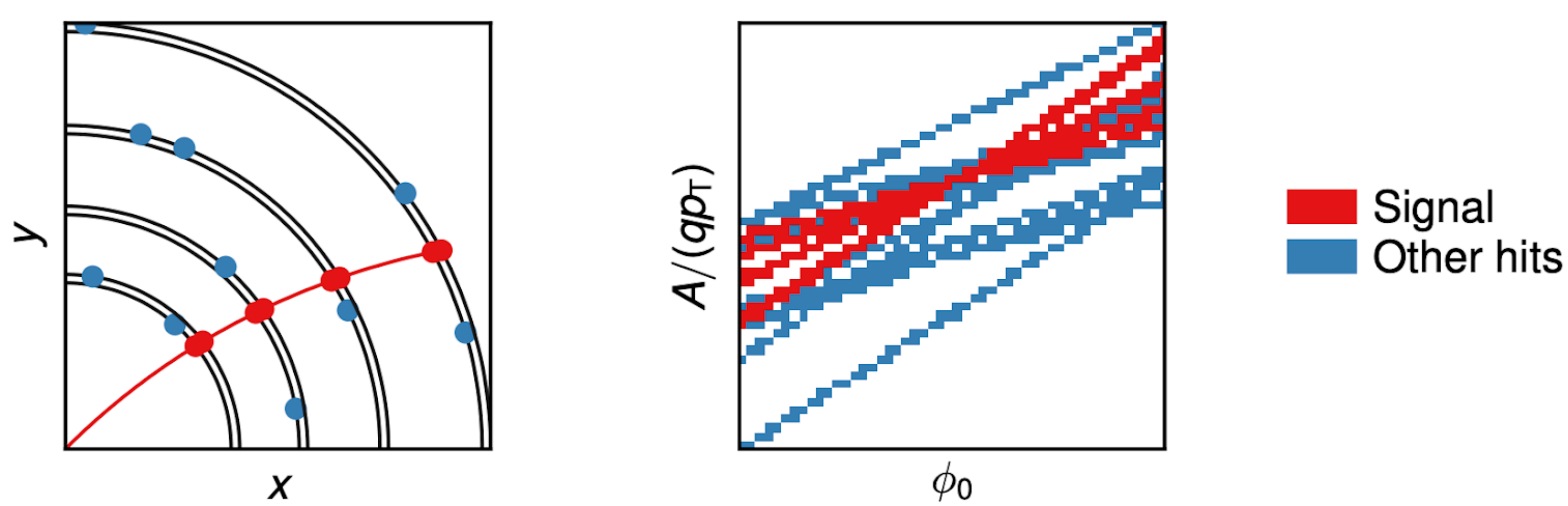


Hough Transform for Track finding

- The track of a charged particle in the transverse plane (x-y plane) of the ATLAS tracker has the shape of a circular arc which can be described by pT and its initial angular direction φ_0.
- If a vertex constraint is imposed, the clusters on track obey:

$$\frac{qA}{p_{\mathrm{T}}}(\phi_0) = \frac{\sin(\phi_0 - \phi_1)}{r_1}$$

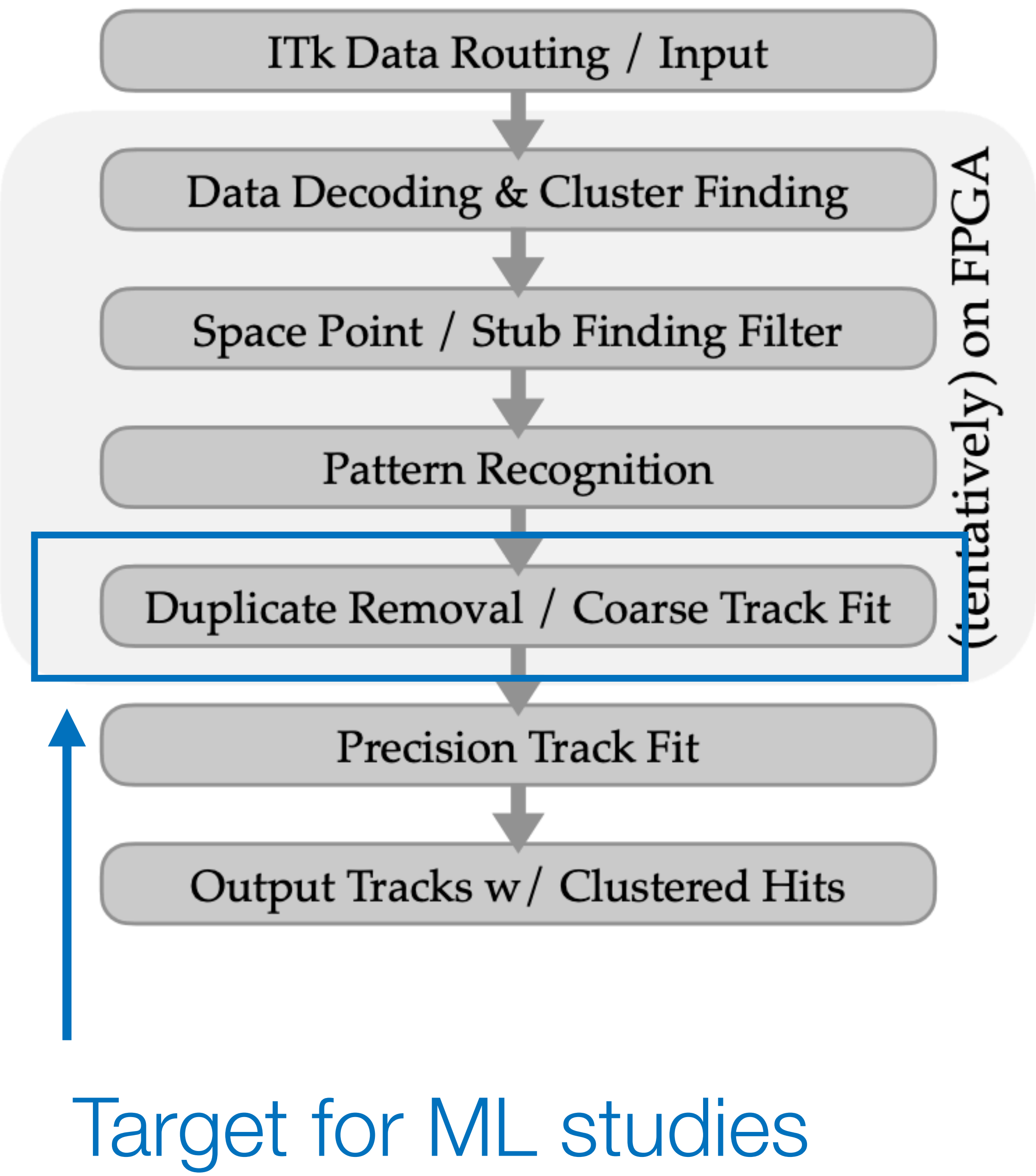
- where (r_1,ϕ_1) are the cluster coordinates, q is the charge of the particle, and $A \approx 3\times10^{-4}$ GeV mm^{-1} is the curvature constant for the 2T magnetic field of the tracker.
- Initialize a histogram (accumulator) with the parameter space to search.
    - Group hits into super strips in ϕ.
- For each point, increment the histogram for all possible curves going through that point.
- Points on the same curve will intersect in the parameter space
- Threshold accumulator at a certain value.
- Extract the hits for all bins passing the threshold.



