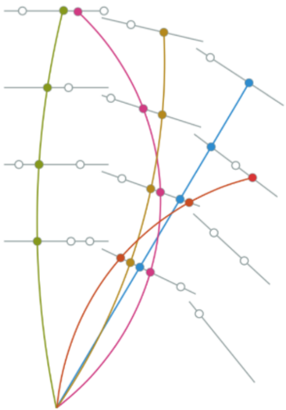
# Machine Learning for Track Reconstruction at the LHC

Louis-Guillaume Gagnon (UC Berkeley)

CoDaS-HEP 2024
2024/07/26

# Introduction: Particle Trajectory Reconstruction

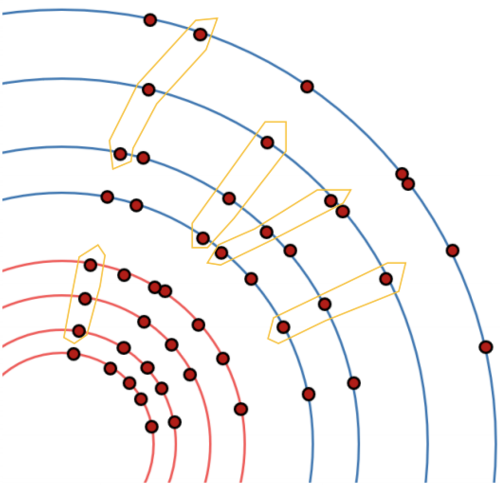
- Particle trajectory reconstruction (Tracking) is a **clustering** problem
- Input: Set of points in 3D space (Hits)
- Output: Set of sets of points each set corresponding to a single particle
- Total N. of hits $\gg$ N. of hits in one track $\implies$ very challenging!
- Typical algorithm: Kalman Filter (KF)
  - ☺Physics performance is excellent
  - ☹Runtime scales badly with $N_{\mathrm{hits}}$



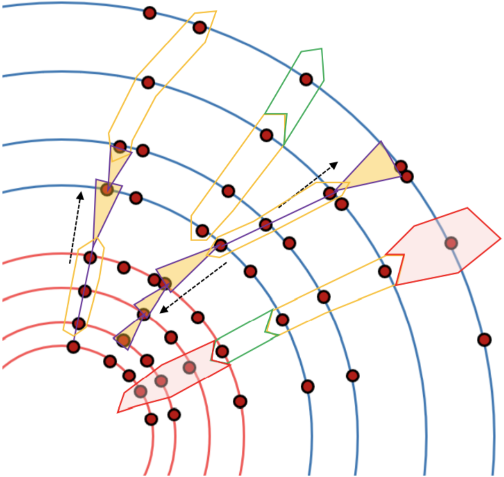
[1904.06778]

# Introduction: Track Reconstruction Stages

- Seed Finding...

