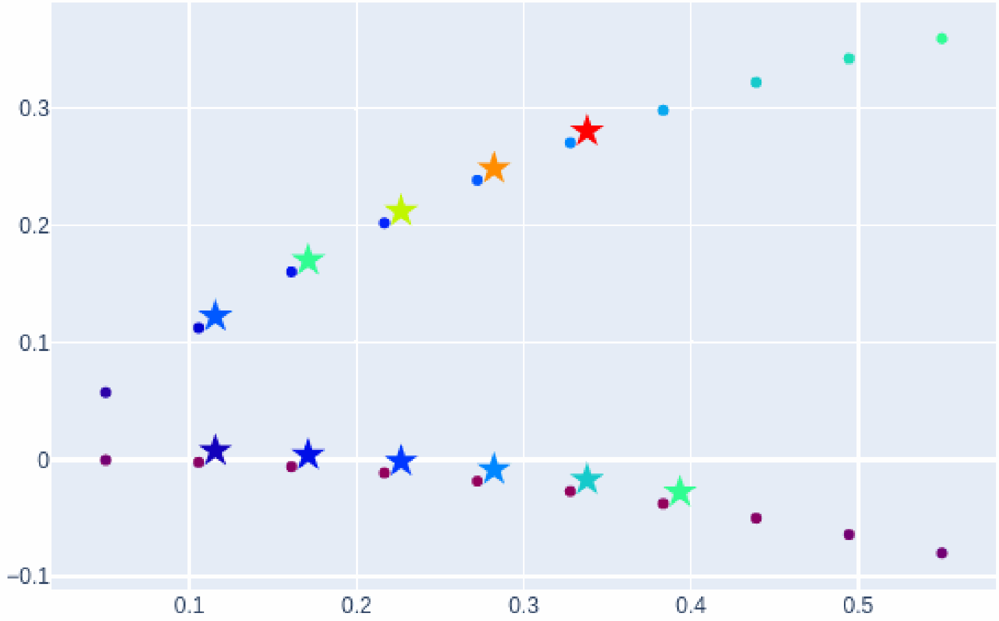
TRACK FINDING-AND-FITTING WITH INFLUENCER OBJECT CONDENSATION

CONNECTING THE DOTS

TOULOUSE, FRANCE, OCT 10TH 2023

DANIEL MURNANE

ON BEHALF OF THE **EXATRKX PROJECT**



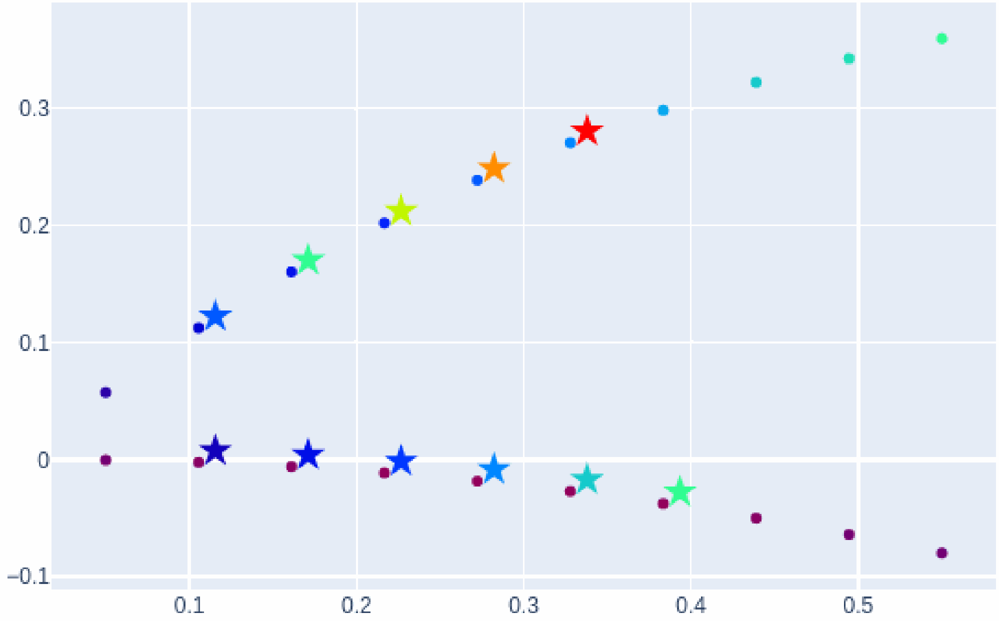
TRACK FINDING ~~AND FITTING~~ WITH INFLUENCER OBJECT CONDENSATION

CONNECTING THE DOTS

TOULOUSE, FRANCE, OCT 10TH 2023

DANIEL MURNANE

ON BEHALF OF THE **EXATRKX PROJECT**

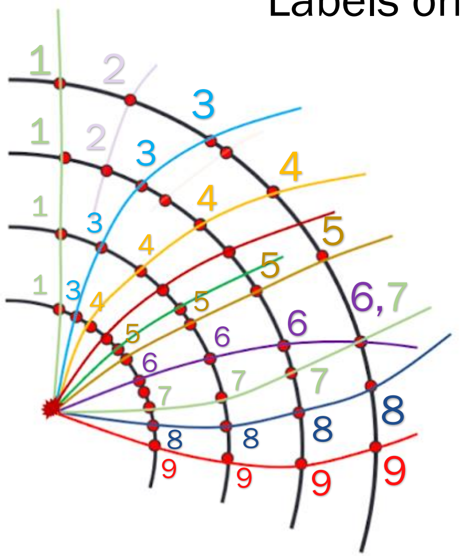


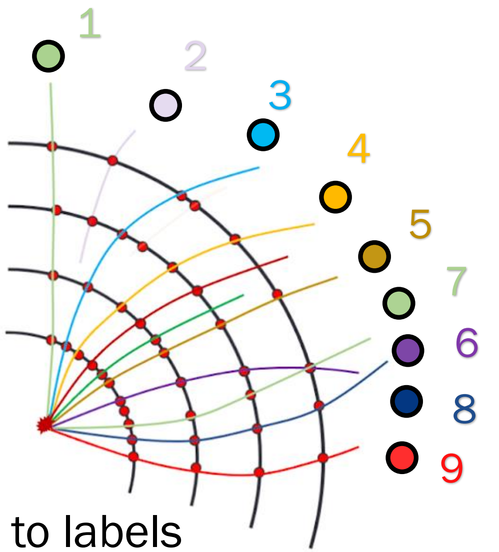
BERKELEY LAB

THE TRACKING PROBLEM

- Protons collide in center of detector, "shattering" into thousands of particles