Machine Learning for tracking

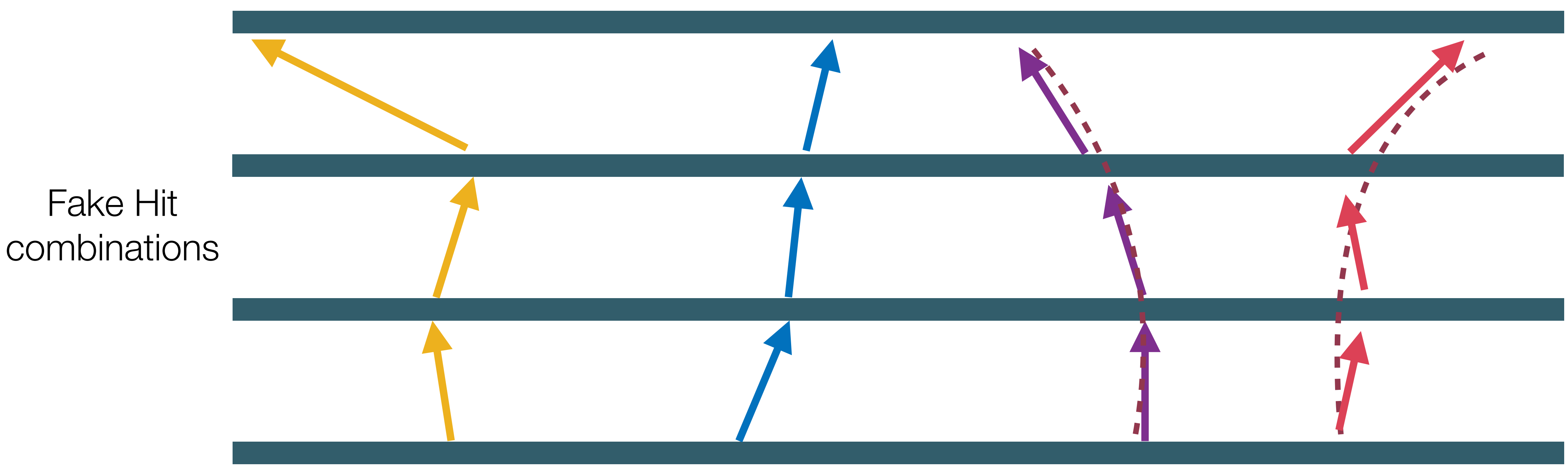
- In general, we have been studying using ML for tracking on FPGAs
  - Approaching from the other end - minimal complexity that can fulfill the requirements while fitting on a FPGA
- For Heterogeneous Commodity TF, main focus is on a ML module to do duplicate removal/ fake rejection
- While the consideration are to fit this on a FPGA, ensuring that the NNs can be used on CPUs/GPUs without hassle
  - Already using ONNX interface to do performance studies in an Athena environment

System block diagram for the system being considered in the Heterogeneous Commodity TF



Problem setup

- Initial studies performed **without** assuming what algorithm the track candidate will come from
  - To keep the conclusions as generic as possible
- **Problem:** Classify a vector of x/y/z position coordinates as coming from a 'track'
  - True combination: Hit combinations from offline track reconstruction
  - Fake combinations: Some dependancy on the exact definition
- Only strip layers were used, with hits both in the Barrel and End Cap
  - But easily configurable



Layer
Combine hits that pass through successive layers

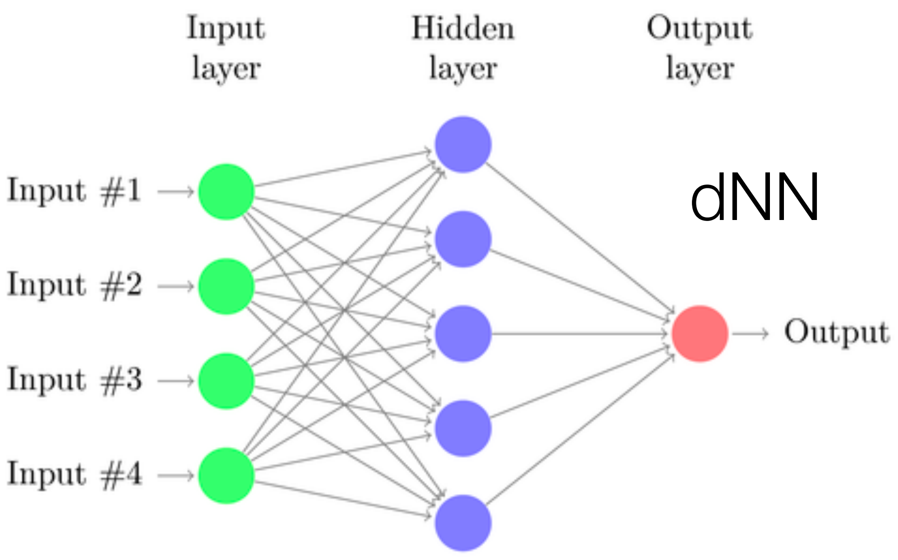
Module
Combine random hits in modules that a true track would traverse

Close by
Replacing random hits with other nearby ones around a true track

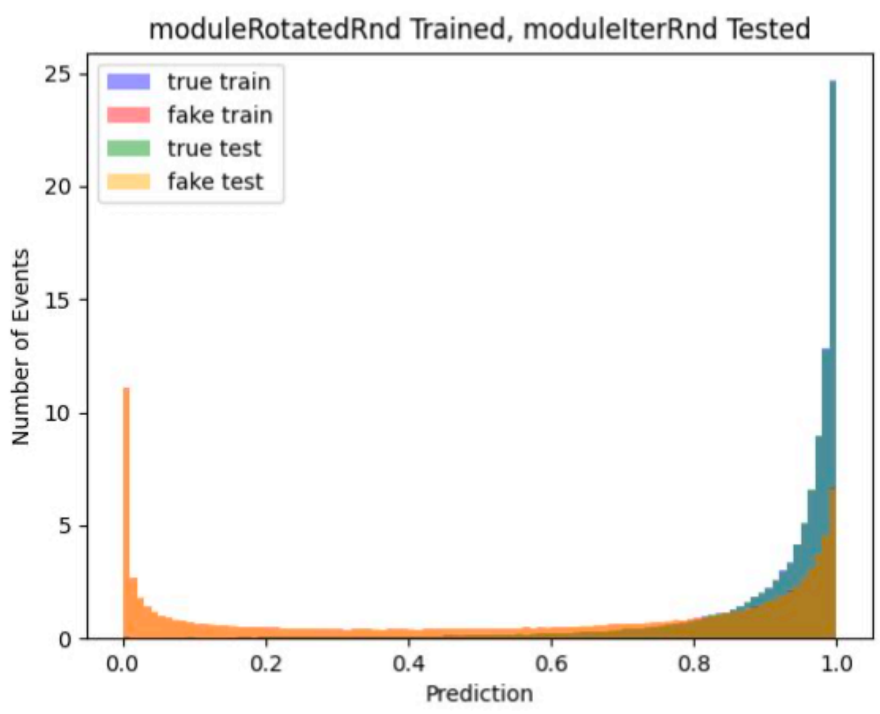
Phi rotated
Find a hits modules that a random phi rotated track would traverse

Initial Results

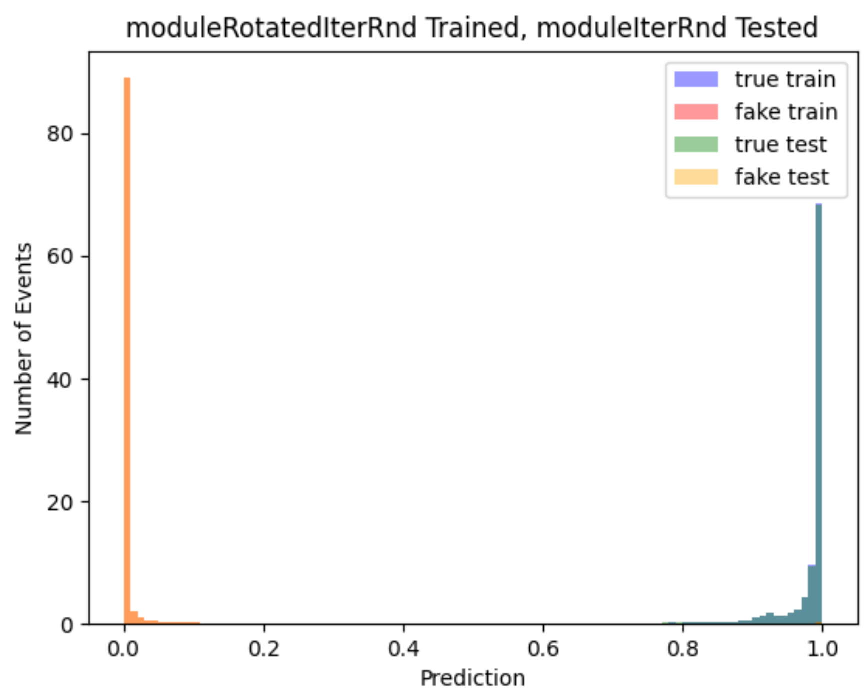
- Very promising results - allowed us to fine tune the recipe
- Architecture - simple dNN performs well, though cNNs perform just as well with less training params
- Pre-processing - Some is required
  - Rotate hits to remove the phi DOF
  - Scale X/Y/Z coordinate such that max value is O(1)
  - Order hits by R



