HyperTrack

Neural Combinatorics for High Energy Physics

Mikael Mieskolainen

m.mieskolainen@imperial.ac.uk

Imperial College London

Combinatorics with Deep Learning

Emerging field, learning of combinatorial computations with discrete object sets having variable input and output(!) cardinality associated with continous observables (vectors)

HEP applications for HyperTrack – Deep Learning to Cluster

- Track reconstruction, Calorimeter object reconstruction, even both combined
- N-Pile-up decomposition: one cluster per each pp-interaction with associated final states (*or* traditional 1 hard pp + soft separation)
- Physics analysis final state clustering: group objects to access the decay (mother) of interest, jet substructure, exotic topologies (QCD sphaleron, new physics 'soft bombs' ...)

In future, multiple tasks unified under the same foundational model?

Hybrid model architecture

Per event:

$$\underbrace{\text{[Voxel-Dynamics]} \rightarrow \text{[GNN]}}_{\text{single pass}} \rightarrow \underbrace{\underbrace{\text{[Pivotal Search]} \rightarrow \text{[Set Transformer]}}_{\text{one iteration trial per cluster}}}_{\text{probabilistic extensions e.g. via normalizing flow PDFs + EM}}$$

Voxel-Dynamics

To obtain a starting graph adjacency which is sparse but informative enough for GNN+Transformer, e.g. in tracking applications[1]

(Smaller) point cloud graphs can be handled directly as fully connected ...

Learned Voxel-Dynamics

Geometry $\leftrightarrow$ Space-time:
Learn adaptive Voronoi voxelization of the detector 3D space (+1 time)

Dynamics $\leftrightarrow$ Combinatorics:
Learn target object (cluster) node combinatorial connectivity 2-point[2] C-matrix

Computationally: inference look-up acceleration is embarissingly parallel

[2] N-point constructions (tensors) possible but computationally (combinatorially) heavy

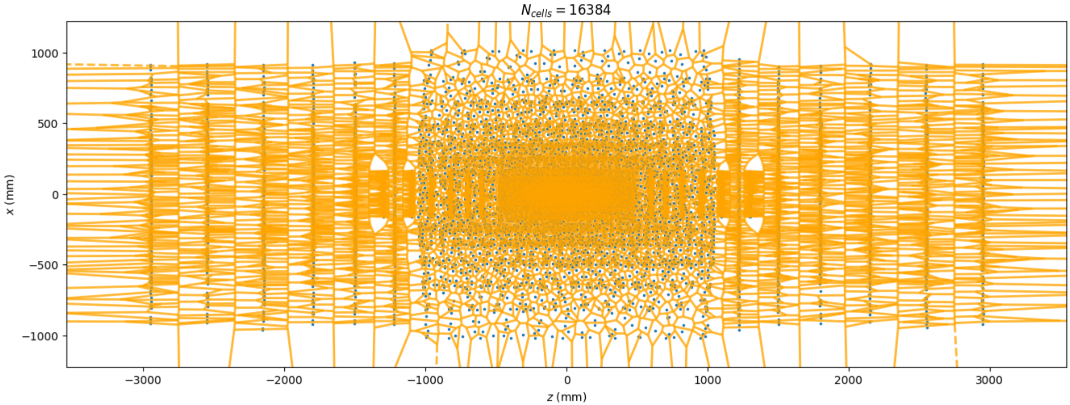


Figure: Learned Voronoi voxelization of a detector with 16384 cells (zx-projection).

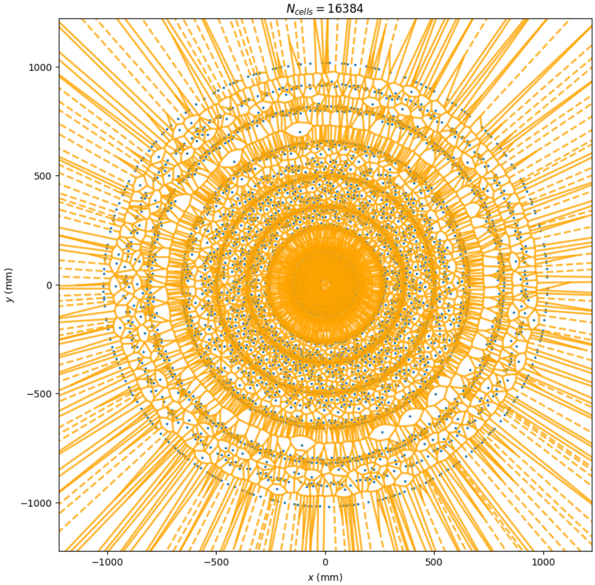


Figure: Learned Voronoi voxelization of a detector with 16384 cells (xy-projection).

C-matrix definition

Matrix elements C_{ij} encode: Given a detector hit x in 3D-space associated with i-th voxel cell, which cells $\{j\}$ could (should) it connect, given all possible track dynamics seen in the training data?

- `eom`: Equations of Motion (e.g. track helix trajectory) $\sim$ space-time local
- `cricket`: EOM + double hops
- `hyper`: Hyperedge (lasso) between all hits of the track $\sim$ space-time local + non-local (!)