- The *charged* particles travel in curved **tracks** through detector's magnetic field (Lorentz force)
- A track is defined by the **hits** left as energy deposits in the detector material, when the particle interacts with material
- **In this study, we use the TrackML Dataset [link], with variable-sized subsets of tracks selected**
- The goal of track reconstruction: Given set of hits from particles in a detector, assign label(s) to each hit.

Labels on hits





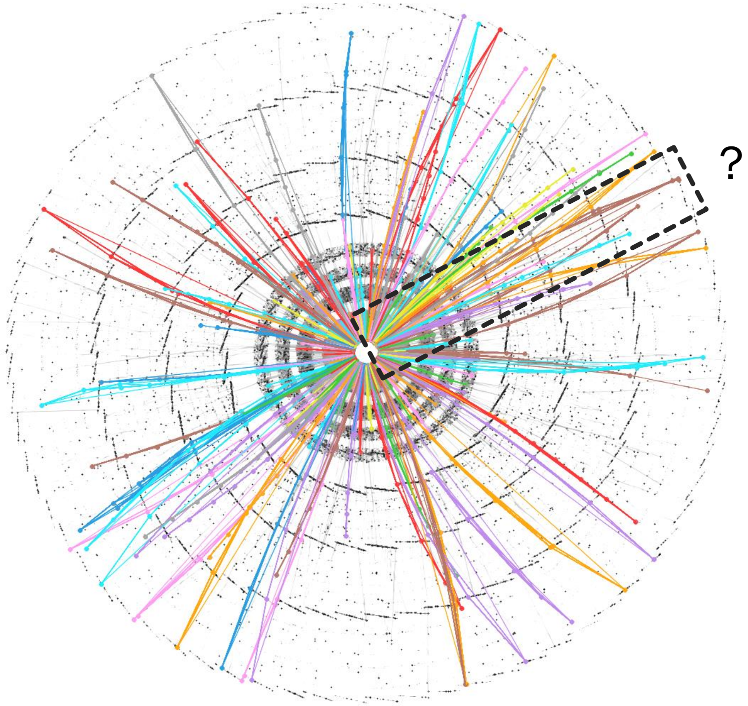
Hits to labels

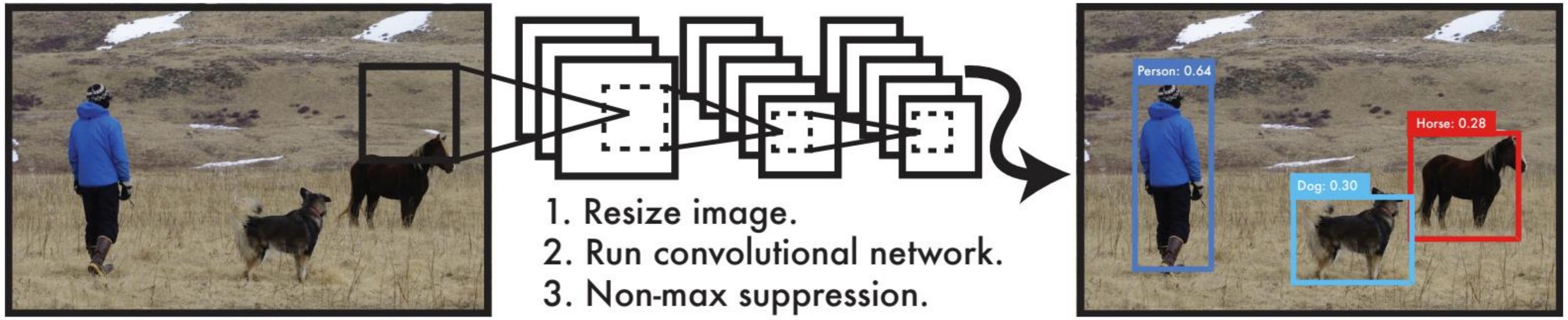
Can reframe the problem of assigning *label* → *hits*