dNN



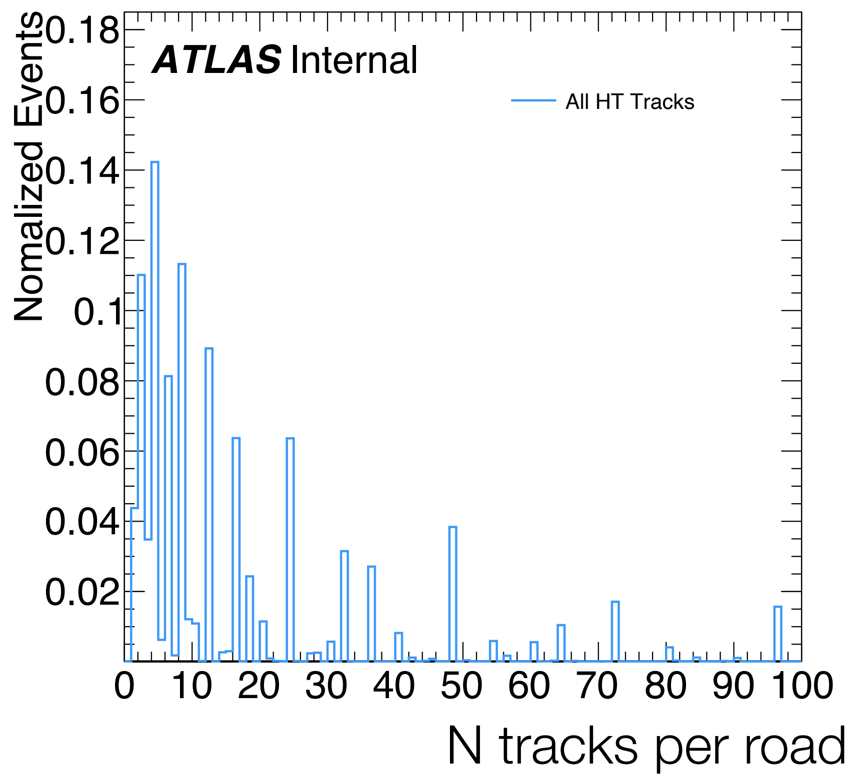
Removing the rotational degree of freedom

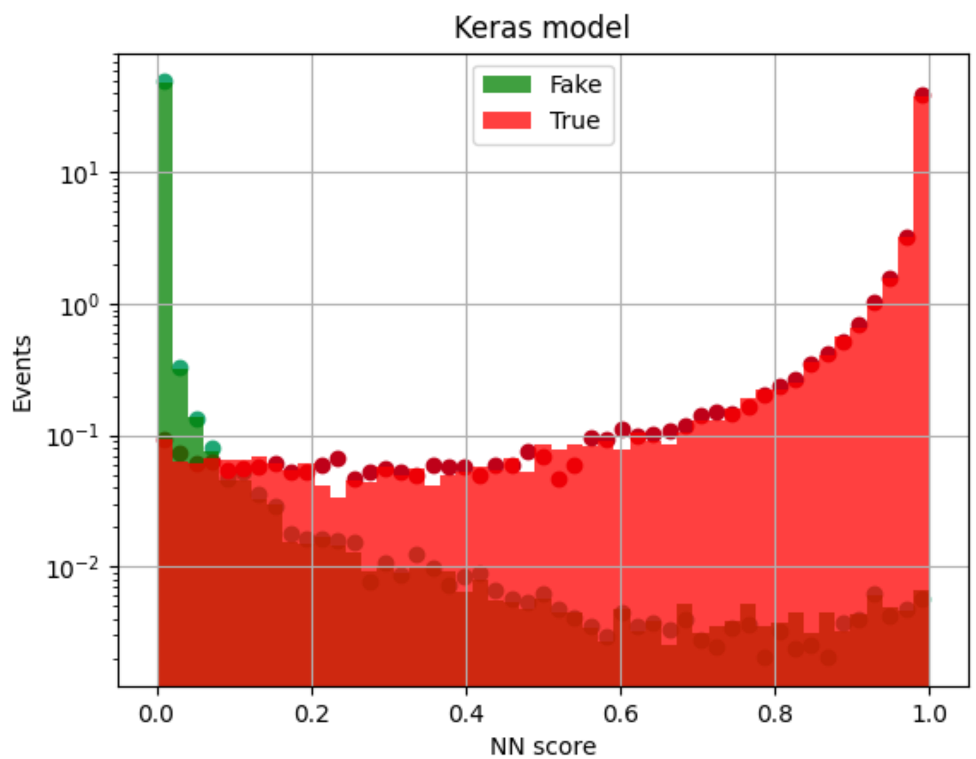


Overlap/Fake removal after Hough Transform

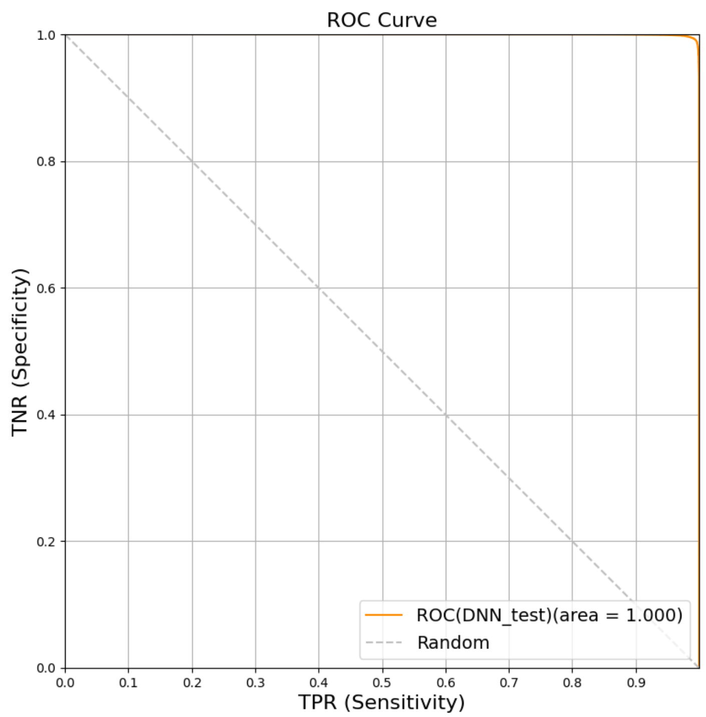
- In the TF system, the Hough Transform step will provide the hit combinations
- NN has two tasks:
  - Pick the best track candidate in a given road
  - Reject fake tracks to improve purity
- Train the network in the 1 pixel + 7 strip layer configuration
  - Fake tracks: HT tracks from 1 single particle + pileup event with truth probability < 0.7
  - True tracks: Offline + truth tracks from single particle + pileup events