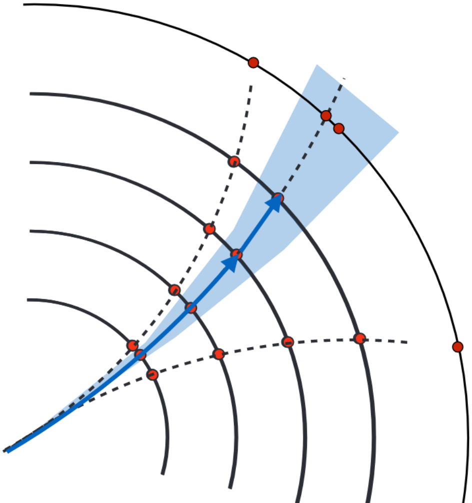


- ... Track Finding

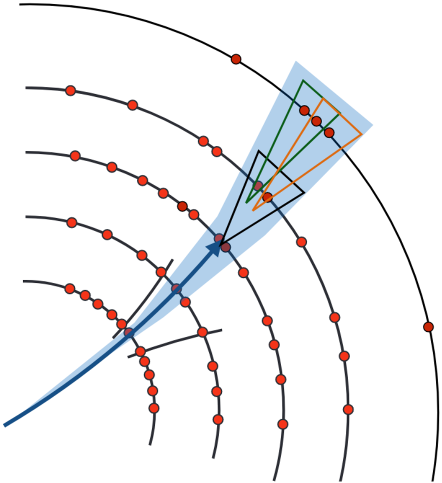


# Introduction: Kalman Filter, the "classical" approach

- Kalman Filtering: finding the "best fit" track from a seed



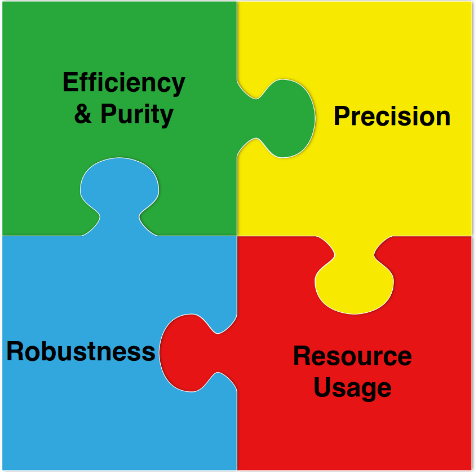
- Combinatorial Kalman Filtering: Find ≈ all good track candidates



# Why ML?

## What do we need most?

- High track finding efficiency
- Low number of combinatoric fakes
- Performance is stable under realistic conditions: alignment, ageing, calibration




- Correct determination of track parameters
- Computing time, memory usage

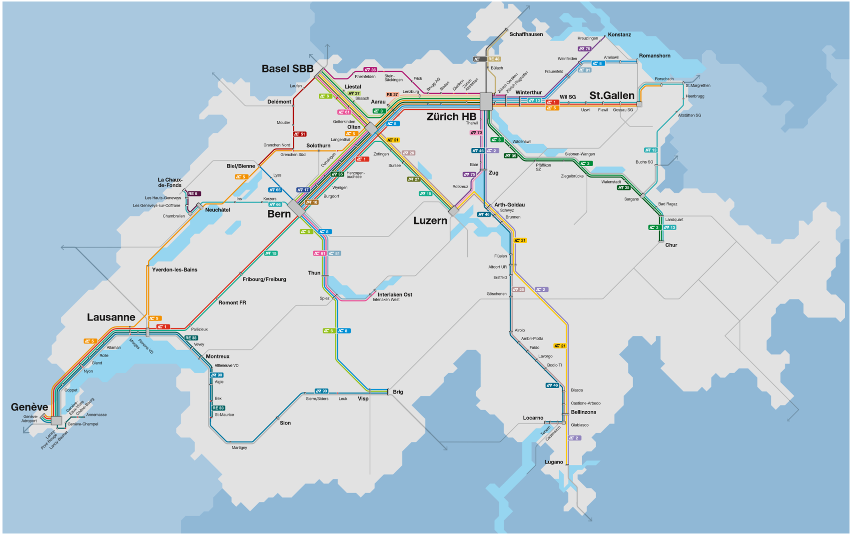
Credit: Christian Grefe

# Why ML?

- Important tradeoff between track-finding **Efficiency**, **Fake rate**, and **Resource consumption**
- Current ATLAS tracking pipeline clearly shows this:
  - "Loose" track seeding stage to initialize KF-based track finding
  - With enough starting seeds, KF finds most particles of interest ...
  - ...along with lots of fake tracks ...
  - ...which necessitates an ambiguity resolution stage.
- All of this compounds into high resource consumption!
- Important: It's not only a computational issue!
  - Keeping the combinatorics in control require setting "fiducial cuts" on particles to reconstruct
  - E.g. $pT$ threshold, Impact parameter ranges, N. of Si hits, ...
- More **computationally efficient** algorithms are needed!
  - Better use of constrained resources
  - Allow widening the space of tracks we attempt to reconstruct
- ML solutions are obvious candidates!

# I. Track finding with graph neural networks

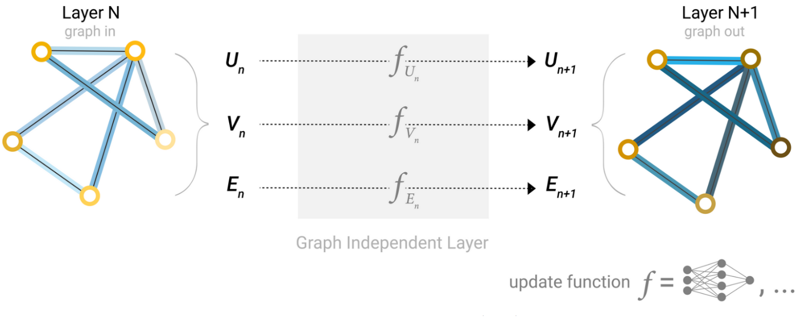
# What is a graph?