1. Assume the existence of some uniquely labelled "*representative point*" in each track object
2. Then our task is to assign *hits* → *representative point*

TRACKING AS OBJECT DETECTION

- A well-studied problem in computer vision: Given an image, can we identify all discrete objects of interest and predict information about them?
- Popular approach is to draw a bounding box as the representative label
- Can't directly use this approach for tracking: tracks are not localized in 3D space

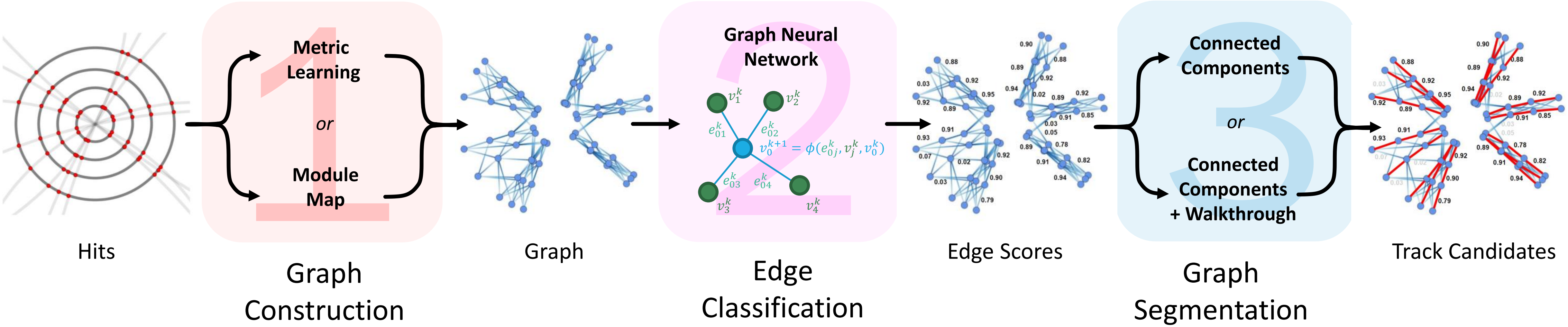




The "You Only Look Once" (YOLO) approach to detection: draw a bounding box and predict the object in a single step.
Redmond et al, arXiv: 1506.02640

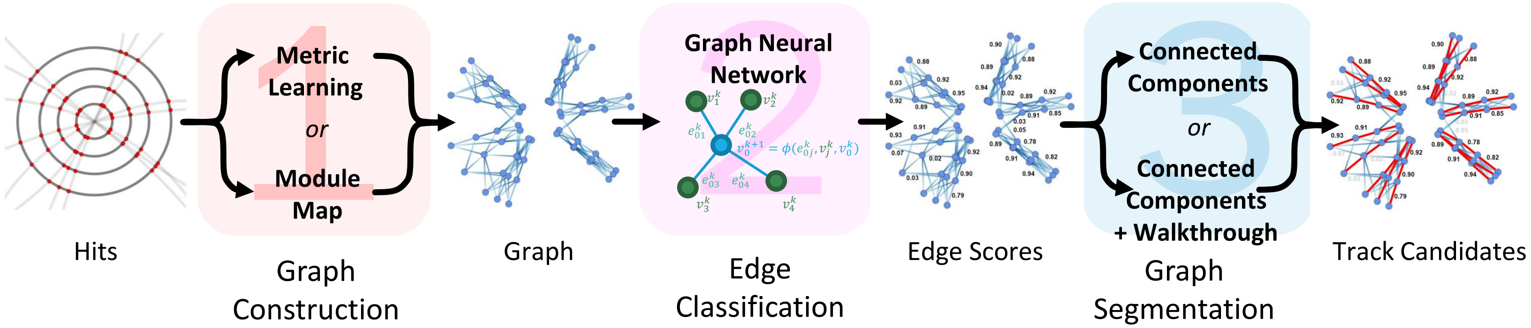
WHAT IS OUR CURRENT ML TRACKING APPROACH?