Adjacency hierarchy: `eom` (most sparse) $\subset$ `cricket` $\subset$ `hyper` (most dense)[3]

[3] `hyper` (only) is strictly compatible with *HyperTrack* clustering, but `eom` can be used if the Transformer based clustering is replaced e.g. with a traditional Kalman type recursion

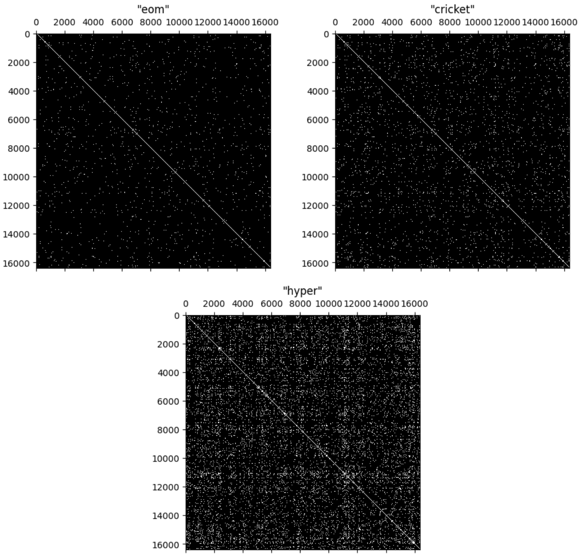


Figure: The learned C-matrix visualized for 3 different connectivity definitions.

Properties

- By construction, pile-up invariant true/false edge efficiency (ROC-pt) (but purity is not)
- Adaptive learned geometry $\rightarrow$ arbitrary detectors handled
- GPU-accelerated using Faiss library which does Voronoi voxelization via K-means and then fast inference [3D hit to cell index] look-up via accelerated geometric distance computations + fast sparsity utilizing [cell to cell] look-up for the graph adjacency based the C-matrix.
- See Appendix for performance numbers (depends on C-matrix definition)

'Multiresolution pyramid' estimator possible via multiple (course ... fine) voxelizations

Continuum version via 2-pt neural net ($R^d \times R^d \rightarrow [0,1]$) possible (SIREN + MLP)?

SuperEdgeConv GNN

HyperTrack generalization of EdgeConv [arXiv:1801.07829] (see Appendix)

The basic design idea is that GNN operates with the largest receptive field (correlations), and Transformer will operate on sub-graphs produced by GNN, taking care of clustering

GNN for latent z-representation and edge prediction

1. [GNN Message Passing over N-layers] (receptive field growth)
   (voxel-dynamic graph) $\rightarrow \{z_i^{(k)}\}_{k=1}^{N}$ (intermediate latent vectors)

2. [Latent (residual) Fusion MLP]
   $\{z_i^{(k)}\}_{k=1}^{N} \rightarrow \{z_i\}$ (final latent node vector)

3. [2-pt correlation MLP]
   $\{(z_i, z_j)\} \rightarrow \{p_{ij}\}$ (edge probability)

Meta-Learned Clustering via Set Transformer

Permutation invariant (equivariant) Encoder-Decoder Set Transformer applied iteratively on an edge sparsified (cut) event graph from GNN together with multi-pivotal seeding

For pioneering work in ML, see: Lee, Lee, Teh, *Deep Amortized Clustering* [arXiv:1909.13433]

Set Transformer

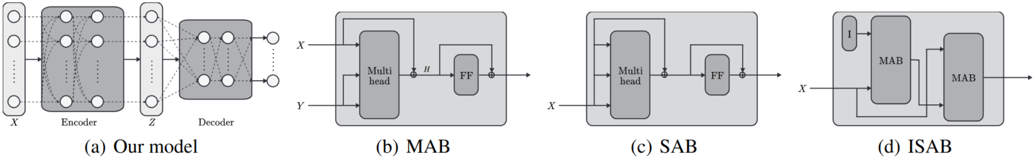


Figure 1. Diagrams of our attention-based set operations.

Figure: Diagram from *Set Transformer* paper [arxiv:1810.00825]

New in *HyperTrack*: GNN + multi-pivotal point (cluster seed) search mechanics and trial logic + adaptive thresholding + new hybrid loss function

(See Appendix for details)

Clustering Mechanics

Input: Edge sparsified 'cut graph' from GNN (based on 2-point probabilities) $\rightarrow$ graph cut yields disconnected subgraphs (proto-clusters) [allows parallelization]

Iterate (loop)

- Greedy or Monte Carlo search walk on the subgraphs $\rightarrow$ Find strongly connected 'pivotal' graph nodes based on edge probabilities, then connect their joint (inclusive) micrograph
- Set Transformer module takes in micrograph and pivotal indices, in GNN latent z-encoding per node + raw input (e.g. 3D hit) and gives a scalar output for each graph node
- Threshold cut on output (fixed or adaptive via min 2-class intra-class variance aka 1D Fisher / Otsu rule)

Output: Event-by-event, variable cardinality set of clusters each with associated graph nodes (hits)

Hybrid end-to-end loss, $\mathcal{L} = \sum_i \beta_i \mathcal{L}_i$

1. Edge Binary Cross Entropy loss
2. Edge (node) contrastive[4] loss
3. Cluster Binary Cross Entropy loss [meta-supervised]
4. Cluster "contrastive"[5] loss [meta-supervised]

Meta-supervision $\sim$ the clustering procedure training has supervised (label) information about which graph nodes correspond to which ground truth cluster, but the meta-loop itself needs to make a decision which ground truth cluster to consider ($\sim$ "Wheelerism") $\rightarrow$ majority vote.

---

[4] Distinguish right/wrong connected nodes per ground truth cluster, see contrastive learning
[5] Not contrastive in representation learning literature sense, but a set intersect score type

Gradient flow example

$\sim$ 100 neural sub-modules $\sim$ 3.1 million parameters

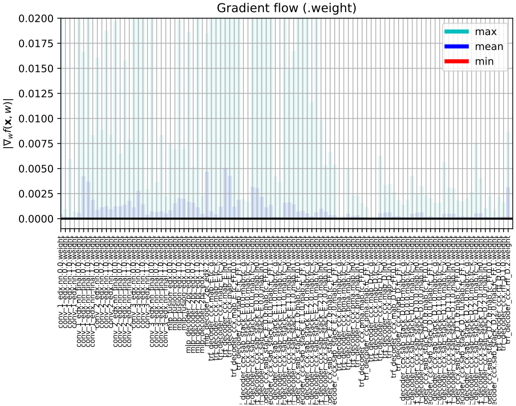


Figure: Network weight gradient component absolute values per neural module. x-axis left to right: GNN $\sim (1/3)$ ... Transformer $\sim (2/3)$ fraction of modules.