Graph $(U)$, Edges $(E)$, Vertices $(V)$

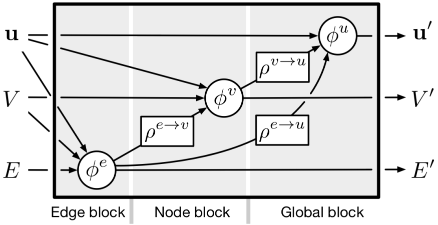
# What is a graph neural network (GNN)?

- Simplest possible GNN (source: [distill.pub]())

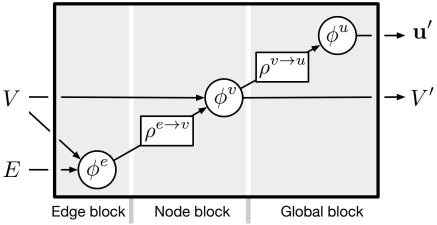


# What is a graph neural network (GNN)?

- Can model arbitrarily complex relationships . . .



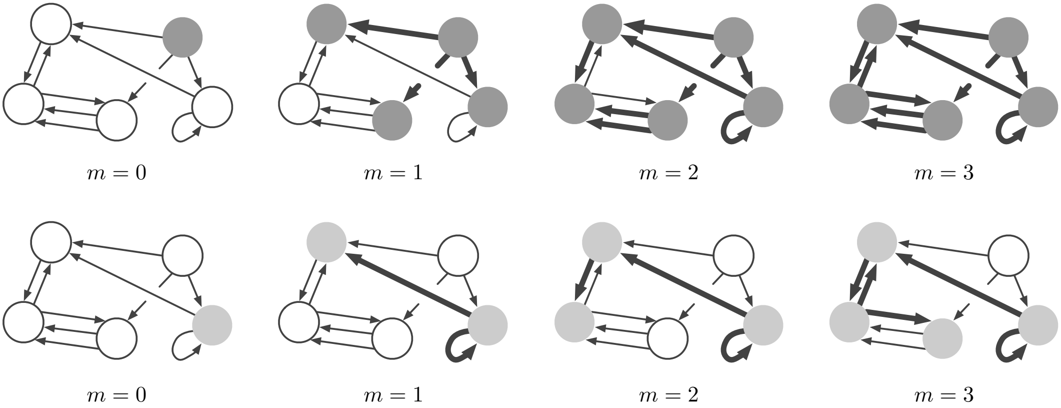

- . . . or slightly simpler ones!




source: [1806.01261]

# "Message passing"

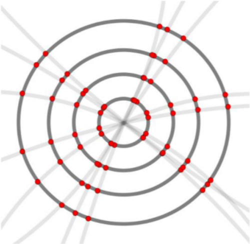
- Graph neural networks model relationships between *adjacent* nodes
- Stacking (or iterating) many $G_n \to G_{n+1}$ blocks allows information to diffuse through network



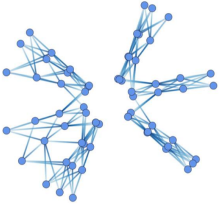

source: [1806.01261]

# ATLAS GNN4ITk

## Our graph definition



Hits

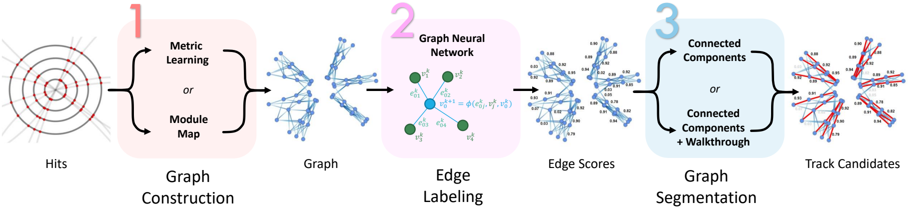


Graph

credit: Heberth Torres, CTD23

- GNN4ITk: R&D project within ATLAS tracking group
- Vertices: 3-D “space-points” (aka “Hits”)
- Edges: Probability that any two space-points are originating from same particle

