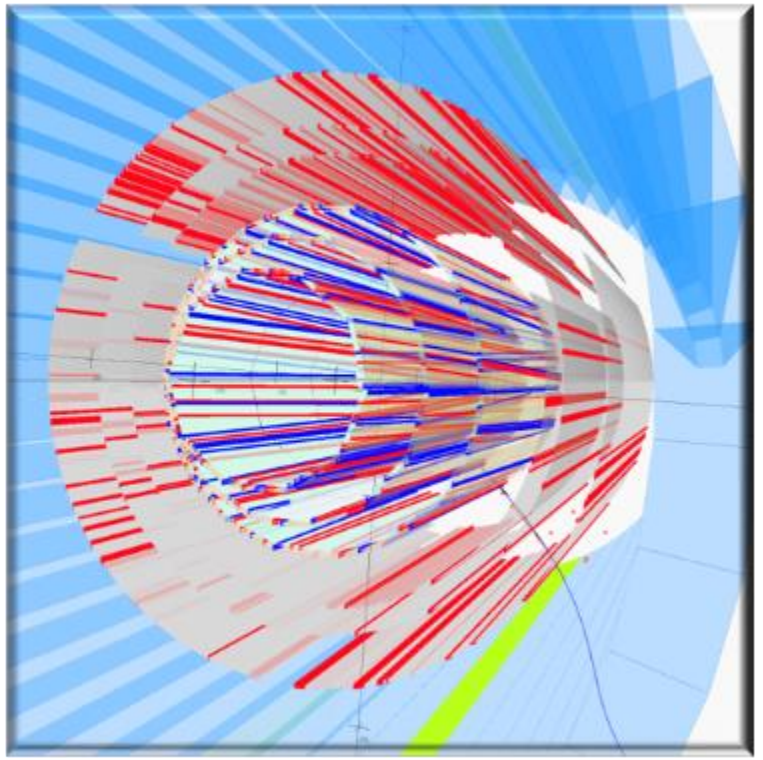
Plans for AI/ML Assisted and Conventional Central Tracking

Richard Tyson, Veronique Ziegler,
Tongtong Cao, Raffaella De Vita
Pierre Chatagnon, Mathieu
Ronayette, Bhawani Singh