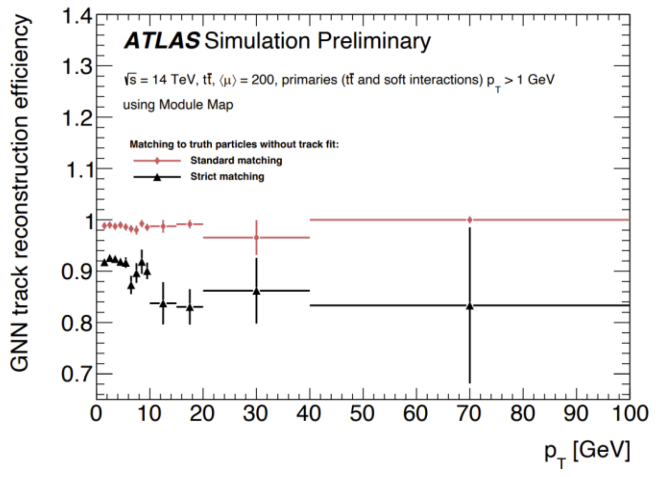
- The GNN4ITk project has a proof-of-concept running on HL-LHC full pileup simulation
- Has the following structure:



WHAT IS OUR CURRENT ML TRACKING APPROACH?

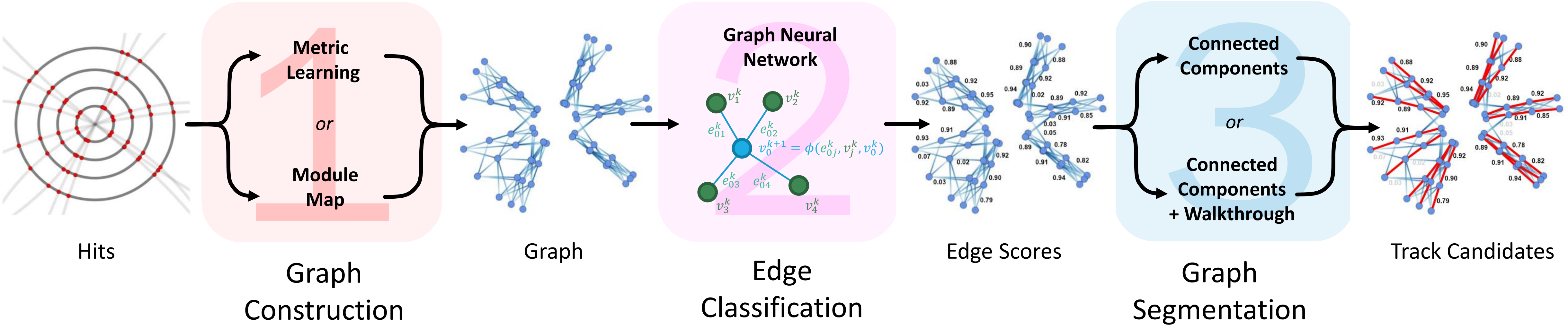
- This pipeline works very well, in terms of physics performance
- But the graph construction (e.g. filtering) and track building (e.g. labelling) together take 70% of the time!
- Would like to skip the graph construction, and do labelling with one step…