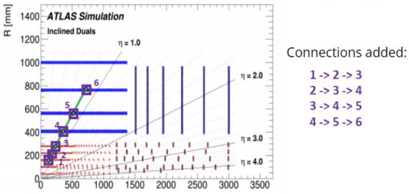
# GNN-based tracking pipeline




- As in "classical" case, pipeline has multiple steps
  1. Need intelligent graph-building stage: too many hits to enumerate all possible connections!
  2. GNN does the edge-scoring task
  3. Create actual tracks with simple graph-walking algorithm

# Step 1: Graph construction

- First approach: The *Module Map*
- In a nutshell:
  - Using a simulated sample, enumerate triplets of hits from single particles
  - If triplet pass certain kinematic requirements: record connection between modules
  - When constructing the graph: only allow connections found in module map

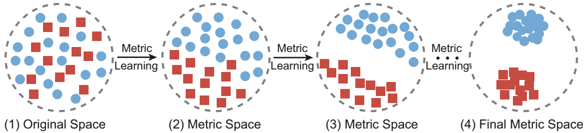



credit: Minh-Tuan Pham

- Approach is "brute-force"-like, but only need to create the map once!

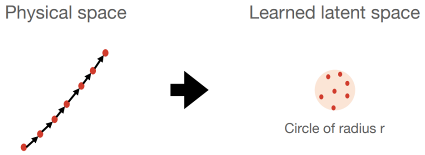
# Step 1: Graph construction

- Second approach: *Metric Learning*
- Metric space ≡ Set with a definition of distance between its elements
- E.g. euclidian space, (aka "physical" space)
- Can train ML model to learn new metrics by minimizing a suitable distance definition




credit: [1805.05510]

- Application to graph construction: Only allow edges if distance in learned space is small

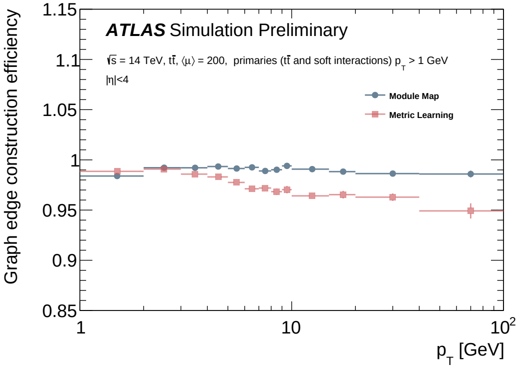



credit: Heberth Torres, CTD23

- Can metric learning be used to perform the track finding stage itself? More on that later!

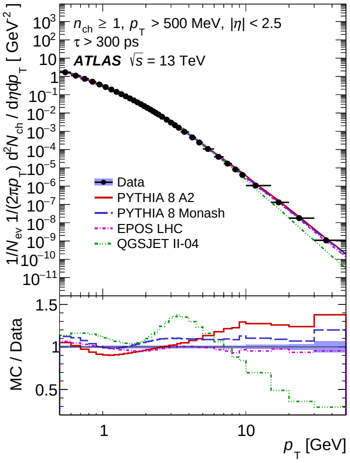
# Step 1: Graph construction: Module map vs metric learning

- Metric learning underperforms at higher $pT$
- Higher $pT \implies$ straighter tracks
- ... Metric gets harder to learn?

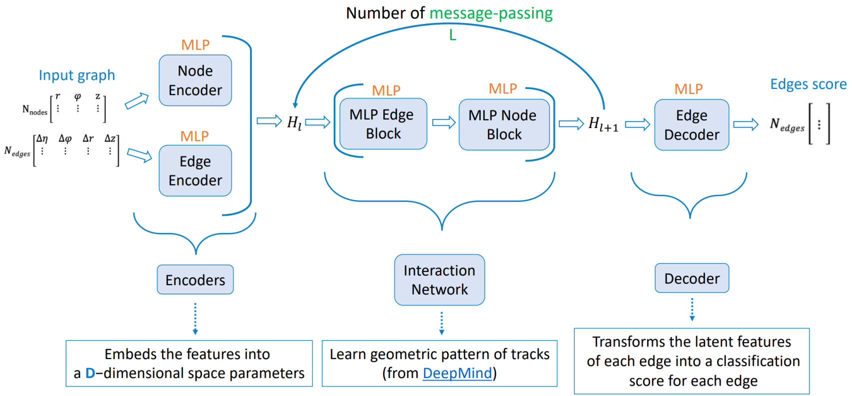



- GNN4ITk currently use module map approach

- It could also be a statistics issue: exponentially less tracks at high pT!