Can go up to 100s tracks per road





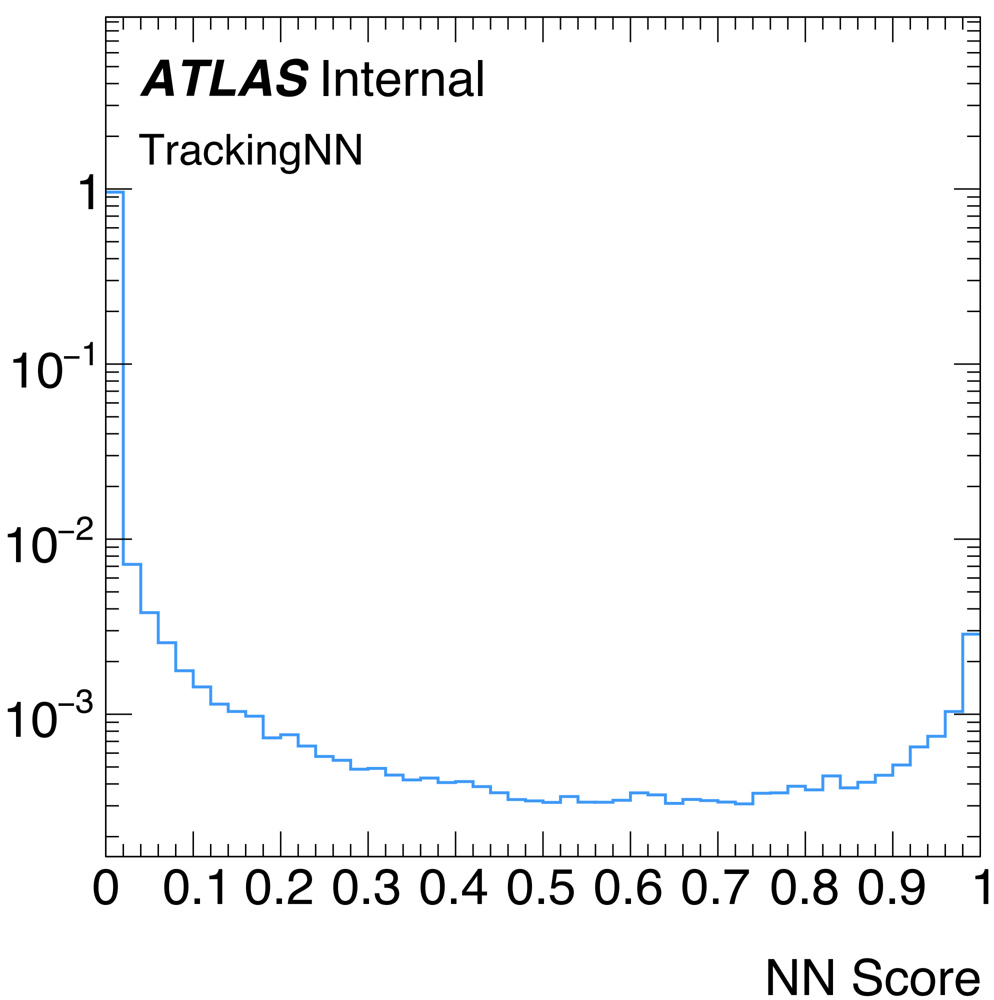
No overtraining observed



ROC integral is ~1

Initial results

- Using the current working setup for the Hough transformation
  - mu=200, with $\eta \in [0.1,0.3]$
- **Reject** any tracks with NN score < 0.2 & pick the highest NN scored track in a road
  - ~ 100x reduction in the number of tracks.



Evaluated NN on HT output

Preliminary

Reco	Total	Selected	Selected && not matched
	3,590,000	41734	13472
Purity	0.08%	6.6%	-
Selection Efficiency	-	1.2%	

With respect to truth	Total	Matched to atleast one HT track	Selected && matched
	3103	2910	2769
Absolute Eff	-	93.8%	89.2%
Eff of NN cut			95.2%

Caveat: These are absolute efficiency wrt the truth, updated results will be wrt offline tracks as per the TF mandate

NN cut threshold

- Choice of NN > 0.20 is arbitrary
    - Depending on how much we can afford, can increase the selection efficiency of the NN

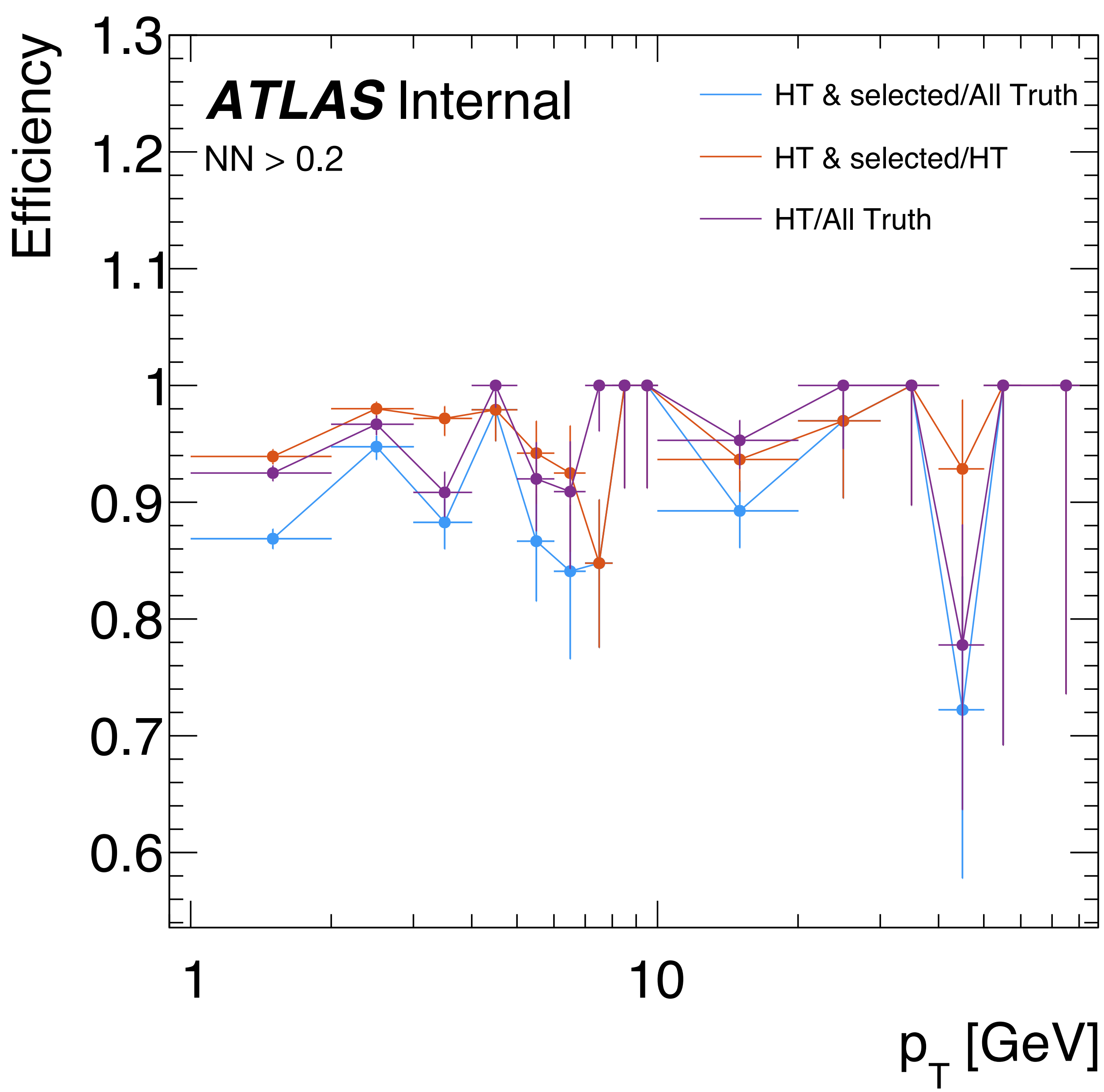
Preliminary

Reco	Total	NN > 0.2	NN > 0.1	NN > 0.05	NN > 0.01	NN > 0.001	NN > 0
	3,590,000	41734	47937	54073	67938	89813	229324
Purity (%)	0.08%	6.6%	5.8%	5.2%	4.2%	3.2%	1.3%
Selection Efficiency (%)	-	1.2%	1.3%	1.5%	1.9%	2.5%	6.4%