Proof-of-Concept

Track ($\sim$ cluster) reconstruction benchmark, a challenging but controllable problem

Track Reconstruction

- TrackML dataset: Pythia $t\bar{t}$ + pp-minimum bias, $dN_{ch}/d\eta \sim 7$, ACTS detector simulation, $\eta \in [-4, 4]$, pile-up $\langle\mu\rangle = 200$
- Here, reduced pile-up $\langle\mu\rangle$ to 2 and 20 $\rightarrow$ approximately 100 and 1000 clusters (tracks) per event, graph nodes (hits) on average 10$\times$ number of tracks. Also, we reduced the pure noise hit fraction ($\sim 15 \rightarrow 5\%$). No (unphysical) gen-level minimum p_T or other cuts applied.
- Only 3D hit information used (not e.g. charge deposits, detector modules)
- High-level Python implementation (torch, torch-geometric, numpy, numba-JIT)
- Performance comparisons based on Double Majority Score [DMS], a set intersection measure described in TrackML challenge, with minimum 4 hits per ground truth cluster

Training

- Single NVidia V100-32 GB VRAM + around 25 GB CPU RAM $\rightarrow$ current unoptimized model/code limit around $\mu \simeq 30$ $\rightarrow$ training time some days to a few weeks
- Terminology here: 1 training iteration $\sim$ 1 gradient pass
- N.B. Training is still on-going on the server for $\mu \simeq 20$ (still improving)
- *No systematic hyperparameter tuning* is done (model layer design, algorithmic thresholds or training scheme and its parameters) $\rightarrow$ will be done on a GPU cluster

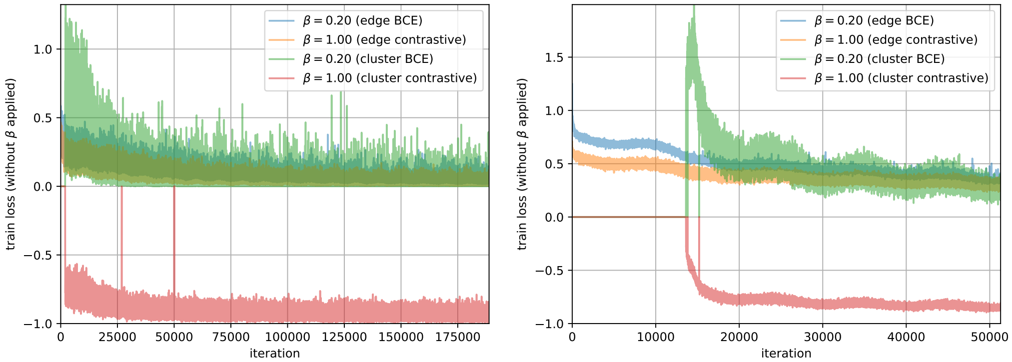


Figure: **Loss function** evolution for pile-up $\mu \simeq 2$ and 20 (left and right). Cosine scheduler oscillations clearly visible. The few spikes are due to the model save-reload code (fixed).

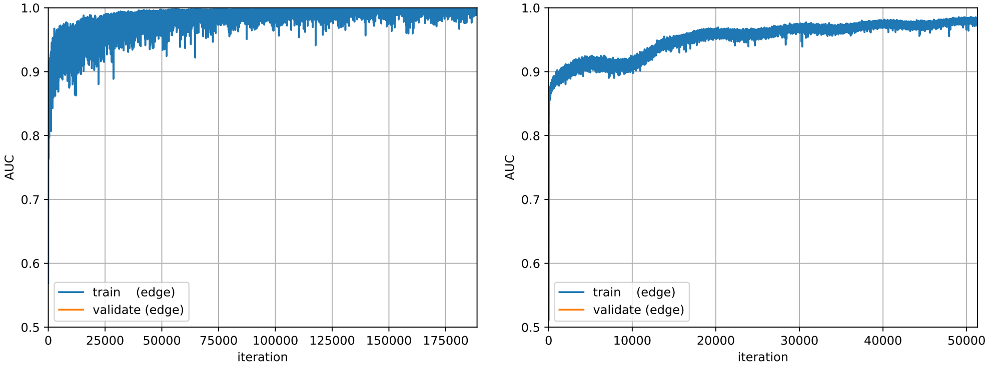


Figure: **Edge prediction AUC** evolution for pile-up $\mu \simeq 2$ and 20 (left and right).

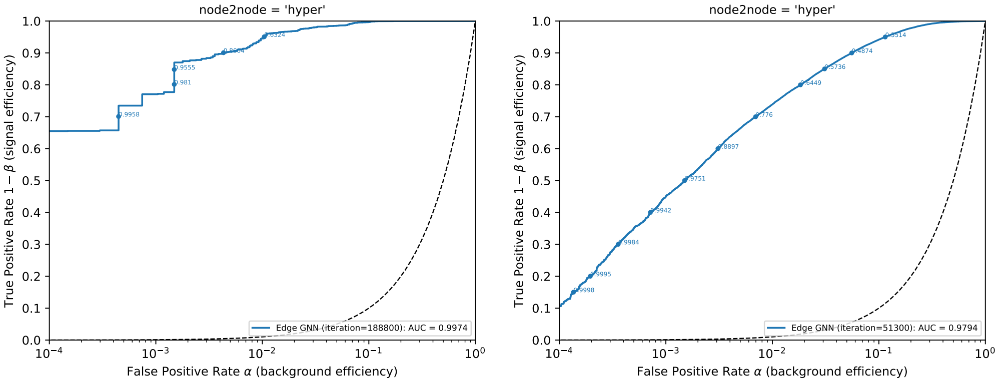


Figure: **Edge prediction ROC** for pile-up $\mu \simeq 2$ and 20 (left and right).