# Step 2: Edge labeling

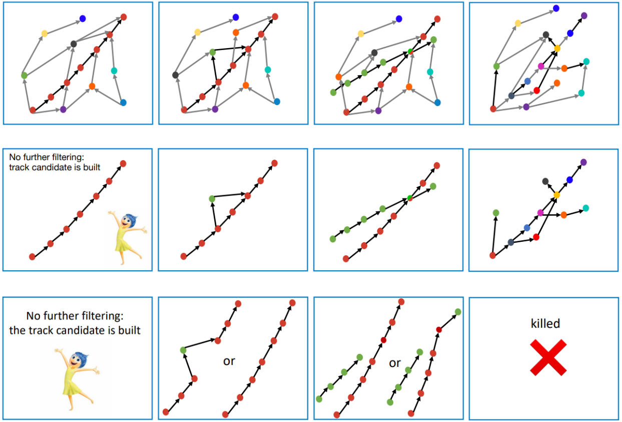



credit: Charline Rougier, CTD22

- Interaction network paper

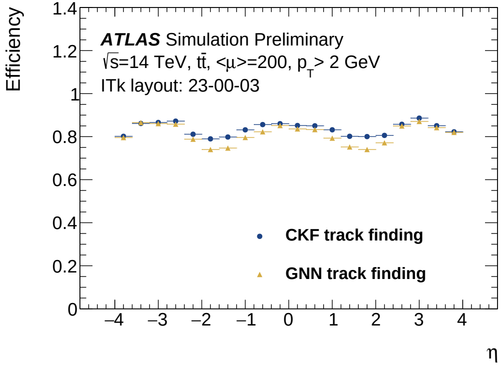
# Step 3: Graph segmentation

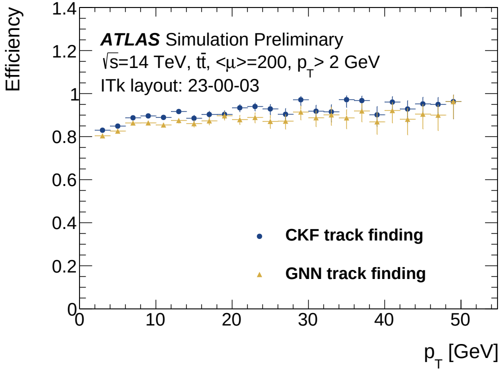
- Form tracks in 2 steps:
  1. Find connections with loose cut
  2. Find paths with tighter cut




credit: Charline Rougier, CTD22

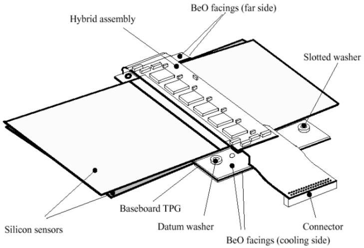
# Putting it all together: Tracking efficiency

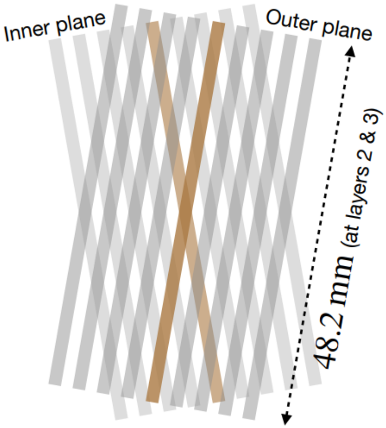



- Performance approaching that of Kalman Filter-based pipeline
- But still falls a few percent short: **why?**