CLAS12 Collaboration Meeting
November 18, 2015

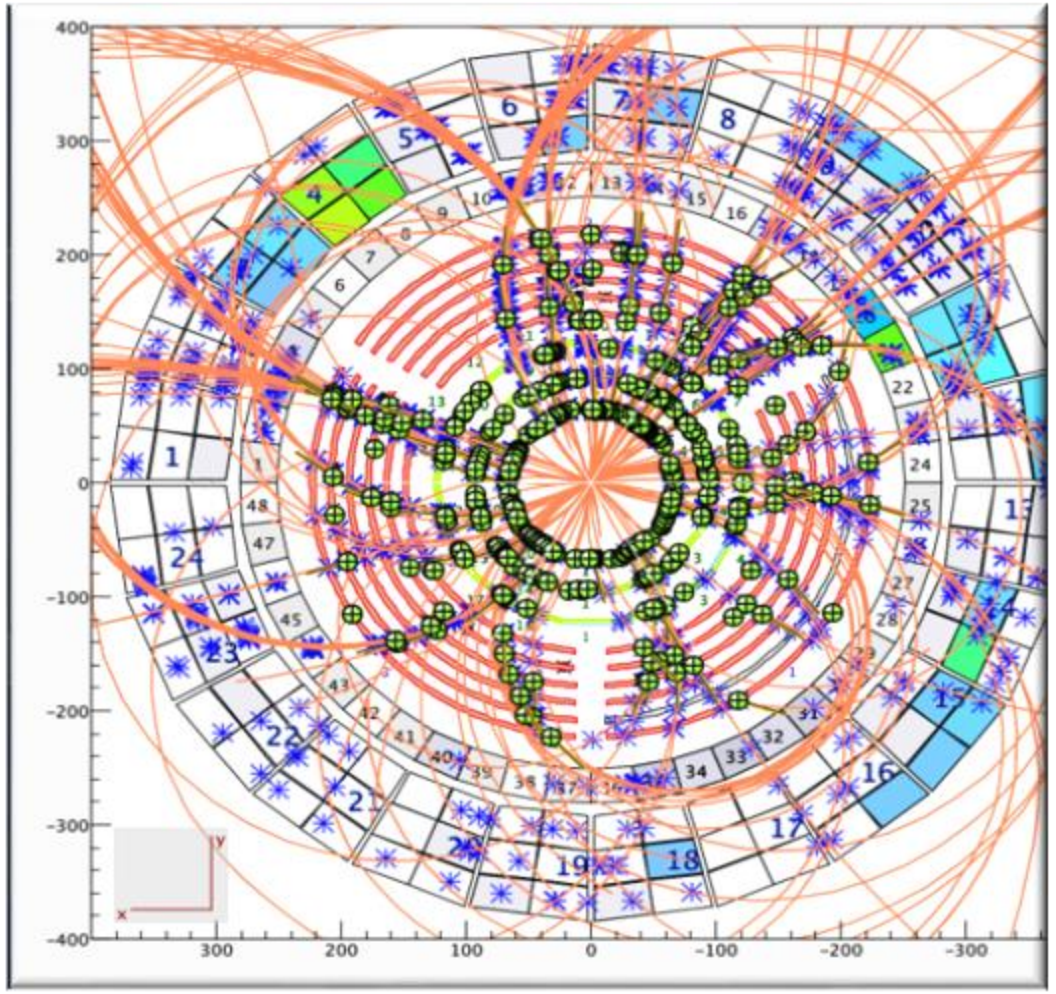
Jefferson Lab clas

AI-Driven "Denoising" Algorithm for CVT Tracking ~ Motivation

Seed Multiplicity in CVT Tracking – RGE Data Example

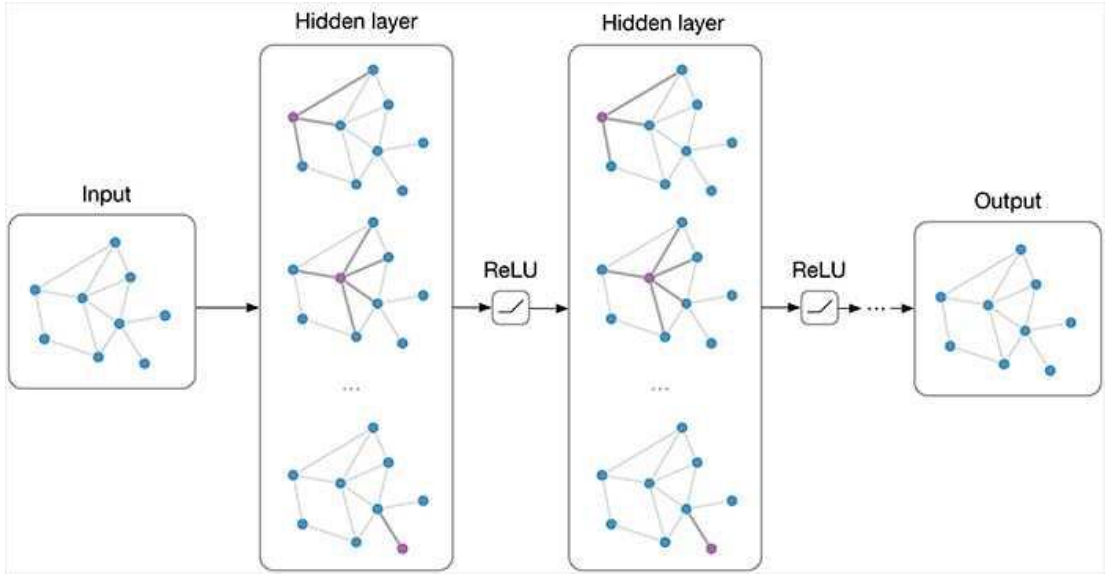
- Many seeds (track candidates) share clusters of hits.
- Difficult to select the "good" candidate amongst a set of candidates that share clusters with it.