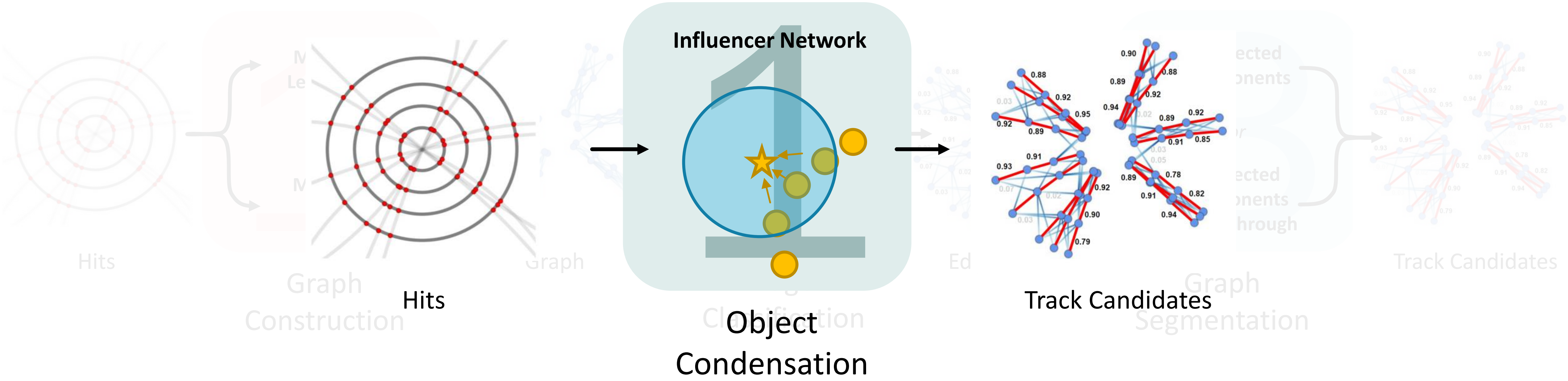


	Baseline	Faiss	cuGraph	AMP	FRNN
Data Loading	0.0022 ± 0.0003	0.0021 ± 0.0003	0.0023 ± 0.0003	0.0022 ± 0.0003	0.0022 ± 0.0003
Embedding	0.02 ± 0.003	0.02 ± 0.003	0.02 ± 0.003	0.0067 ± 0.0007	0.0067 ± 0.0007
Build Edges	12 ± 2.64	0.54 ± 0.07	0.53 ± 0.07	0.53 ± 0.07	0.04 ± 0.01
Filtering	0.7 ± 0.15	0.7 ± 0.15	0.7 ± 0.15	0.37 ± 0.08	0.37 ± 0.08
GNN	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03
Labeling	2.2 ± 0.3	2.1 ± 0.3	0.11 ± 0.01	0.09 ± 0.008	0.09 ± 0.008
Total time	$15 \pm 3.$	3.6 ± 0.6	1.6 ± 0.3	1.2 ± 0.2	0.7 ± 0.1

WHAT IS OUR CURRENT ML TRACKING APPROACH?



WHAT IS OUR CURRENT ML TRACKING APPROACH?



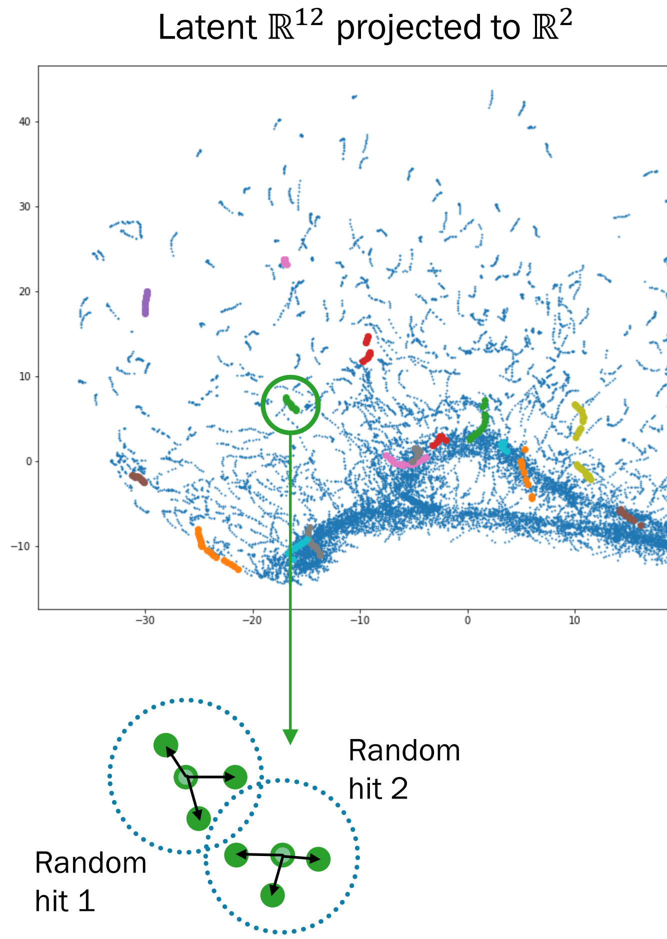
OBJECT DETECTION AS METRIC LEARNING

- We consider a "naïve" solution to the object detection problem
- Using a transformer or graph neural network (GNN), embed each hit x_i in a latent space $\mathcal{U}(\mathrm{x_i})$
- Use a hinge loss to encourage hits from the same particle ($y_{ij} = 1$) to be close, hits from different particles ($y_{ij} = 0$) to be distant:

$$L = \begin{cases} \Delta_{\mathrm{ij}}, & when\ \ y_{ij} = 1 \\ \max(0, 1 - \Delta_{\mathrm{ij}}), & when\ \ y_{ij} = 0 \end{cases}$$

To create *representative points*, we use a "greedy condensation" approach. For all points:

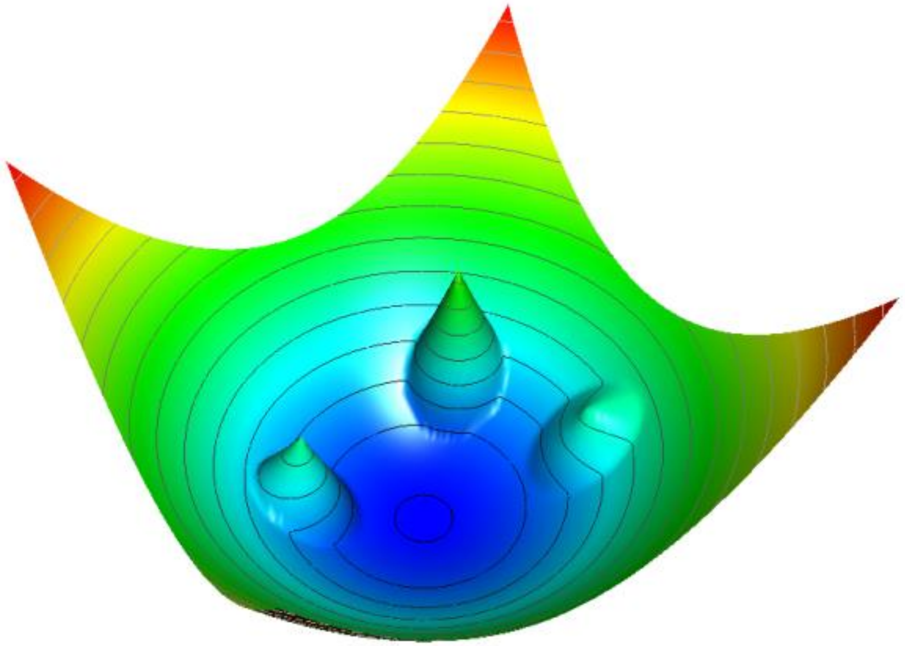
1. Randomly select a point
2. Find all neighbors (within radius R)
3. If none of the neighbors are already a representative, then convert the point to a representative, and attach all neighbors to that representative



Let's call this the **naïve benchmark.** Works quite well, but some points are clearly better candidates for representative than others. Can we learn which points are good representative points?

OBJECT CONDENSATION: LEARNING REPRESENTATIVE POINTS

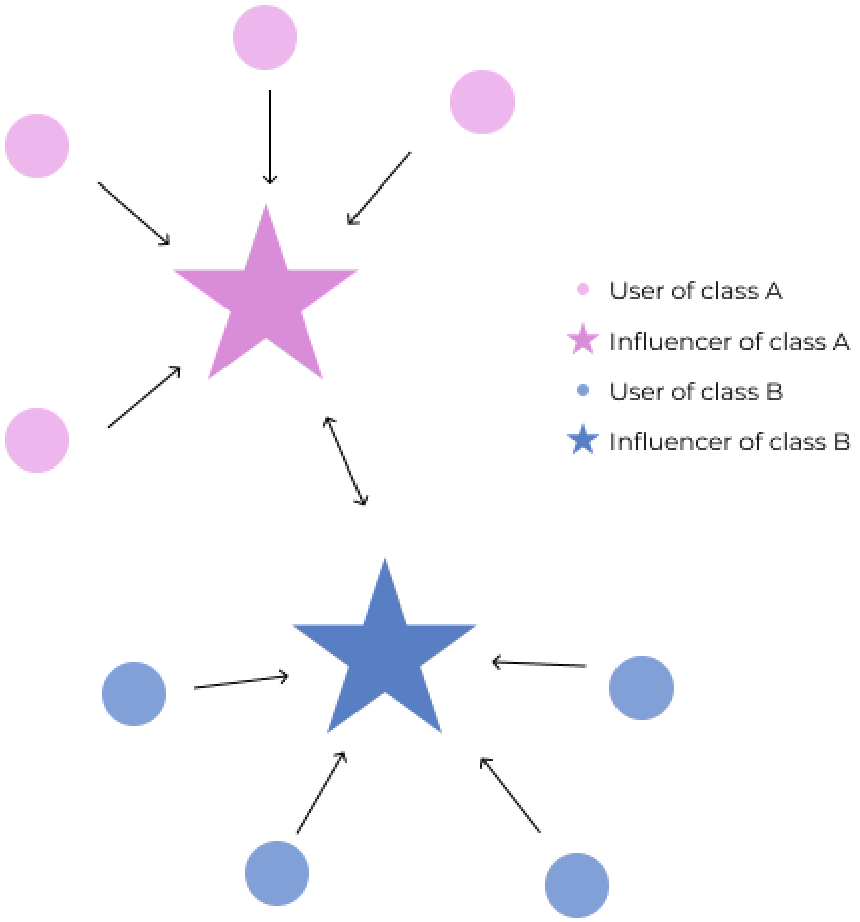
- Idea from particle flow reconstruction: *Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph, and image data*, Kiesler 2020 [link]
- Simultaneously learn an embedding similarity space **and** a condensation score for each hit, where a higher score is a more "attractive" point charge in similarity space
- All hits with learned condensation score β above some threshold are considered candidates for representation points, then we apply greedy condensation to the representatives sorted by β
- Shortcomings:
  - Having this "hard cut" charge threshold requires fine-tuning
  - Inference requires sorting likely condensation points and sequentially considering each condensation point based on all previous condensation points
  - Training (as a simplification) only considers *maximum-scoring* condensation point in each class, which neglects global optima