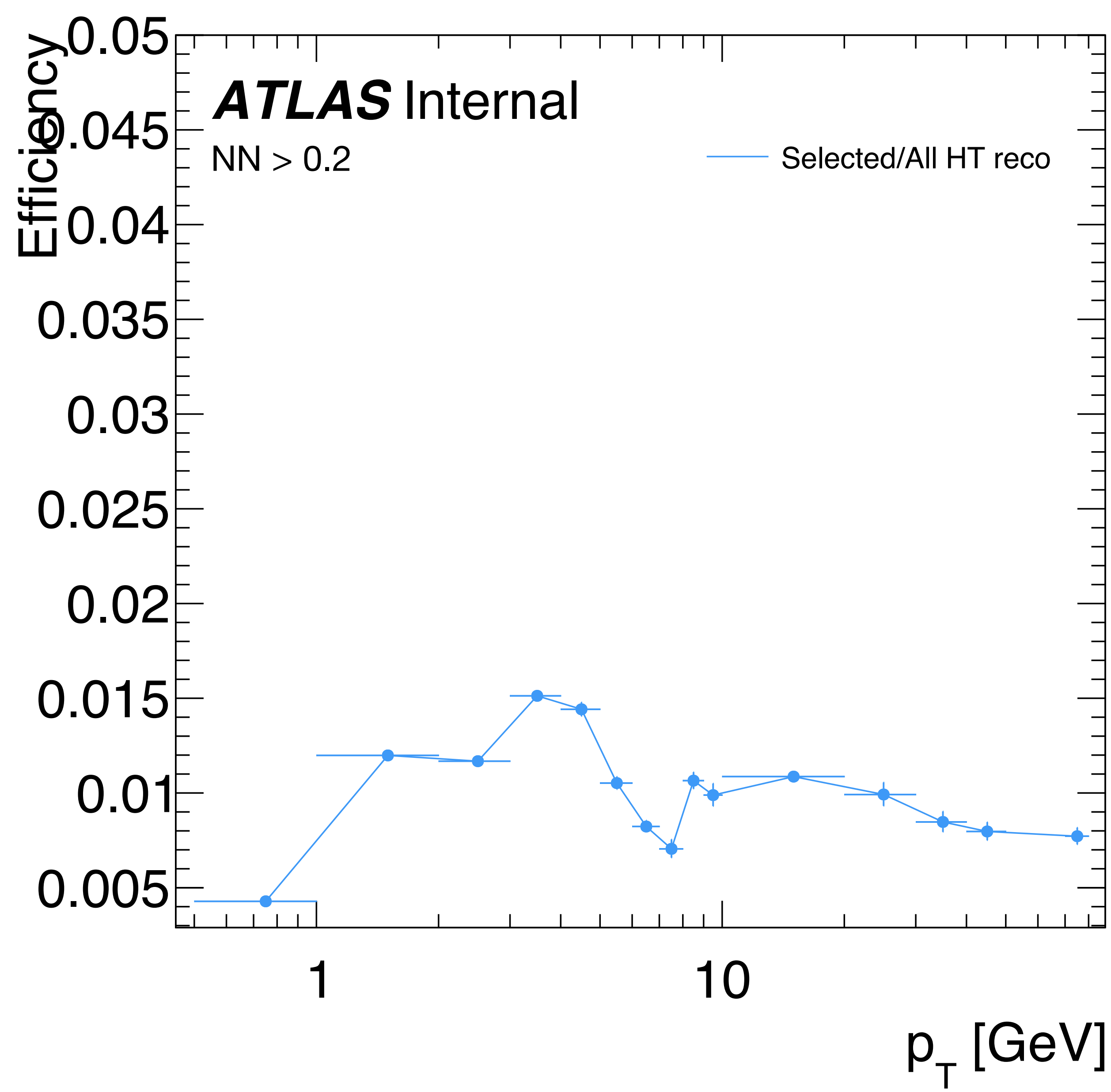
Truth Efficiency	Total	NN > 0.2	NN > 0.1	NN > 0.05	NN > 0.01	NN > 0.001	NN > 0
	3103	2769	2795	2826	2847	2862	2883
All Truth && HT		95.2%	96.0%	97.1%	97.8%	98.4%	99.1%