- Leads to a large number of "fake tracks" → wrong clusters-on-track selection (combinatorials), loss of tracking efficiency, increased computation times
- Worsening of track parameters resolution due to background hits polluting clusters, and resulting in wrong clusters-on-track selection

- Need for development of denoising algorithm(s)



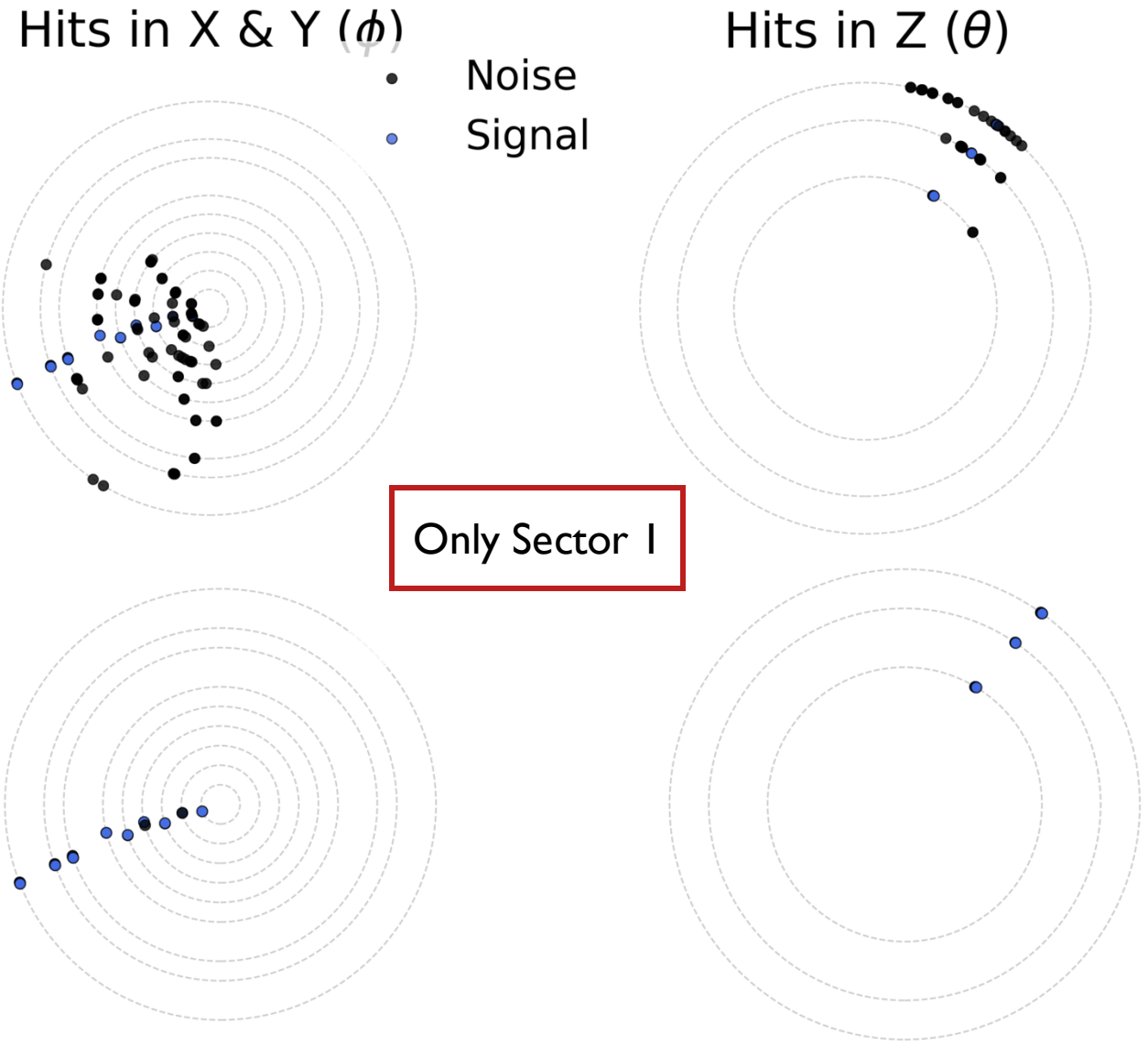
Denoising Model: Graph Neural Networks (GNN)