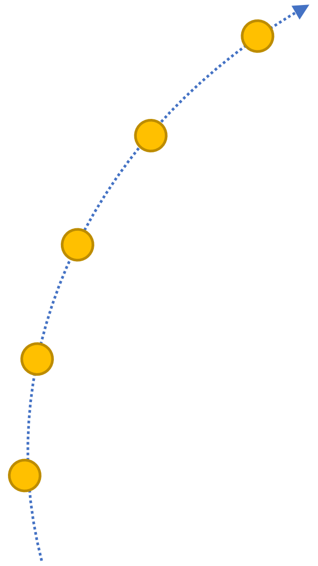
The potential function of members of the same class relative to the representation point of that class (*Kiesler 2020*)

COLLISION EVENT AS A SOCIAL NETWORK

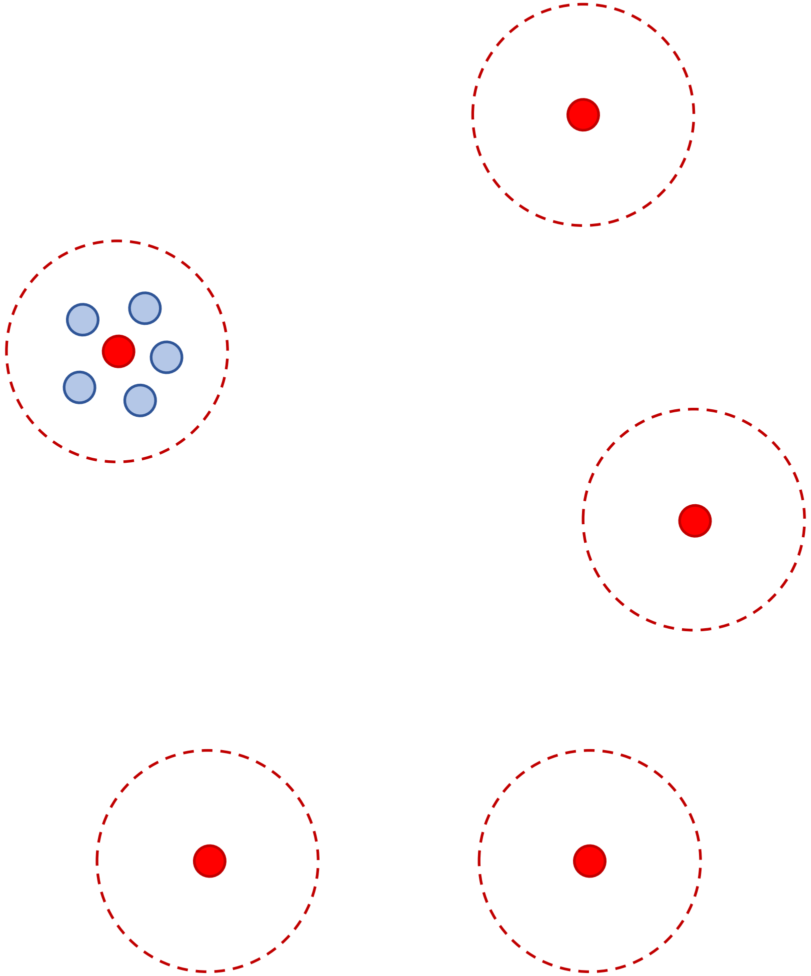
- A social network has nodes that are *more important* than others – representative nodes, or “Influencer” nodes
- In a directed graph, they have many incoming edges
- “User” nodes are represented by an Influencer, and have one outgoing
- How to use metric learning to build a directed graph?
- **Key idea:** A member of a network can be both a User and Influencer
- We can build a directed graph by learning for each member of the point cloud two embeddings in the *same space*: a **user-embedding** and an **influencer-embedding**



The goal…



$$f_1 \, , f_2$$



Embed *all* hits with *two, separate* functions.

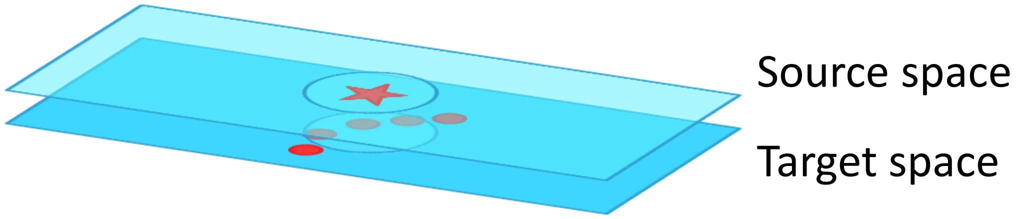
For each red hit, find all neighboring blue hits. That is a track.