Efficiency as a function of pT

Truth tracks



All HT tracks



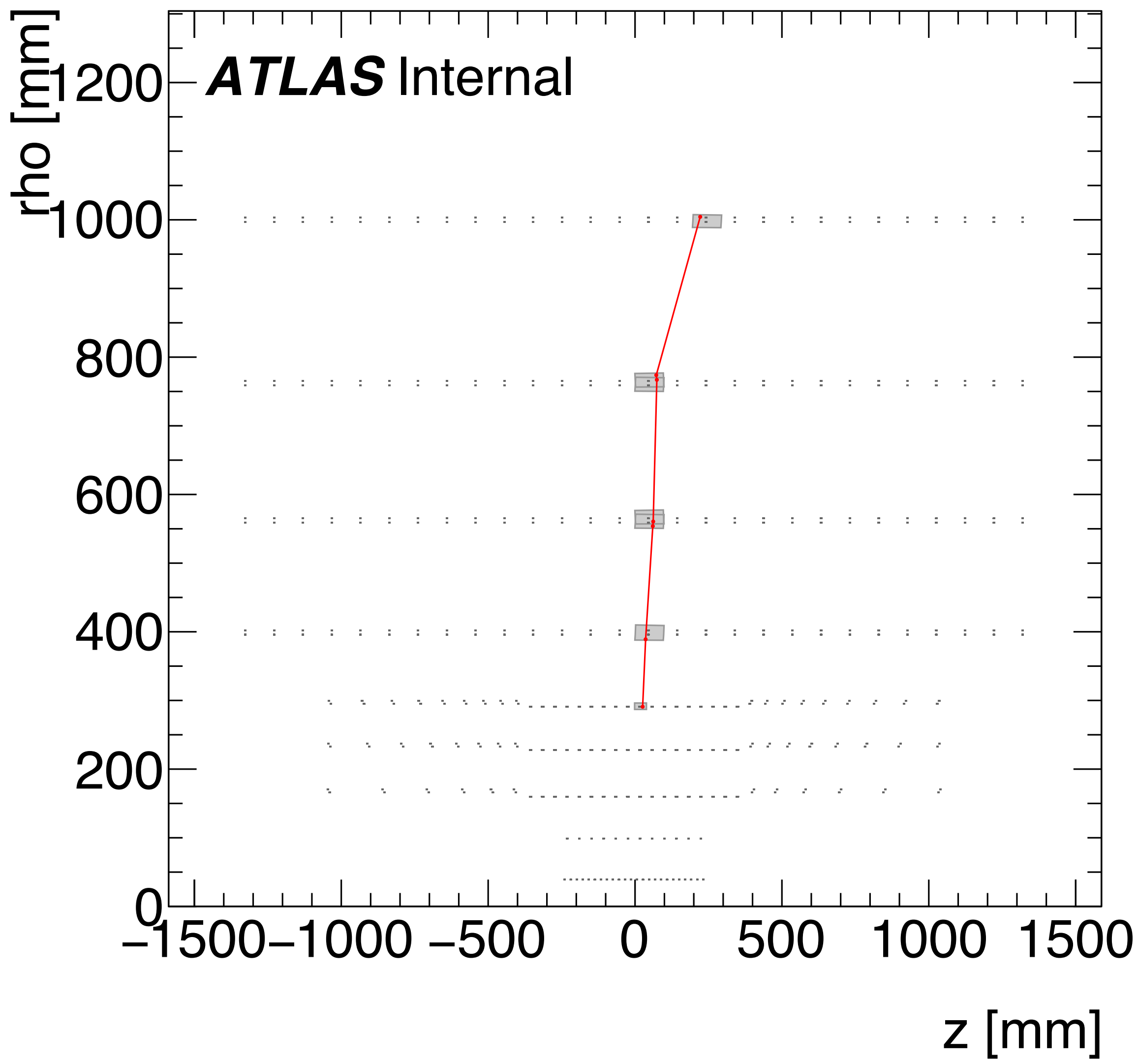
Efficiency of HT

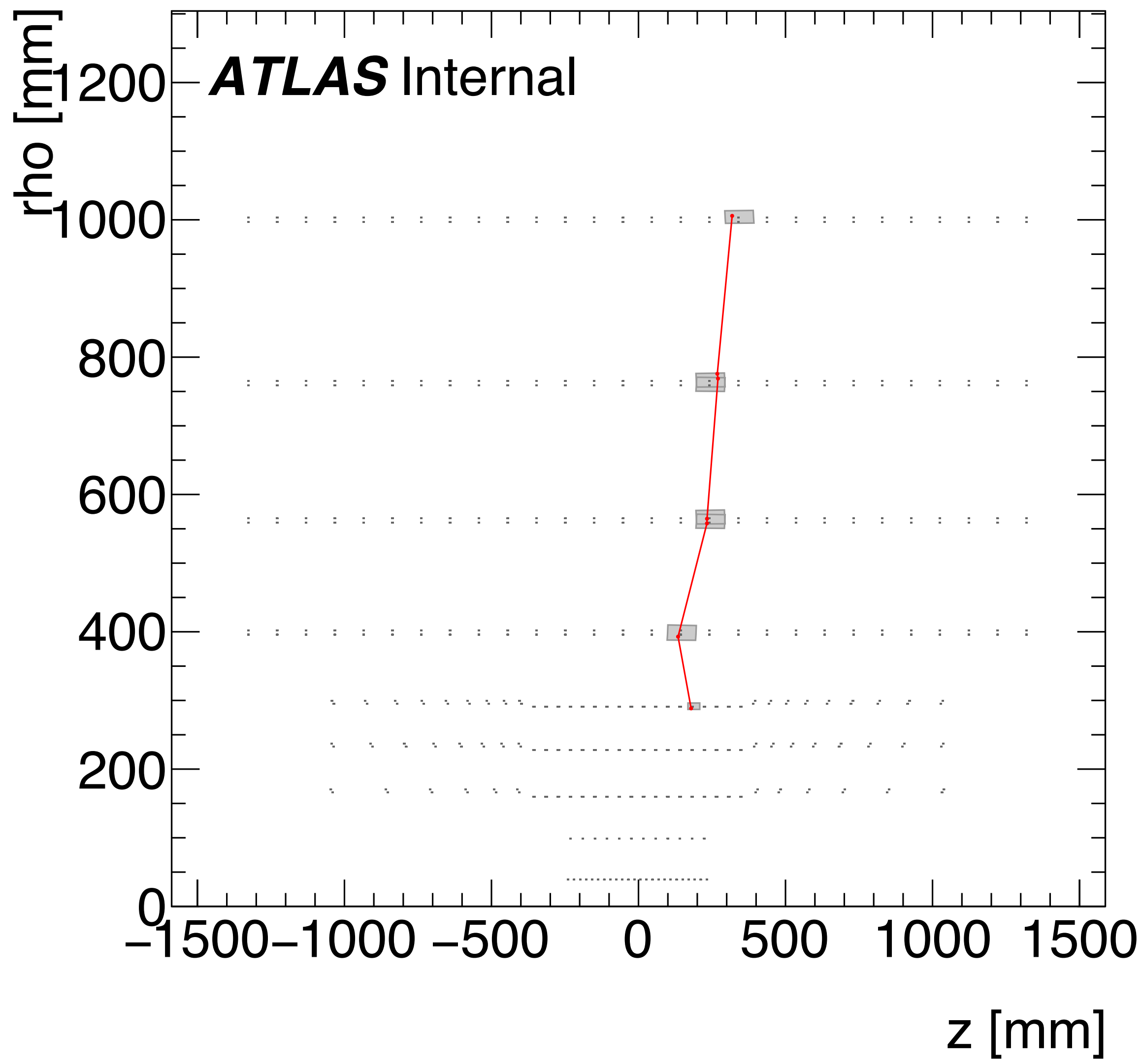
Efficiency of after NN cut

Efficiency of NN cut wrt to HT

Caveat: These are absolute efficiency wrt the truth,
updated results will be wrt offline tracks as per the TF mandate

Tracks rejected by the NN

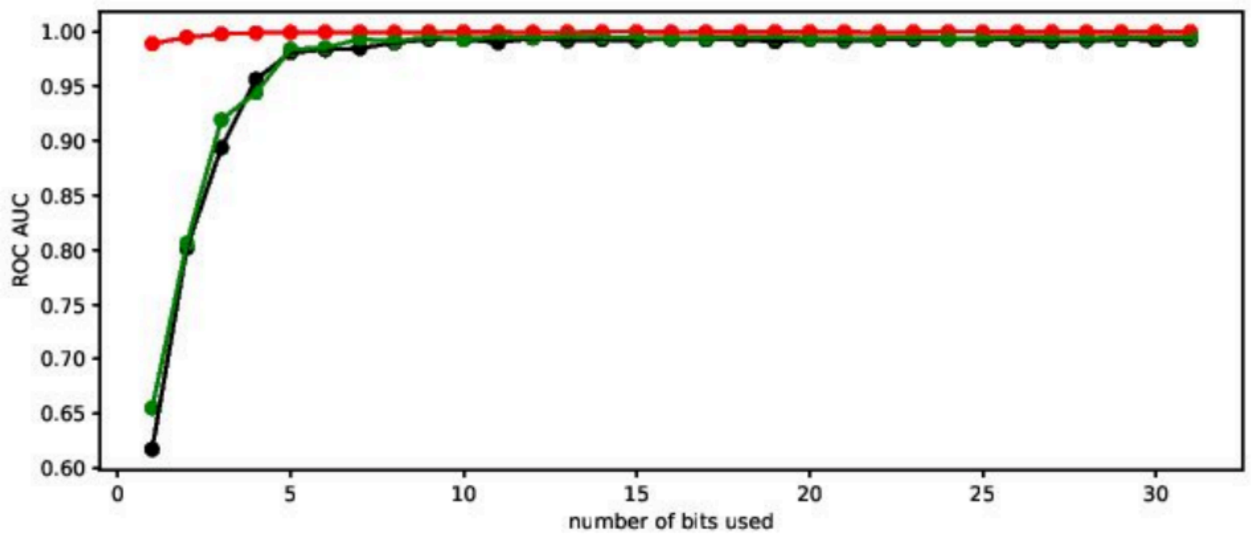




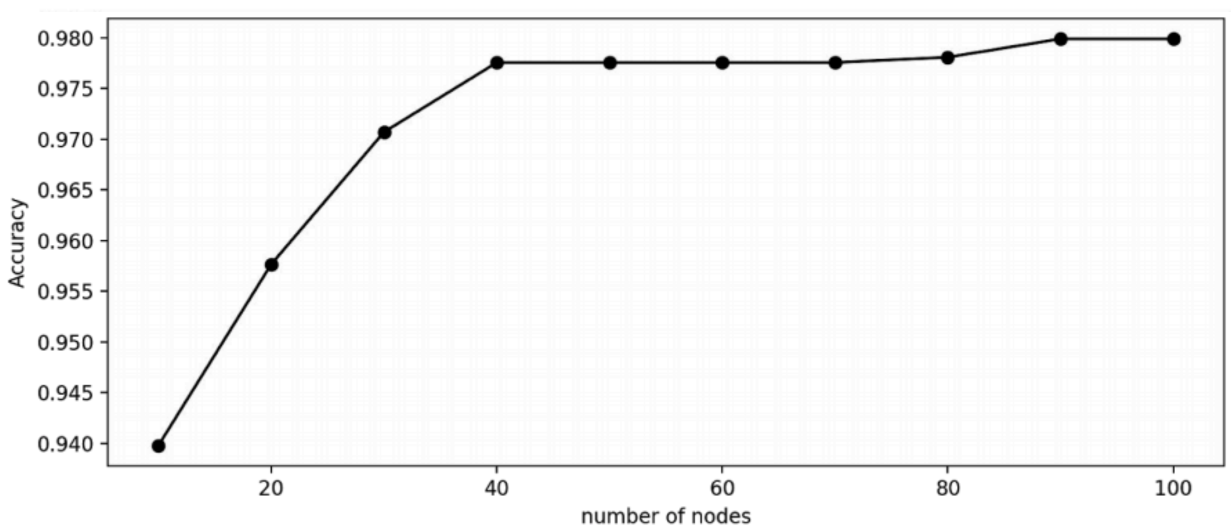
These track have Truth probability score > 0.7, but rejected by NN
Need to check if these are reconstructed by reco