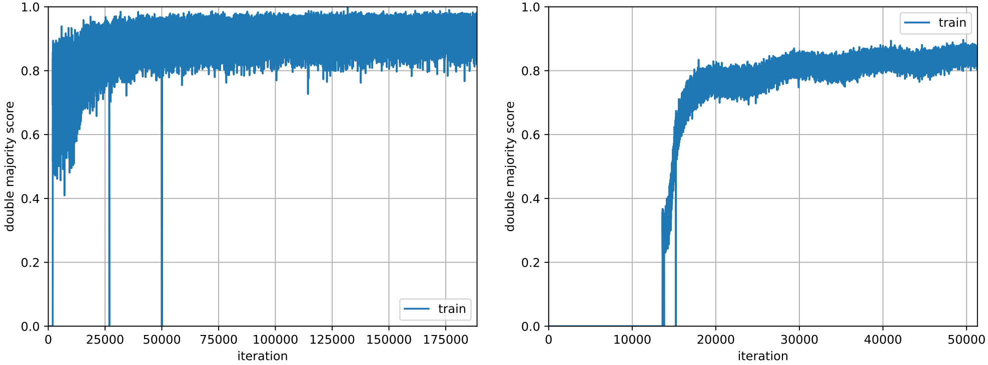


Figure: Clustering **Double Majority Score** (DMS) evolution for pile-up $\mu \simeq 2$ and 20 (left and right).

Physics inference performance

Efficiency is defined as *HyperTrack*/MC(nhits $\geq$ 4), i.e. minimally reconstructable/feasible tracks matched with Double Majority Score (DMS)

Track parameter fitting based on the clustering output is a next-step problem and not done here – classic (recursive, robust fitting ...) or neural solutions can be applied (regression GNN+MLP, normalizing flow PDF based ... $\rightarrow$ perhaps in next version of *HyperTrack*)

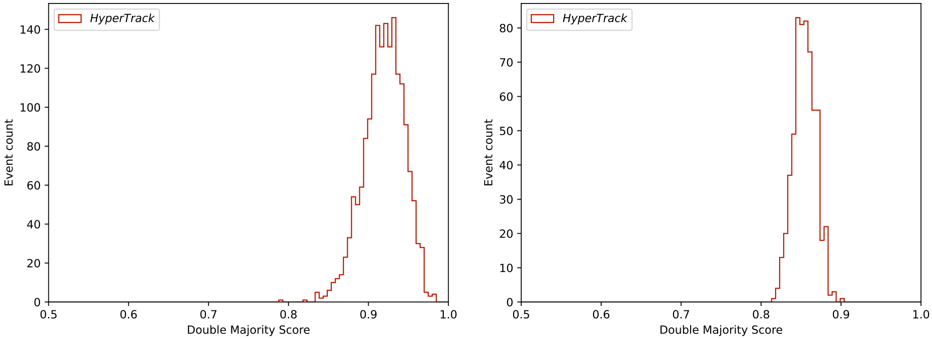


Figure: Per event **Double Majority Score** ($\sim$ overall efficiency) for pile-up $\mu \simeq 2$ and 20 (left and right).

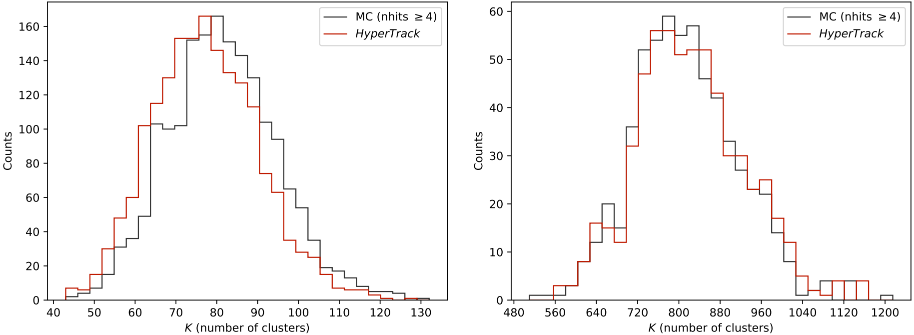
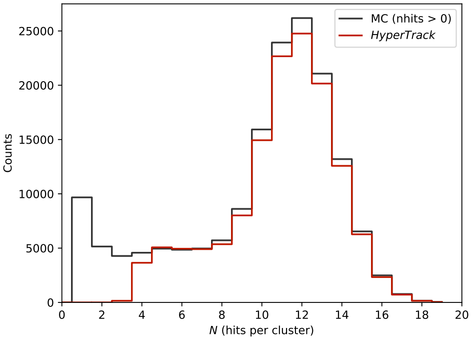


Figure: **Per event raw cluster multiplicity** for pile-up $\mu \simeq 2$ and 20 (left and right). *HyperTrack* learns this implicitly – the number of clusters is not a parameter of the algorithm.



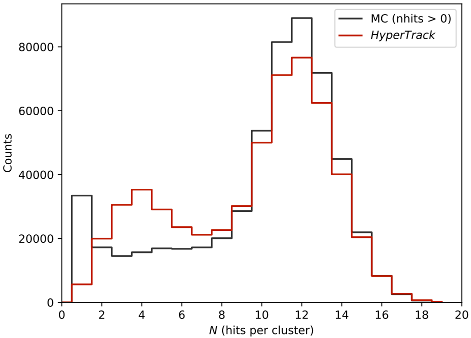


Figure: **Raw hit multiplicity per cluster** for pile-up $\mu \simeq 2$ and 20 (left and right).