- Graph are made of nodes (eg hits) & edges.
- During learning, "message passing" updates each node with information from other nodes.
- This is learned (eg what information is passed is the output of a small neural network).
- GNNs can be used in different ways:
  - Node classification (denoising)
  - Edge classification (track finding)

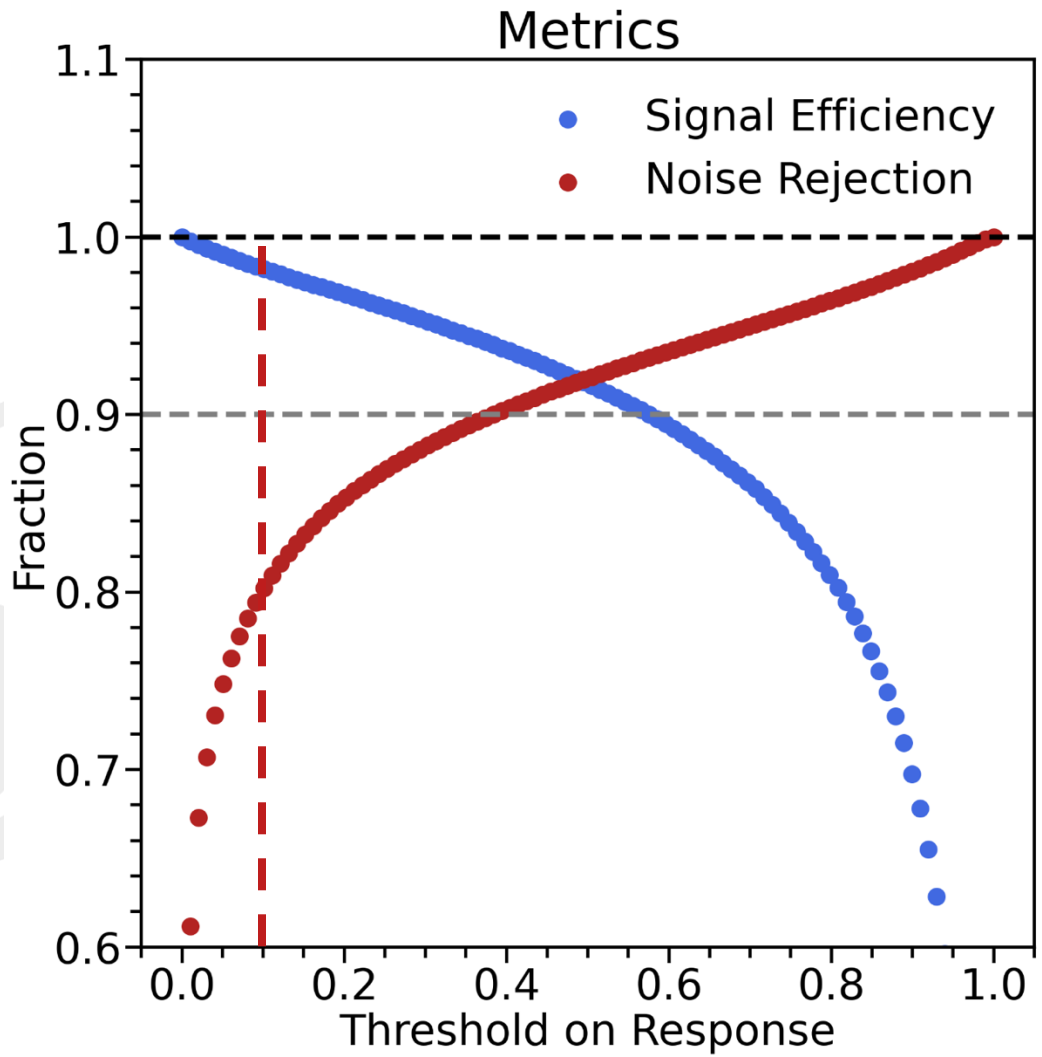


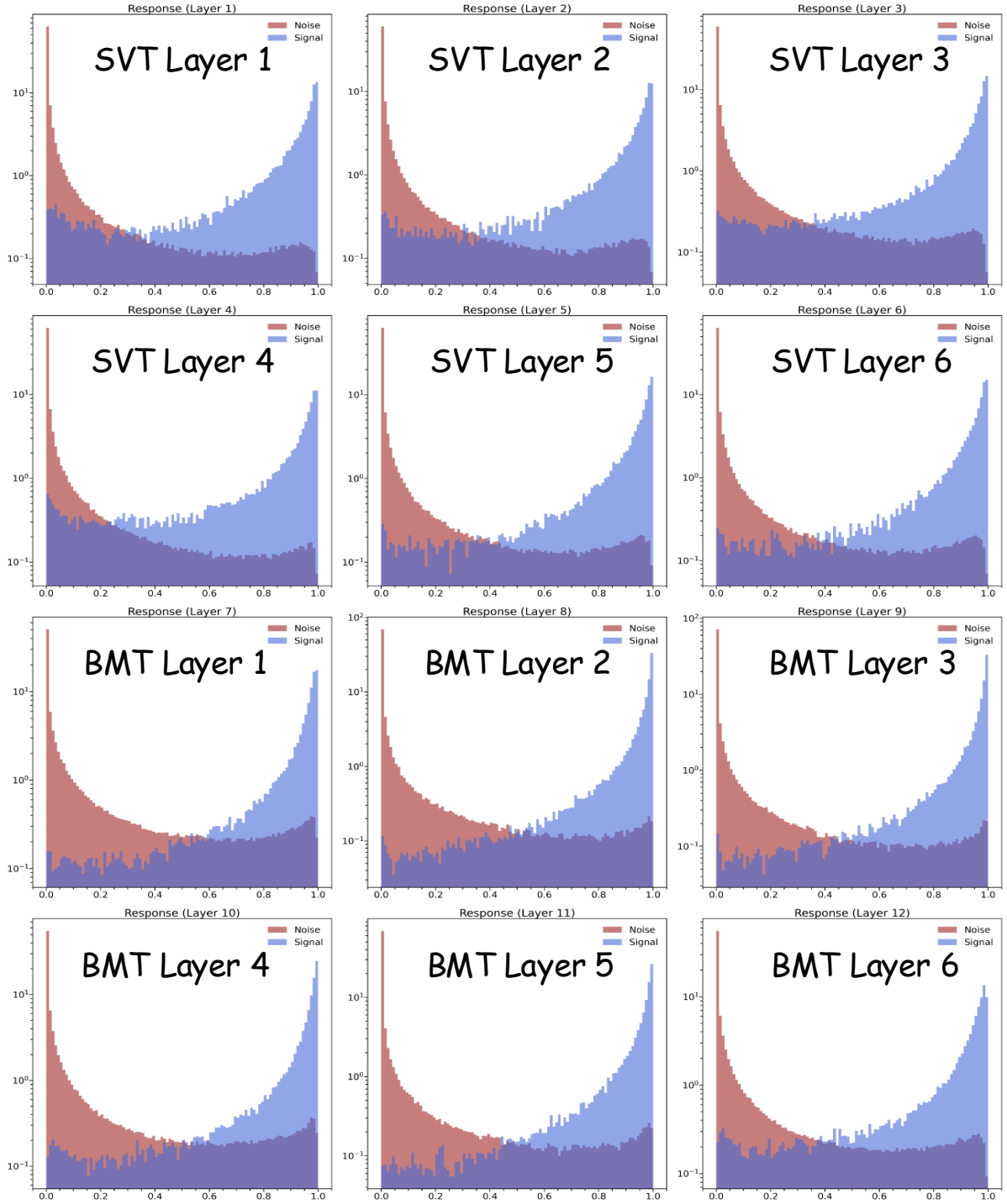
Denoising AI Development

- MC sample with single generated muon track (signal hits, order 0) and merged background (order!=0).
- Input variables are:
  - Sector, layer, strip
  - x, y, z (1 & 2)
  - Centroid and seed weights (normalized distance to cluster centroid/seed)
- High segmentation in CVT means we cannot use fixed size arrays → employ different technology to FD (GNNs instead of CNNs).
- Data processing, algorithm, training and coatjava inference code is available (see github).



Performance





AI Developments Next Steps

Doing exploratory work on improvements to CVT.

We have preliminary results that look encouraging

There is still a lot to do → expect timescales of a year.

Workforce:

- Mathieu, Pierre, Bhawani will work on improving AI/ML model
- Veronique will work on conventional improvements to tracking
- Tongtong & Veronique will work on coatjava integration and validation

Conventional Tracking Improvements – CTOF Matching

All seeds

- Many seeds (track candidates) share clusters of hits
- Difficult to select the "good" candidate amongst a set of candidates that share clusters with it
- Seeds are fit using a global method that assumes a constant 5-T field solenoidal field → using swimming to fit each seed not viable; global chi^2 and NDF used to select the seed from a set of seeds that share clusters → when the "real" seed (MC studies) has missing layer(s) in its trajectory, the wrong seed can be selected.