NN on FGPA

- To run NN inference on a FPGA, need to quantize and synthesize it
- HLS4ML framework: Generates C-code that Xilinx tools can synthesize
- Trained with quantized aware nodes - QKeras
    - Weights & bias are quantized while training - easily translatable on FPGA without any loss
- First pass at optimizing the hyper parameters - Fine tuning to come when HT parameters are fixed



Optimizing number of bits to represent weights
(different colours are different fakes types)



Optimizing the number of nodes in the NN

FPGA: Resource estimates

- First estimates - Can be optimized for latency and for resource usage
- Will need a few more resources for preprocessing the input

Preliminary

Latency

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
11	11	55.000 ns	55.000 ns	1	1	function

FPGA resources

Xilinx FPGA AlveoU250

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	1	3589	34521	140549	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	8402	-	-
Total	1	3589	42923	140591	0
Available	5376	12288	3456000	1728000	1280
Utilization (%)	~0	29	1	8	0

Pt regression

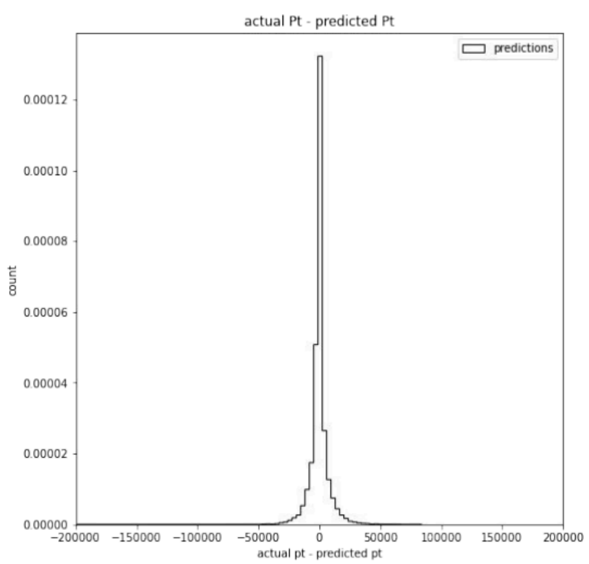
- Alex started looking into:
  - Simultaneous track classification and pt regression
    - Given a sequence of hits, determine the track parameters (focus on pt for now)
    - Does not require input of magnetic field, detector geometry info, etc.
  - Need a more complicated network as expected

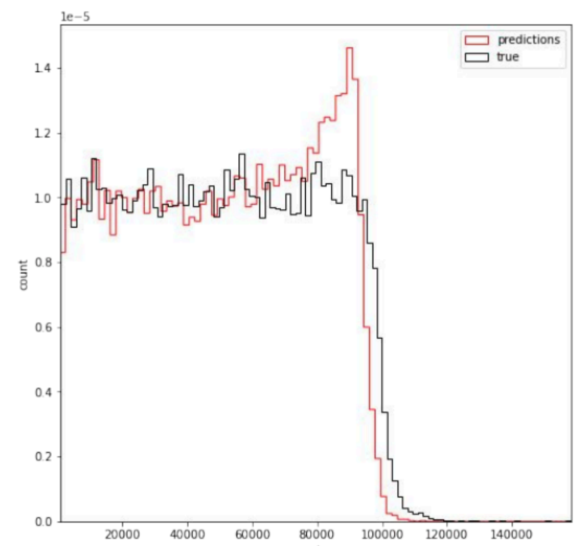
1D-CNN

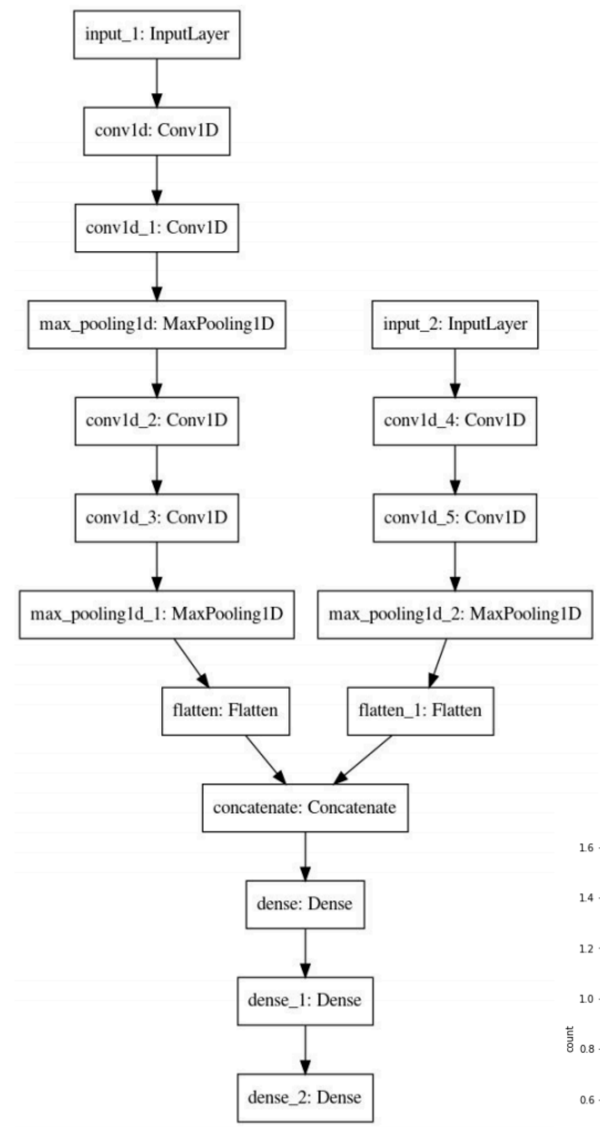
Model: "sequential_4"

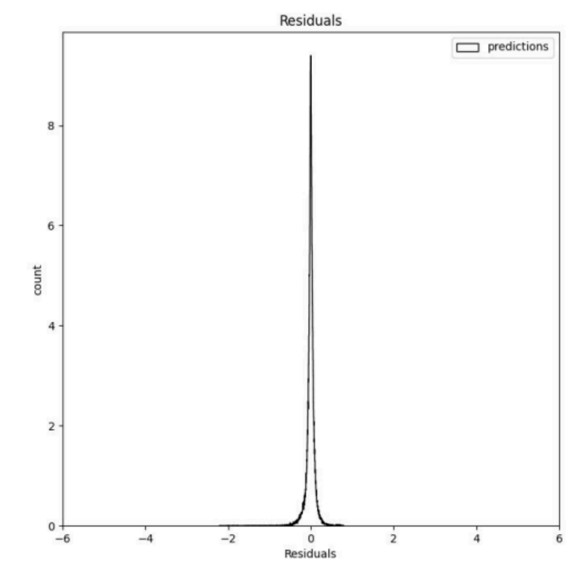
Layer (type)	Output Shape	Param #
conv1d_16 (Conv1D)	(None, 19, 128)	1280
conv1d_17 (Conv1D)	(None, 17, 128)	49280
max_pooling1d_8 (MaxPooling1	(None, 8, 128)	0
conv1d_18 (Conv1D)	(None, 6, 64)	24640
conv1d_19 (Conv1D)	(None, 4, 64)	12352
max_pooling1d_9 (MaxPooling1	(None, 2, 64)	0
flatten_4 (Flatten)	(None, 128)	0
dense_8 (Dense)	(None, 32)	4128
dense_9 (Dense)	(None, 16)	528