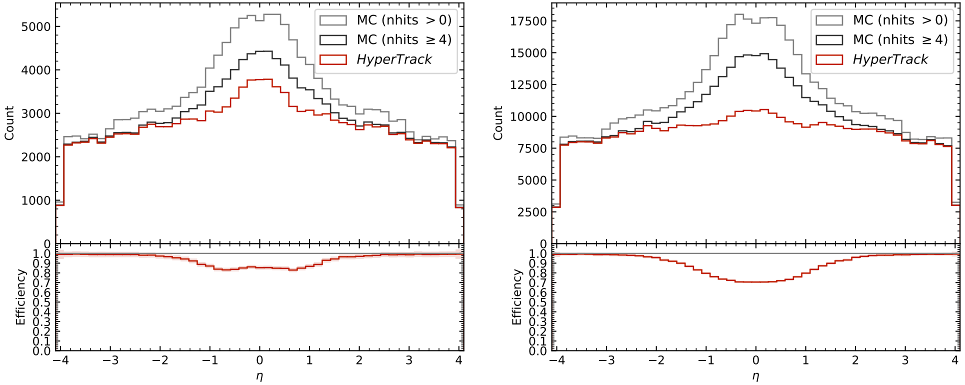


Figure: DMS matched track **pseudorapidity** for pile-up $\mu \simeq 2$ and 20 (left and right).

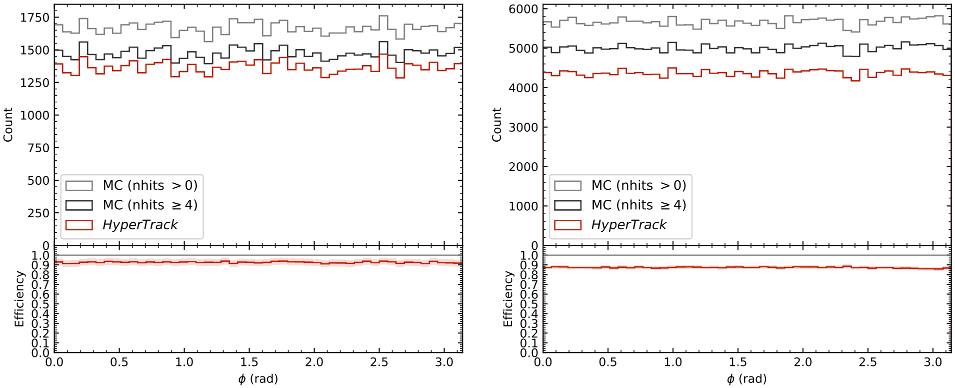
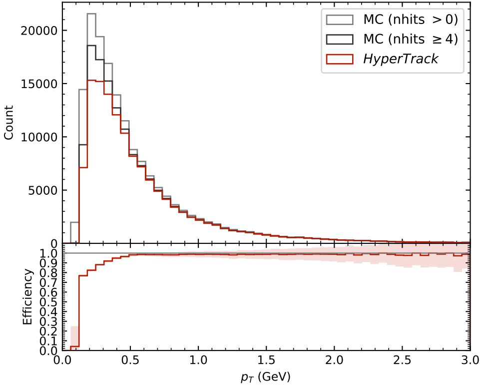


Figure: DMS matched track **atzimuthal angle** for pile-up $\mu \simeq 2$ and 20 (left and right). Rotationally symmetric so OK (diagnostics).



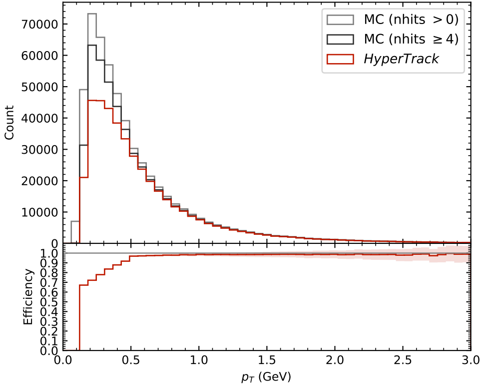


Figure: DMS matched track **transverse momentum** for pile-up $\mu \simeq 2$ and 20 (left and right). Low p_T is physically hard, as usual.

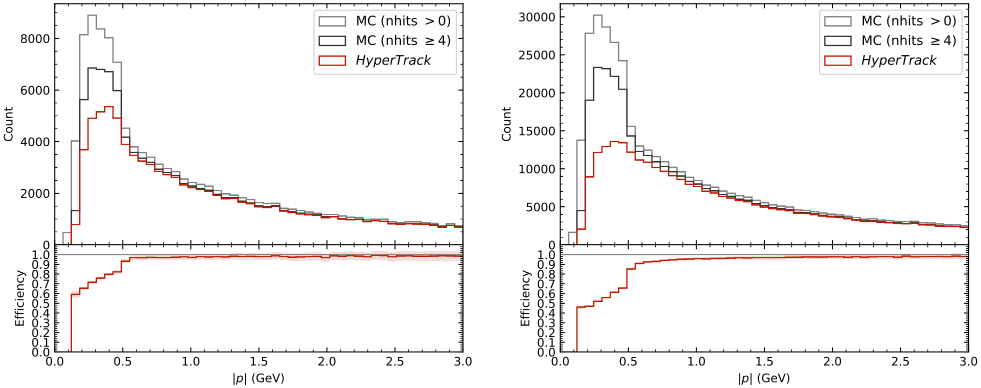


Figure: DMS matched track **3-momentum norm** for pile-up $\mu \simeq 2$ and 20 (left and right).