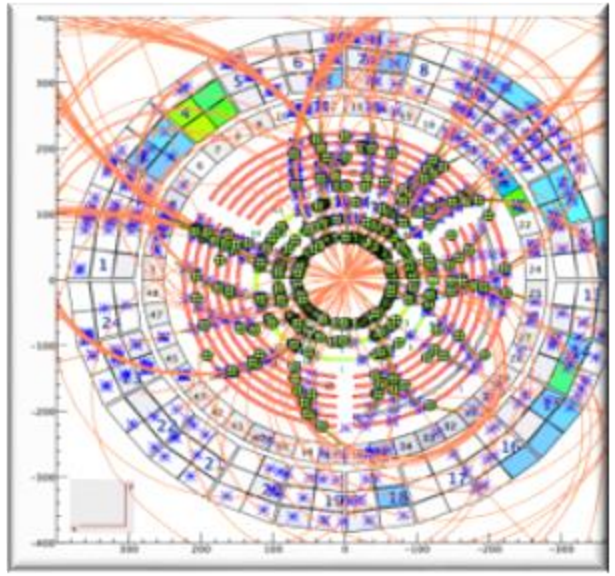
Seeds matched to the CTOF

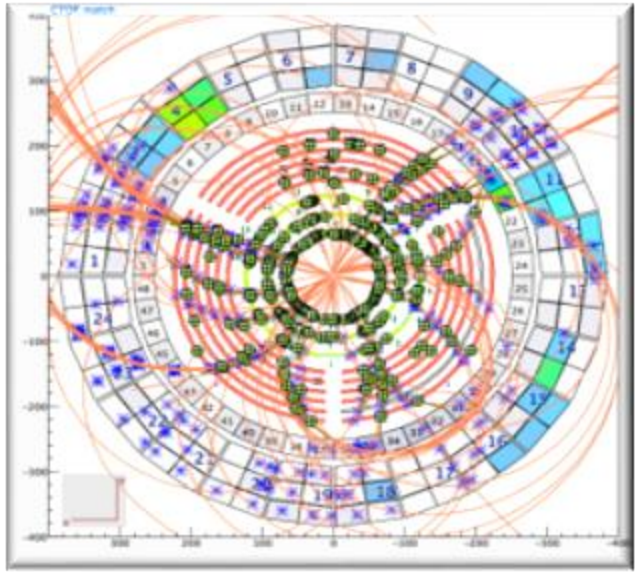
- Reducing the number of seeds is helpful to diminish the number of candidates sharing clusters and the overall number of seeds passed on to the fitter (faster tracking)
- Matching to the CTOF done by reading the CTOF hits bank and selected hit with OR/AND TDC/ADC hits (only ADC hits are displayed in CED).

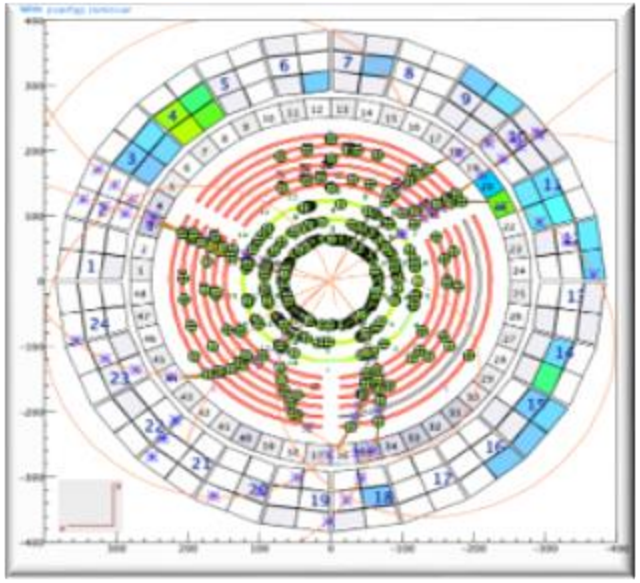
Selected seeds matched to the CTOF

- Seeds remaining after "best" candidate selection amongst a group of seeds that share clusters of hits.

- With CTOF matching, fewer selected seeds which have better chi^2/NDF
- Reduction of ghost tracks to be validated in MC (with background merging)







Plans

- Further improvements in seed selection
  - Current algorithm uses sorting algorithms by chi^2/NDF
  - Investigating missing hits on track using layer efficiency tables → if a hit on track is missing in an efficient layer ➔ flag as potential ghost track

- Investigating using roads for SVT seeding
  - Concept:
    - Development of a precomputed pattern-matching algorithm (road finder).
    - A road = expected strip hits for a track hypothesis.

Conventional Tracking – Roads

Goal: Convert millions of compressed road patterns into an efficient lookup structure.

Roads required to intersect the CTOF
→ ***built-in CTOF-matching***

For a given CTOF paddle each road (with trajectory landing on that paddle) saved in gz file:
short nElements
nElements:
long packedStripID (16-bit encoded sector+layer+strip)

Process:

1. Load per-paddle .bin.gz files

1. Each file contains *bit-packed roads, created by a script*
2. Each road = sequence of (layer, sector, strip) hits created by computing helical tracks trajectories looping over, q, p, theta, phi, z, and determining the intersecting surface and nearest strip from geometry

2. Extract "index keys" *(from file created by index-building script)*

1. For each element:
   bin in phi corresponding to the strip implant end point (360 bins)
   key = (phiBin<<16 | sector<<8 | layer) embedded in a long

3. For each paddle, build temporary structure:

Map< key → list of roadIds >

4. Compact into final UBER index:

Map<Integer paddle → Map<Long key → int[] roadIds>>

5. Store full road list in RoadIndex.UberEntry[] roadTable

→ Each entry stores (paddle, bitOffset, nElements)

→ No full decode yet (lazy loading)

UBER index built:
keys → road ID lists
roadTable → bit offsets

Conventional Tracking – Roads

Fast Road Lookup (Event Processing)
Input per event: A set of cluster keys per paddle

R*everse inverted index*:
•Instead of iterating roads → check clusters
•We iterate clusters → check road membership
Complexity:
O(clusterKeys + matchedRoads)
No decoding done yet.