# A word on strip space-points

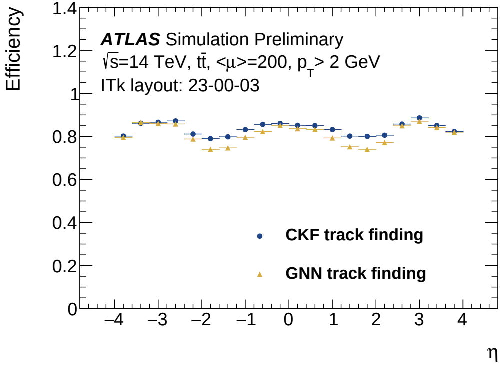


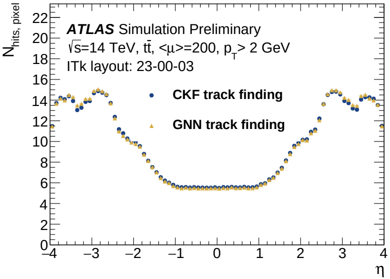



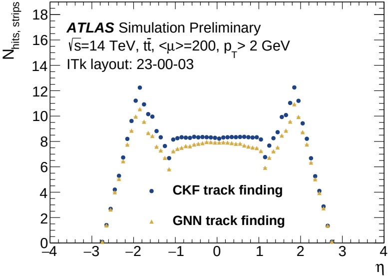

credit: Heberth Torres, CTD23

- Strip space-points are created from stereo-pair of 1-D measurements: Precision is less than for pixel space-points!

# Putting it all together: Track content



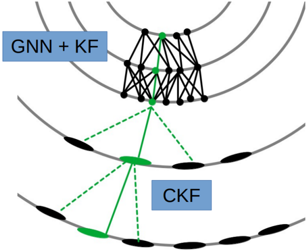




- Part of the answer: missing strip measurements?
- Behavior aligns well with the $\eta$ plot as well!
- Currently under investigation

# Hybrid models?

## Idea

- **GNN:**
  - Resolve combinatorics with high resolution spacepoints in pixels
  - Use ordinary KF here
- **CKF**
  - Completes tracks in strips
- **Benefits of combination:**
  - High quality seeds without duplicates for CKF
  - Use CKF in region with lower density (→ less branching)
  - CKF can e.g. use single strip measurements
  - Smaller graph (pixel only)



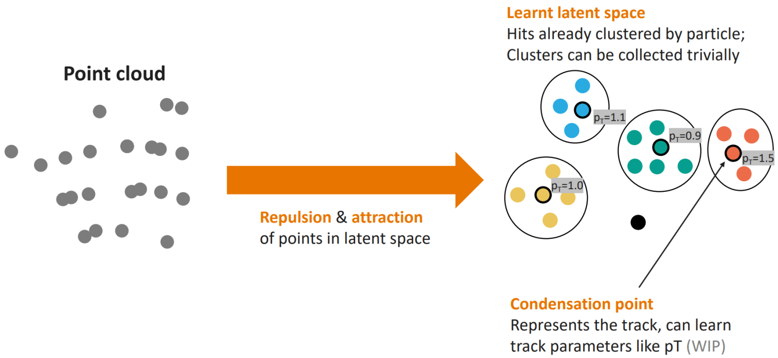

*„Full GNN pixel seeding + CKF"*

credit: Benjamin Huth, CTD23

- Project within ACTS to combine GNN & KF pipelines
- Sill in early WIP status!

# II. Track finding with metric learning

# Tracking with metric learning: Object Condensation

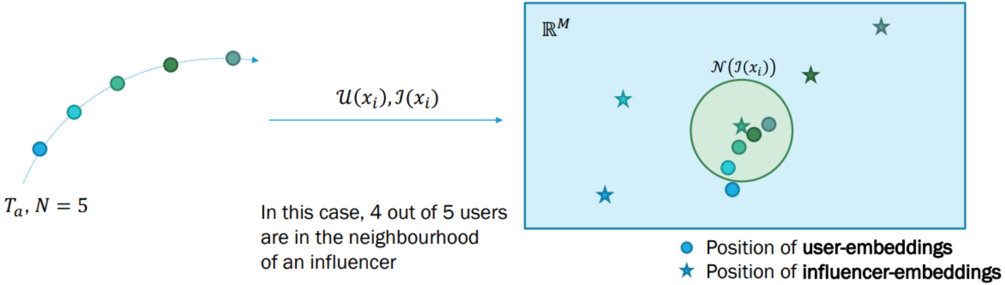



credit: [Kilian Lieret, CHEP23]

# Tracking with metric learning: Object Condensation

- More formally:

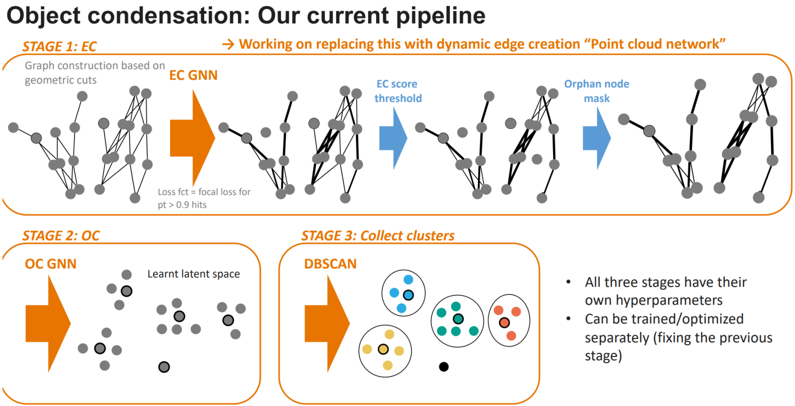
  - We want a minimum in the loss when *all* hits $x_i \in T_a$ have $\mathcal{U}(x_i)$ inside neighbourhood $\mathcal{N}(\mathcal{I}(x_i))$ for **at least one influencer, and *only* one influencer**




credit: [Daniel Murnane, CTD23]

- Technical details: [2002.03605]

# Tracking with metric learning: Object Condensation Tracking Pipeline

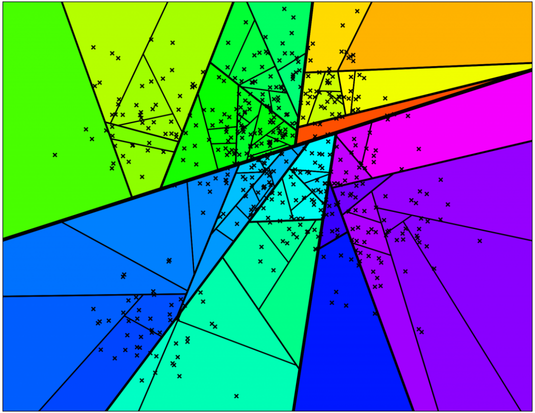



credit: Kilian Lieret, CHEP23

- Instead of Metric learning for graph building to be passed to GNN ...
  $\Longrightarrow$ Build graph with GNN then implement metric learning!

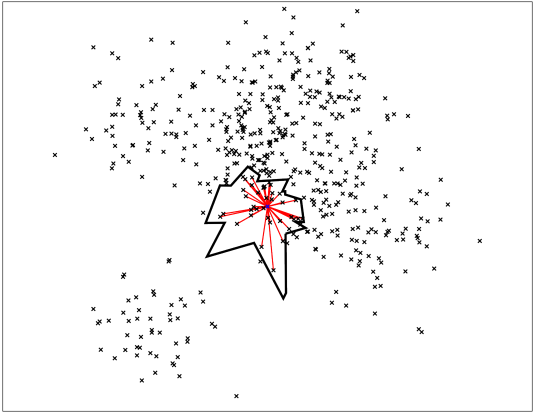
# Hybrid models? Approximate Nearest Neighbor Search

- Learn a suitable metric space
- Segment it in different regions, in $\mathcal{O}(\log N_{\text{hits}})$



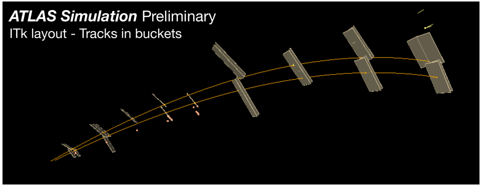
source

- Quickly lookup union of regions being approximately closest to a query point
- Perform "classical" track finding in each region

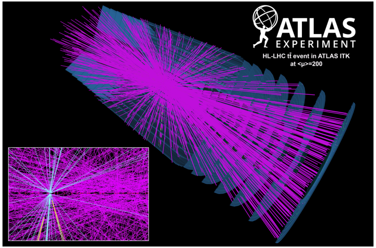


# Approximate Nearest Neighbor Search: Divide-and-conquer

- This is much easier. . .




- . . . Than this!




- Allows "easy" parallelisation over regions!

# Conclusion

- Today we've seen:
  - Elements of "classical" tracking pipelines
  - Why is there intense R&D to replace or ameliorate them
  - The GNN approach
  - The metric learning approach
  - And hybrid methods!
- This is just a small fraction of the landscape!
- Tracking is a great playground for ML due to non-standard nature of the problem

# Merci!