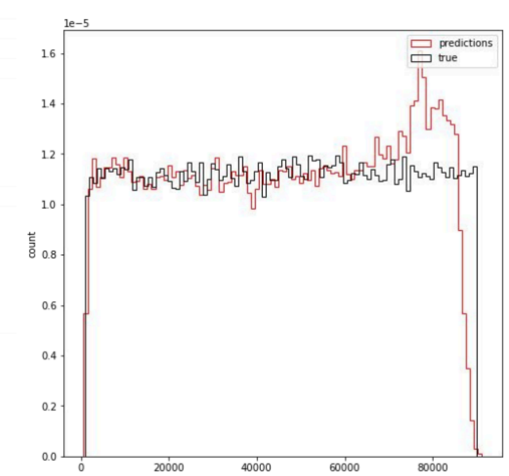
Total params: 92,208
Trainable params: 92,208
Non-trainable params: 0









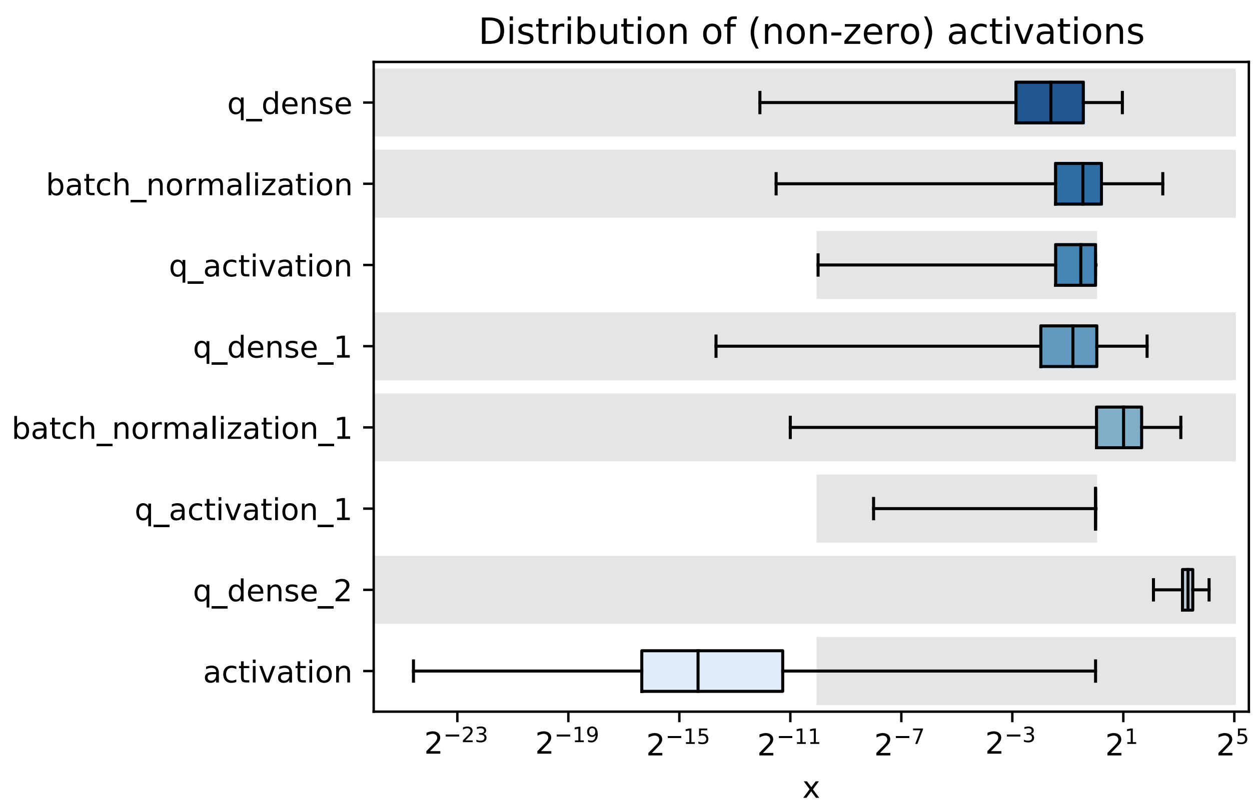


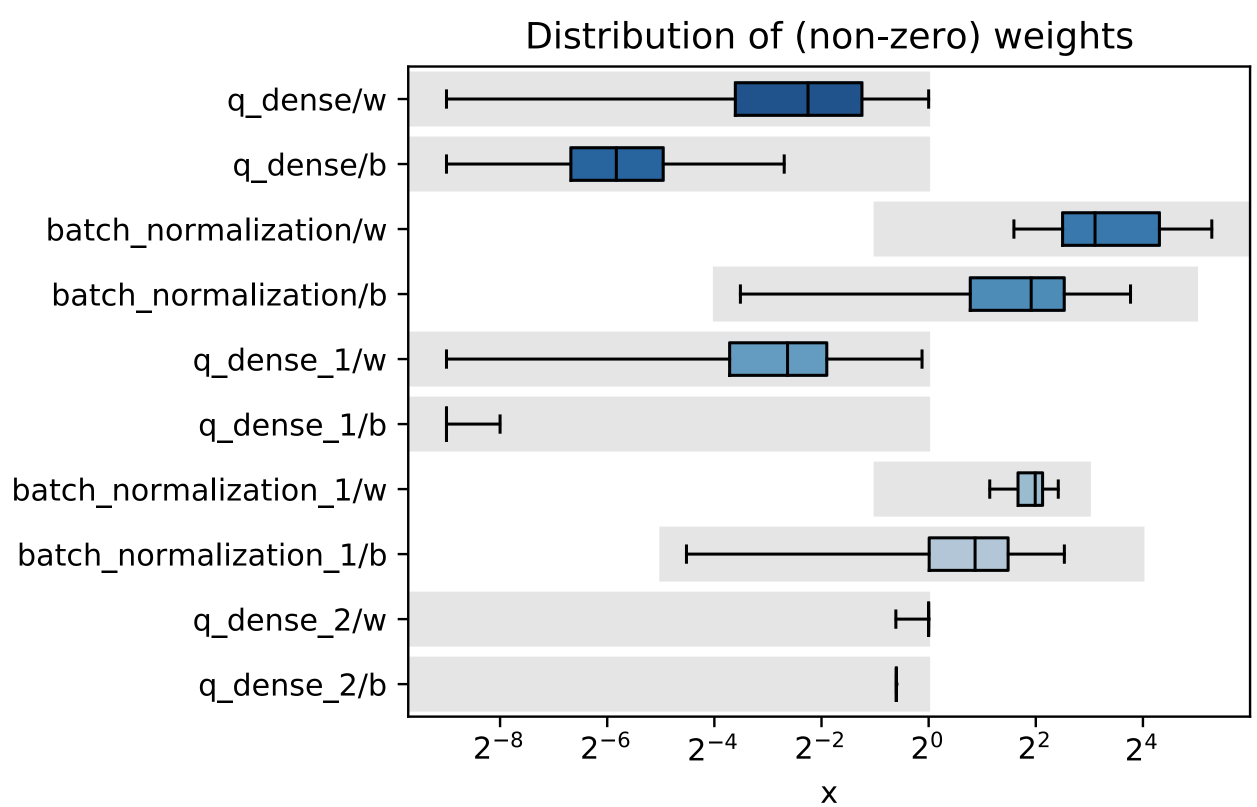
Conclusions

- First pass at using ML for duplicate removal after HT
    - Initial results look extremely promising - further tuning to come once HT settings are finalized
- Have an idea of the resource estimates for FPGA usage
    - NN also executable on CPUs/GPUs
- Further iterations on going to allow for estimate of efficiency with respect to offline tracking
- Start looking at a coarse track fit - initial studies ongoing

Backup

NN weights


Distribution of (non-zero) activations
q_dense
batch_normalization
q_activation
q_dense_1
batch_normalization_1
q_activation_1
q_dense_2
activation
2^{-23}
2^{-19}
2^{-15}
2^{-11}
2^{-7}
2^{-3}
2^{1}
2^{5}
x


Distribution of (non-zero) weights
q_dense/w
q_dense/b
batch_normalization/w
batch_normalization/b
q_dense_1/w
q_dense_1/b
batch_normalization_1/w
batch_normalization_1/b
q_dense_2/w
q_dense_2/b
2^{-8}
2^{-6}
2^{-4}
2^{-2}
2^{0}
2^{2}
2^{4}
x