Lazy Road Decode + LRU Cache

Steps:
1.Build **64-bit cache key**:
[paddle:16][offsetBytes:32][nElements:16]
2.Check LinkedHashMap LRU cache (20k entries)
3.If miss:

1. Seek into paddle byte array using bit-skips (offsetBytes)
2. Read (layer, sector, strip) triplets
3. Build CompactRoad
4. Insert into cache

Why lazy decode?
•50M+ possible roads
•Only a few hundred decoded per event
•Saves GBs of RAM

Conventional Tracking – Roads

Seeding SVT Tracks from Roads

Seed using roads with input: clusters, CTOF paddles, paddleBytes (bit packed roads information for a given paddle)
I. Obtain the list of CTOF paddles with hits in the event
II. Per paddle:
1. Build the same cluster keys used for road indexing
2. findRoad() → best matching UBER entry
3. getRoad() → decode only that road
4. matchRoadToClusters():
    1. Match by (sector,layer)
    2. Pick closest strip distance per element
5. If ≥ **4 matched clusters → accept as SVT seed**

Seeding CVT Tracks from Roads

Seed CVT tracks using the conventional seeding algorithm (SSA, SLA) for each set of clusters matched to a road, including BMT clusters

Each such seed is fitted using the Kalman Filter algorithm

- In development…

Conventional Tracking – Roads

Validations

- Roads-building binning:
  - p → 10 MeV
  - Theta, phi → 1 deg.
  - z (RGA) → target length / 5 (small z sensitivity for SVT)

- Performance
  - Memory footprint
  - Reconstruction speed
  - Tracking efficiency
  - Impact on resolution

Plans

- Run validations on phase space coverage for roads
- Validate feasibility of incorporation in reconstruction stack (performance)
- Validate tracking performance