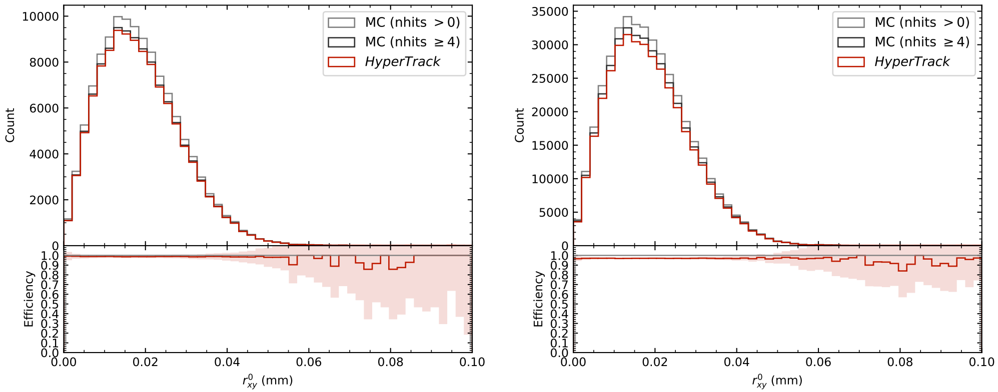


Figure: DMS matched track **vertex transverse displacement** [0, 0.1] mm for pile-up $\mu \simeq 2$ and 20 (left and right). Very high performance for non-displaced tracks (but this is naturally integrated over kinematics).

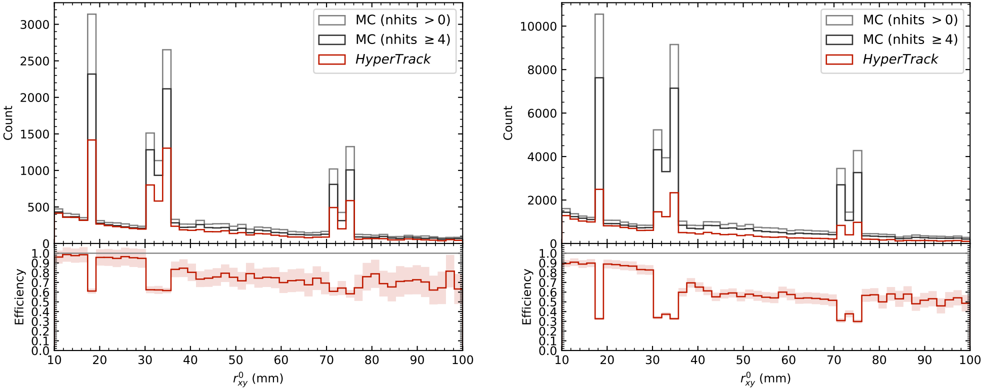


Figure: DMS matched track **vertex transverse displacement** [10, 100] mm for pile-up $\mu \simeq 2$ and 20 (left and right). Distribution peaks are (presumably) material secondaries (gamma conversion $\gamma \to e^+e^-$ in the detector layers) – ACTS detector simulation.

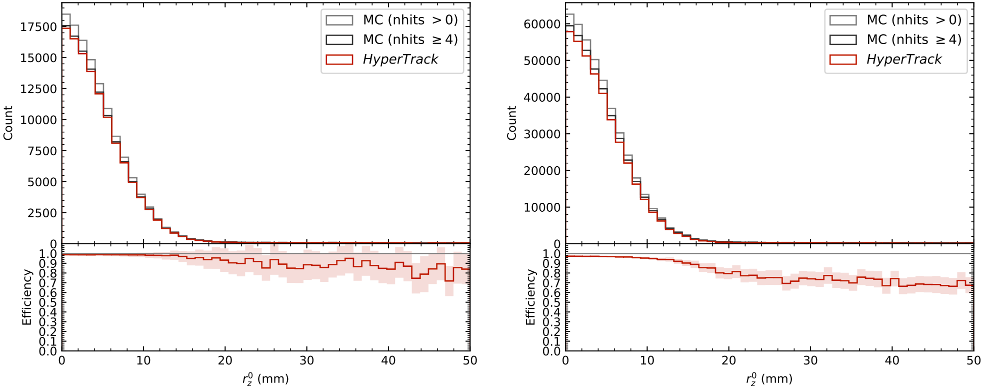


Figure: DMS matched **track vertex longitudinal displacement** for pile-up $\mu \simeq 2$ and 20 (left and right).

Open Source

github.com/mieskolainen/hypertrack (MIT license), *to be available*

All inclusive

TrackML dataset processing, geometric and graph processing tools, torch-based model definitions, training code, inference code and performance plots

+ docs

Hybrid Quantum Computing?

Ideally: each graph node can belong to any cluster in superposition. Read in classical information, prepare the quantum state, do the measurement (circuit read-out) $\rightarrow$ each node collapses into one of the clusters $\rightarrow$ repeat measurements $\rightarrow$ get a combinatorial assignment probability distribution and trace it.

Quantum ML/AI models?

GNN: Message Passing $\rightarrow$ Quantum ML Walk on a graph [arXiv:2302.00892]
Transformer: Classical attention + Query-Key-Value logic $\rightarrow$ unitary gate quantum version [arXiv:2209.08167]

Why?

1. Theoretically interesting and hopefully quantum speed up (one day ...)
2. Quantum representation perhaps more expressive (c.f. Weisfeiler-Lehman test type GNN limitations, oversmoothing for deep GNNs ...)

Future

- Technical (speed, architecture, mixed precision, scale up)
- Math fundamentals (probabilistic, more general representations beyond graphs: true hypergraphs, matroids ...)
- Domain adaptation / transfer learning (adapt against real data)
- Benchmark various HEP applications, especially very complicated clustering!
- Self-supervision $\rightarrow$ *HyperTrack* + anomaly detection?