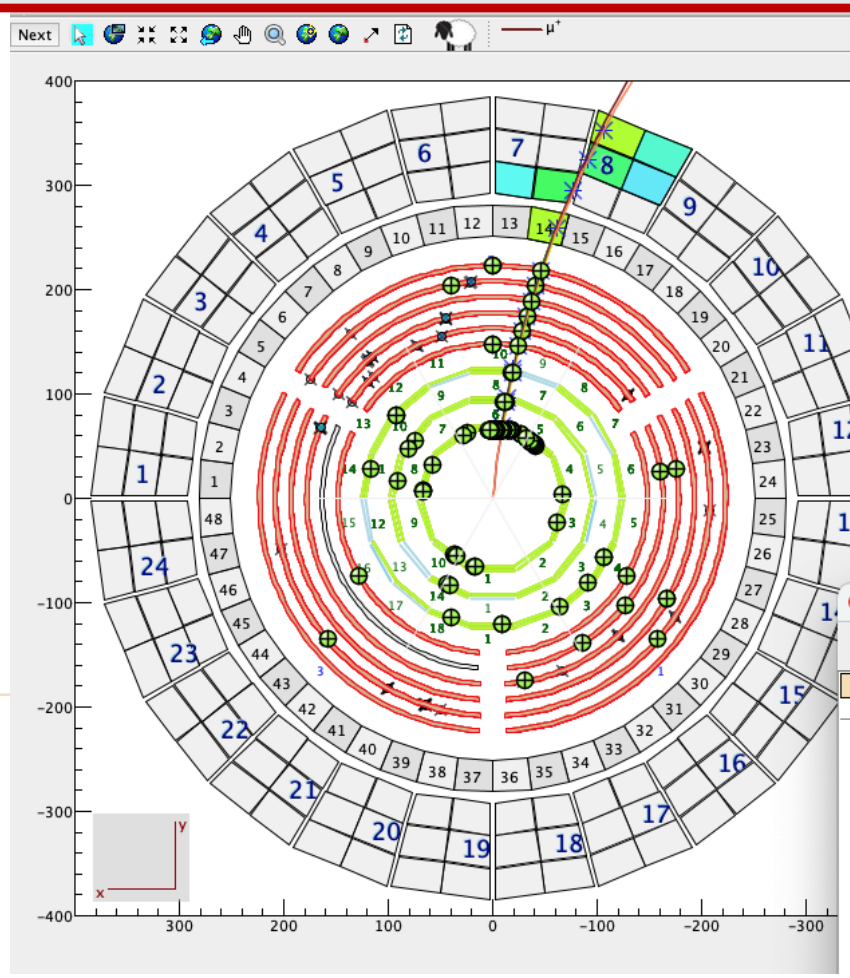
BACKUPS

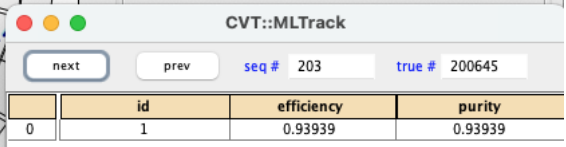
Dedicated ML Banks

Bankdefs: cvtnn.json

```
{
   "name": "CVT::MLHit",
   "group": 20500,
   "item" : 52,
   "info": "reconstructed hits (2nd pass tracking)",
   "entries": [
      {"name":"id",           "type":"S", "info":"id of the hit"},
      {"name":"sidx",         "type":"S", "info":"row in the MLSeed bank"},
      {"name":"tidx",         "type":"S", "info":"row in the MLTrack bank"},
      {"name":"rectid",       "type":"S", "info":"reconstructed track id associated with the hit"},
      {"name":"recsid",       "type":"S", "info":"reconstructed seed id associated with the hit"},
      {"name":"mctid",        "type":"S", "info":"MC track id associated with the hit"},
      {"name":"sector",       "type":"B", "info":"sector"},
      {"name":"layer",        "type":"B", "info":"layer (1...6)=SVT; (7...12)=BMT"},
      {"name":"type",         "type":"B", "info":"detector type: 0=SVT; 1=BMT-Z; 2=BMT-C"},
      {"name":"strip",        "type":"S", "info":"strip number"},
      {"name":"order",        "type":"B", "info":"order 0=MC hit-on-track; 1=noise hit"},
      {"name":"cid",          "type":"S", "info":"associated cluster id"},
      {"name":"cweight",      "type":"F", "info":"1/1+|normalized difference to associated cluster centroid|"},
      {"name":"sweight",      "type":"F", "info":"1/1+|normalized difference to associated cluster seed|"},
      {"name":"x1",           "type":"F", "info":"geometric strip first end-point x-coordinate"},
      {"name":"y1",           "type":"F", "info":"geometric strip first end-point y-coordinate"},
      {"name":"z1",           "type":"F", "info":"geometric strip first end-point z-coordinate"},
      {"name":"x2",           "type":"F", "info":"geometric strip second end-point x-coordinate"},
      {"name":"y2",           "type":"F", "info":"geometric strip second end-point y-coordinate"},
      {"name":"z2",           "type":"F", "info":"geometric strip second end-point z-coordinate"}
   ]
},
```

```
{
   {
      "name": "CVT::MLSeed",
      "group": 20500,
      "item" : 63,
      "info": "reconstructed seeds (2nd pass tracking)",
      "entries": [
         {"name":"id",           "type":"S", "info":"id of the track"},
         {"name":"purity",       "type":"F", "info":"seed purity: Nb of reconstructed true hits in seed / total hits on seed"},
         {"name":"efficiency",   "type":"F", "info":"seed purity: Nb of reconstructed true hits in seed / MC hits in gemerated track"}
      ]
   },
},
{
   {
      "name": "CVT::MLTrack",
      "group": 20500,
      "item" : 64,
      "info": "reconstructed seeds (2nd pass tracking)",
      "entries": [
         {"name":"id",           "type":"S", "info":"id of the track"},
         {"name":"purity",       "type":"F", "info":"seed purity: Nb of reconstructed true hits in track / total hits on track"},
         {"name":"efficiency",   "type":"F", "info":"seed purity: Nb of reconstructed true hits in track / MC hits in gemerated track"}
      ]
   }
}
```

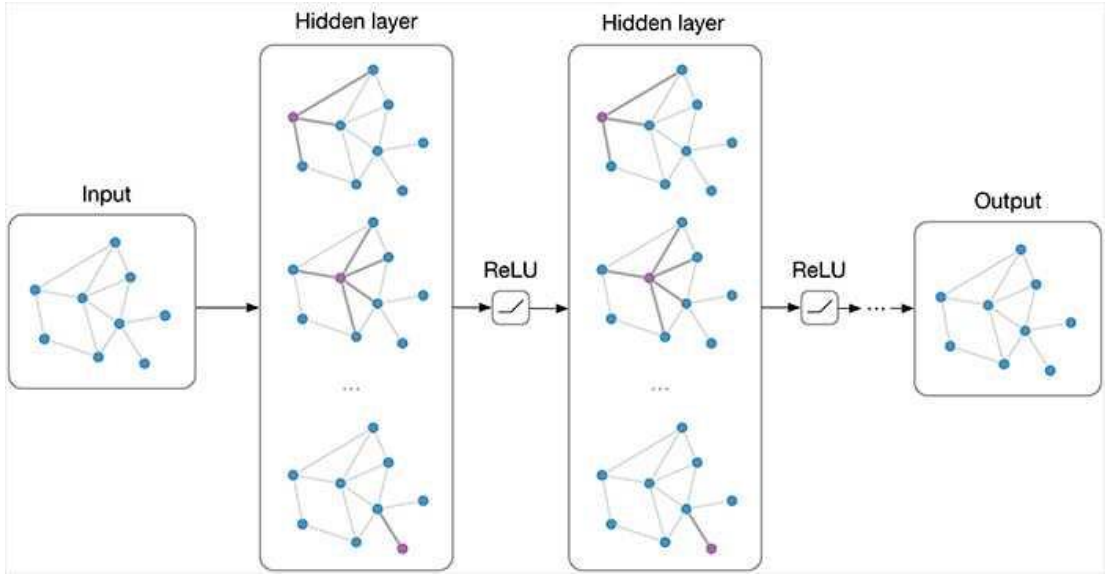




- n_T = #reconstructed hits on track with order 0 (true hits on track)
- n_{Tot} = #reconstructed hits on tracks
- n_G = #generated hits belonging to simulated track
- Purity = n_T / n_{Tot}
- Efficiency = n_T / n_G
- If purity < efficiency: list of HOTs contains fakes
  & # HOTs > generated HOTs

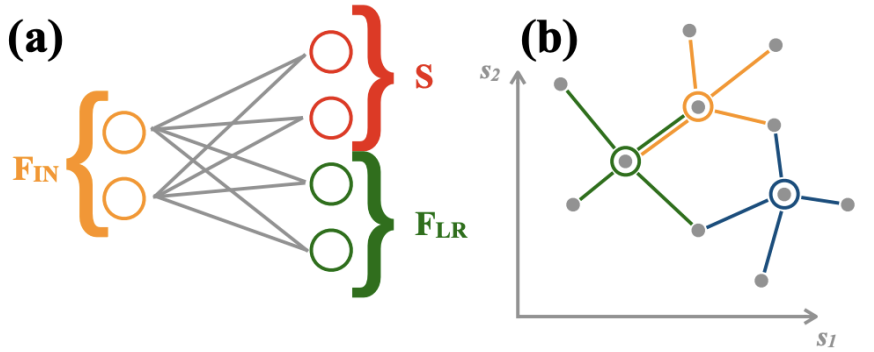
Denoising Model: Graph Neural Networks (GNN)

- Graph are made of nodes (eg hits) & edges.
- During learning, “message passing” updates each node with information from other nodes.
- This is learned (eg what information is passed is the output of a small neural network).
- GNNs can be used in different ways:
  - Node classification (denoising)
  - Edge classification (track finding)



GravNet Layers (arxiv:902.07987)