Summary

Introduced a new neural combinatorial algorithm, HyperTrack – learning to cluster, fundamentally 'AI-driven' approach for HEP reconstruction challenges

- Based on the tracking proof-of-concept, scaling up seems to be mostly limited by computational and timing constraints (not by learning capability)
- No hand built track dynamics was used or utilized, all machine learned $\rightarrow$ generic applications such calorimetry or QCD phenomenology / new physics searches
- Simply increasing GNN + Transformer layers and latent dimension may allow extremely complicated combinatorics, way beyond any hand-engineered clustering approaches

Acknowledgements

Alex Tapper, Liv Vage, Simon Williams for discussions

Appendix

Voxelized Adjacency, $\mu \simeq 2$, *'hyper'*

```
hypertrack.trackml.reduce_tracks: Input (output) tracks = 9077 (84) [0.009]
func:reduce_tracks took: 0.0118 sec
hypertrack.tools.compute_ground_truth_A: Found 42 (4.762E-02) unassociated hits (self_connect_noise = False)
func:compute_ground_truth_A took: 0.0621 sec
hypertrack.predictors.voronoi_predictor: Loaded a model with ncell = 131072] | node2node = hyper
hypertrack.predictors.compute_cell_based_adj: Geometric index search (0.03139 sec) | Adjacency construction (0.00998 sec)
hypertrack.tools.print_graph_metrics:

Ground Truth Adjacency (A)
 Nodes N                    = 882
 Positive edges POS         = 9496
 Negative edges NEG         = 768428
 POS / NEG                  = 1.24E-02

Estimate (A_hat)
 True Positive  TP          = 9066
 True Negative  TN          = 763094
 False Positive FP          = 5334
 False Negative FN          = 430
 Accuracy                   = 0.9926    (TP + TN) / (POS + NEG)
 Purity                     = 0.6296    TP / (TP + FP)
 True  Positive Efficiency  = 0.9547    TP / POS = TP / (TP + FN)
 False Positive Efficiency  = 0.0069    FP / NEG = FP / (FP + TN)

Edge count
 |A|            = 9496
 |A_hat|        = 14400
 |A_hat| / |A|  = 1.4E+04 / 9.5E+03 = 1.52
 |A_hat| / N^2  = 1.4E+04 / 7.8E+05 = 1.85E-02
 |A| / N^2      = 9.5E+03 / 7.8E+05 = 1.22E-02
```

Voxelized Adjacency, $\mu \simeq 20$, *'hyper'*

```
hypertrack.trackml.reduce_tracks: Input (output) tracks = 9732 (1013) [0.104]
func:reduce_tracks took: 0.0342 sec
hypertrack.tools.compute_ground_truth_A: Found 498 (4.760E-02) unassociated hits (self_connect_noise = False)
func:compute_ground_truth_A took: 0.1611 sec
hypertrack.predictors.voronoi_predictor: Loaded a model with ncell = 131072] | node2node = hyper
hypertrack.predictors.compute_cell_based_adj: Geometric index search (0.0543 sec) | Adjacency construction (0.2181 sec)
hypertrack.tools.print_graph_metrics:

Ground Truth Adjacency (A)
 Nodes N                    = 10462
 Positive edges POS         = 114594
 Negative edges NEG         = 109338850
 POS / NEG                  = 1.05E-03

Estimate (A_hat)
 True Positive   TP         = 108614
 True Negative   TN         = 108530578
 False Positive FP          = 808272
 False Negative FN          = 5980
 Accuracy                   = 0.9926    (TP + TN) / (POS + NEG)
 Purity                     = 0.1185    TP / (TP + FP)
 True  Positive Efficiency = 0.9478    TP / POS = TP / (TP + FN)
 False Positive Efficiency = 0.0074    FP / NEG = FP / (FP + TN)

Edge count
 |A|            = 114594
 |A_hat|        = 916886
 |A_hat| / |A| = 9.2E+05 / 1.1E+05 = 8.00
 |A_hat| / N^2 = 9.2E+05 / 1.1E+08 = 8.38E-03
 |A| / N^2      = 1.1E+05 / 1.1E+08 = 1.05E-03
```

Voxelized Adjacency, $\mu \simeq 100$, 'hyper'

```
hypertrack.trackml.reduce_tracks: Input (output) tracks = 8922 (4537) [0.509]
func:reduce_tracks took: 0.1163 sec
hypertrack.tools.compute_ground_truth_A: Found 2243 (4.761E-02) unassociated hits (self_connect_noise = False)
func:compute_ground_truth_A took: 0.5900 sec
hypertrack.predictors.voronoi_predictor: Loaded a model with ncell = 131072] | node2node = hyper
hypertrack.predictors.compute_cell_based_adj: Geometric index search (0.1398 sec) | Adjacency construction (2.621 sec)
hypertrack.tools.print_graph_metrics:

Ground Truth Adjacency (A)
 Nodes N                    = 47111
 Positive edges POS         = 517210
 Negative edges NEG         = 2218929111
 POS / NEG                  = 2.33E-04

Estimate (A_hat)
 True Positive  TP          = 489866
 True Negative  TN          = 2202973796
 False Positive FP          = 15955315
 False Negative FN          = 27344
 Accuracy                   = 0.9928    (TP + TN) / (POS + NEG)
 Purity                     = 0.0298    TP / (TP + FP)
 True  Positive Efficiency  = 0.9471    TP / POS = TP / (TP + FN)
 False Positive Efficiency  = 0.0072    FP / NEG = FP / (FP + TN)

Edge count
 |A|            = 517210
 |A_hat|        = 16445181
 |A_hat| / |A|  = 1.6E+07 / 5.2E+05 = 31.80
 |A_hat| / N^2  = 1.6E+07 / 2.2E+09 = 7.41E-03
 |A| / N^2      = 5.2E+05 / 2.2E+09 = 2.33E-04
```

SuperEdgeConv GNN architecture 1/2

Message Passing + inner MLP (k-th layer)

$$z_i^{(k)} = \boxplus_{j \in \mathcal{N}_i} MLP_{MP}^{(k)}([x_i, x_i - x_j, x_i \odot x_j, e_{ij}]), \qquad (1)$$

- Both additive and multiplicative operations, and the graph neighborhood $\mathcal{N}_i$ accumulator $\boxplus$ takes vector mean (can be changed to max, attention ... based)
- Edge features e_{ij} computed as difference between the node vertex degrees $(d_i - d_j)/\langle d \rangle \to$ helps to resolve certain graph ambiguities
- Some applications may benefit (heavily) from edge features such as Lorentz invariants $s = (p_i + p_j)^2, t = (p_i - p_j)^2$, c.f. invariant/equivariant architectures (see applications in *ICENET*)

SuperEdgeConv GNN architecture 2/2

Residual layer fusion MLP

$$z = MLP_F(cat[z^{(1)}, z^{(2)}, \dots, z^{(k)}]) \tag{2}$$

Requires intermediate memory, but can be critical for learning.

2-point correlation MLP

$$p_{ij} = MLP_C(z_i \odot z_j) \in [0, 1] \tag{3}$$

Multiplicative ($i \leftrightarrow j$ symmetric) input operator. Other options also implemented.

Set Transformer Architecture

Input: $Z \sim$ graph node vectors in GNN z-repr. + 3D-hits concatenated

Encoder: Attention wrt pivotal nodes and self-attention

$$H_Z = SAB_E^{stack}(MAB_E(Q = Z, K = Z[\text{pivot indices}])) \tag{4}$$

Pooling: Adaptive via PMA

$$H_\theta = PMA(H_Z) \tag{5}$$

Decoder: Attention wrt pooled representation and self-attention

$$H_m = SAB_D^{stack}(MAB_D(Q = H_Z, K = H_\theta)) \tag{6}$$

Mask decoder:

$$m = MLP_D(H_m) \tag{7}$$

Output: Scalar $\in [0, 1]$ per graph node

Object flow

Point cloud [Voxel-Dynamics input]
$\rightarrow$ Starting graph adjacency with point cloud data [GNN input]
$\rightarrow$ GNN probability sparsified (cut) event graph
$\sim$ { Disconnected sub-graphs } [Pivotal search input]
$\rightarrow$ { Fully connected micro-graphs } [Transformer input]
$\Rightarrow$ { clusters with associated hits } [Final output]

Overall model details

- Voxel-Dynamics is based on $2^{17} = 131072$ cells $\rightarrow$ 17 billion C-matrix elements
- Neural model contains approximately 3 million parameters for GNN ($\sim$ 1M) + Transformer ($\sim$ 2M)
- 3 GNN layers, 3 Set Transformer self-attention (SAB) layers for encoder and decoder (with number of multihead=4), around 3 layers per MLP (several)
- Increasing GNN and Transformer layers increases 'receptive field' $\rightarrow$ larger number of graph node multi-point combinations and correlations considered
- Latent z-representation dimension $\sim$ 200, larger may be needed e.g. for higher pile-up

Loss definitions

1. Edge Binary Cross Entropy loss is between GNN prediction $\hat{p} \in [0, 1]$ and the ground truth edge label $p \in \{0, 1\}$, as encoded by the chosen ground truth adjacency (e.g. `hyper` definition).
2. Contrastive edge loss is a loop over ground truth clusters (particles), with each associated positive $\hat{p}_+$ and negative $\hat{p}_-$ pointing edge (node) connection collected and finally a softmax type contrastive loss computed between $\hat{p}_+$ and $\hat{p}_-$.
3. Cluster Binary Cross Entropy loss is between the transformer output $\hat{m} \in [0, 1]$ per node vs ground truth node label $m \in \{0, 1\}$, with a meta-supervision chosen majority vote ground truth cluster.
4. Cluster set score loss computes an intersection set between the hard thresholded estimates Thresh[$\hat{m}$] and the ground truth m. Then a cluster local sum over non-thresholded values $\hat{m}$ is taken over this set.

Training details 1/2

- 3 disjoint dataset splits: A. Voxel-Dynamic (VD) train (crucially not the same as for neural), B. Neural model train, C. Inference evaluation
- VD train [1400 events $\sim$ 13(130) million tracks (hits)], Neural [3000 events]
- Random combinatorial resampling of tracks in pile-up reduction $\rightarrow$ for $\mu \simeq 20$ this gives $C_k(n) = (10000, 1000) = 8.7 \times 10^{1409}$ combinatorial variations per event
- AdamW gradient descent, weight decay (reg.) 10^{-5}, oscillating cosine scheduler with lr$=10^{-4} \ldots 10^{-5}$ (could be changed in the late train phase to pure decay)

Training details 2/2

- GNN is first trained above an AUC threshold (0.95), then Transformer training is activated end-to-end (computational speed up)
- Batch size = 1, i.e. neural weights updated after every event. Batch size can be increased for the low pile-up case (to balance/optimize the gradient noise) (VRAM limited)
- VD training and ROC point is pile-up invariant, neural model can be trained to be a pile-up generalist by sliding the μ-value between a large range during the training

Inference time

$\mu \simeq 2$:
VD index search: 0.05 sec | VD adjacency: 0.05 sec | GNN: 0.02 sec | Clustering (Pivotal search + Transformer) loop: 1 sec (0.01 sec per cluster)

$\mu \simeq 20$:
VD index search: 0.05 sec | VD adjacency: 0.25 sec | GNN: 0.1 sec | Clustering (Pivotal search + Transformer) loop: 10 sec (0.01 sec per cluster)

N.B. For acceleration, clustering search loop can be parallelized e.g. with libtorch C++ implementation and finally Transformer input can be tensorized