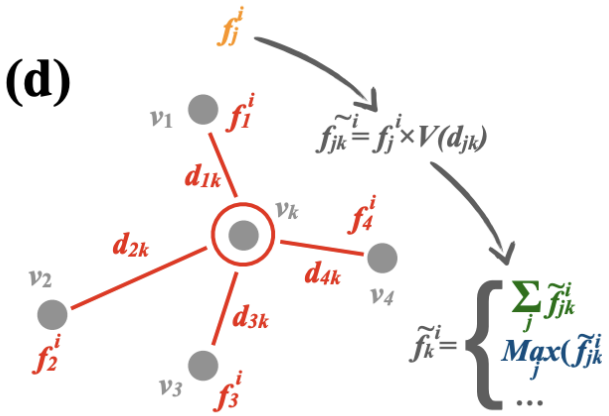
- GravNet offers a “smart” way of grouping nodes before message passing (useful due to high hit multiplicity).
- A neural network predicts a latent space position for each hit
- We group the hits by distance in latent space (kNN)
- Message passing is weighted by distance
- A neural network outputs the probability that each hit is signal, based on info from the hit and neighbours



Note: kNN in many dimensions is slow.

In 1D use sorting (fast).

In ND, first sort over random dimension, select 2*k, then do knn

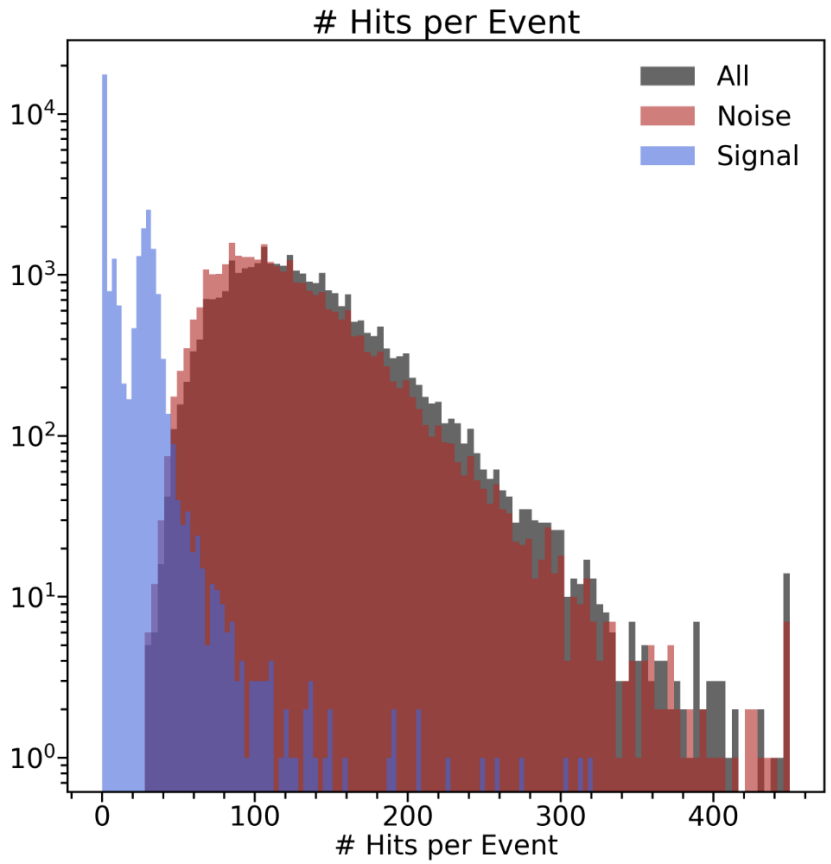


Model Restrictions

- Due to number of hits per event, we cannot create edges between all hits:
  - GravNet reduces impact of this.
  - Further restrict hits to sectors (based on SVT)
  - Restrict by ϕ? Adjacent hits in a layer? Distance between neighbour layers?

- I thought that the Deep Java Library in coatjava meant we need stick to standard PyTorch:
  - Use padding eg define a maximum number of hits per event, pad with zeros for the rest of it
  - Padded events are not included in message passing.

```
# Sector mapping
sector_mapping = {
    1: {  # File 1
        (1, 2): [1, 2, 3, 4],
        (3, 4): [1, 2, 3, 4, 5, 6],
        (5, 6): [1, 2, 3, 4, 5, 6, 7],
        (7, 12): [1],
    },
    2: {  # File 2
        (1, 2): [4, 5, 6, 7, 8],
        (3, 4): [6, 7, 8, 9, 10],
        (5, 6): [7, 8, 9, 10, 11, 12, 13],
        (7, 12): [2],
    },
    3: {  # File 3
        (1, 2): [8, 9, 10],
        (3, 4): [10, 11, 12, 13, 14],
        (5, 6): [14, 15, 16, 17, 18],
        (7, 12): [3],
    },
}
```



Improvements

To Do

Need 10x more training data
Further restriction in graph formation?
Sort out training data
Sort out variables
Try out different hyperparameters (record efficiency, purity, training and inference times) Integration in Coatjava

Coatjava Integration

Example code in the github & networks in these slides
Can use similar engine to DC denoising
Needs functionality:

- parse hits bank into arrays
- change order based on output, mask in tracking
- Measure full reconstruction efficiency & resolutions

Training Data

Events with only background, keep?
Events with partial track due to splitting into sectors, keep?
More than 1 track per event
10x more data means ~20h training, we can randomly sample from full dataset and train in subsets of data for subsets of total number of epochs

Variables

Polar coordinates instead of X/Y/Z?
Cluster info?
What variables are ”well” simulated?

Hyperparameter

Decrease learning rate for smoother loss v epochs
Latent space dimension (s=1 for now)
Number of neighbours (k=30)
Message passing space (34)
Number of GravNet layers (16)
Output neural net layout