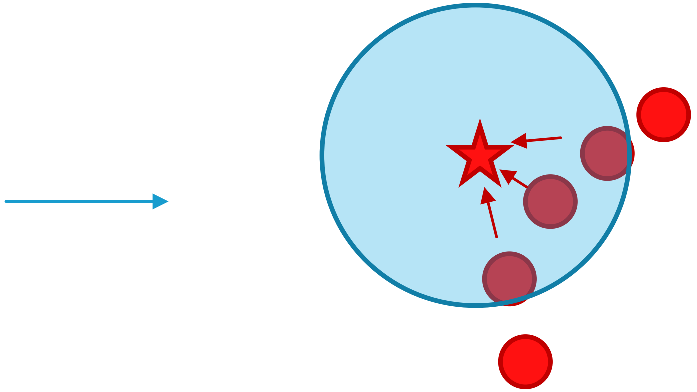
THE STORY SO FAR…

1. We want to go from hits to tracks: a set of points $\{x_1, x_2, \ldots, x_n\}$ to a set of sets of points $\{T_1, T_2, \ldots, T_N\}$
2. We need introduce track-like objects *somewhere* to represent these sets
3. We can do this as a post-processing (as in GNN4ITk or the naïve baseline),
4. We can also do this from within the set of hits itself to be differentiable (as in object condensation where we classify hits as good track-like representatives)
5. Rather than choose which hits could be representatives, let all hits be track-like representatives, and those that have hits that crowd around them are selected as good representatives

RECIPE: METRIC LEARNING FOR A DIRECTED GRAPH

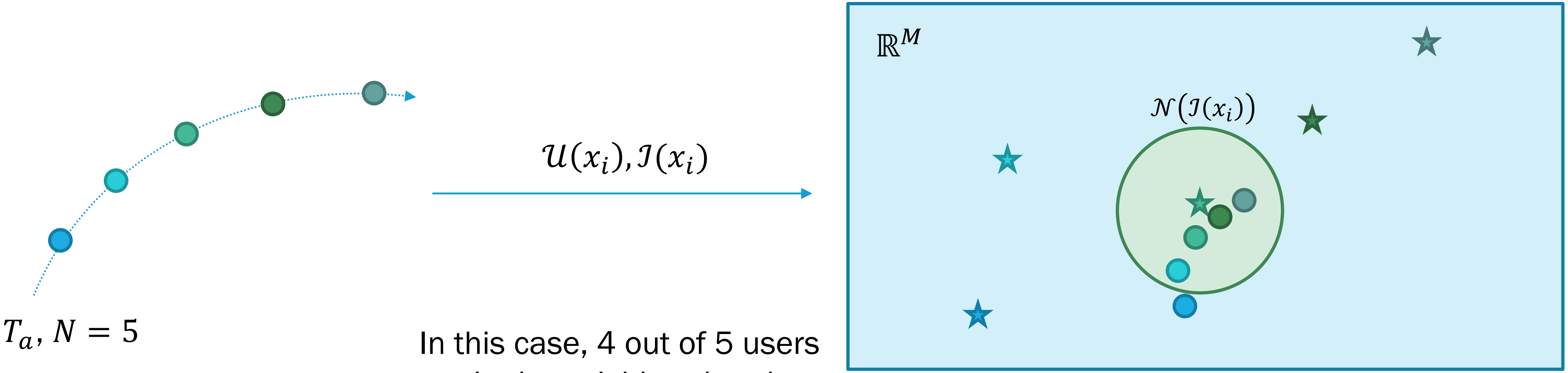
1. We want one hit from each track to represent *all* hits in that track
2. Rather than learning some "representative score" for each hit, we simply want to learn an embedding where each hit "points to" its representative
3. To create this pointing (a directed graph), we need *two* embeddings: one source space, one target space
4. All hits are embedded into both the source space and the target space
5. A directed graph is constructed by connecting nodes in the target space that are close to nodes in the target space





DESIRED LOSS FUNCTION BEHAVIOUR

- Given each of N points x_i in track T_a embedded into $\mathbb{R}^M$ with two models: a user-embedding $\mathcal{U}$ and an influencer-embedding $\mathcal{I}$
- We want a minimum in the loss when *all* hits $x_i \in T_a$ have $\mathcal{U}(x_i)$ inside neighbourhood $\mathcal{N}\big(\mathcal{I}(x_i)\big)$ for **at least one influencer, and *only* one influencer**

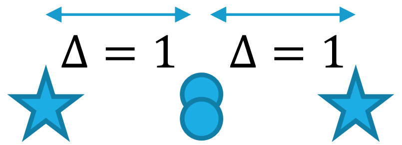


In this case, 4 out of 5 users are in the neighbourhood of an influencer

- Position of **user-embeddings**
- Position of **influencer-embeddings**

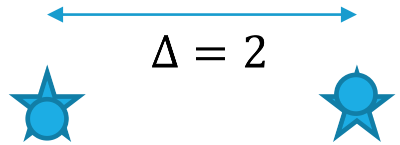
DESIRED LOSS FUNCTION BEHAVIOUR

- Given each of N points x_i in track T_a embedded into $\mathbb{R}^M$ with two models: a user-embedding $\mathcal{U}$ and an influencer-embedding $\mathcal{I}$
- We want a minimum in the loss when ***all*** hits $x_i \in T_a$ have $\mathcal{U}(x_i)$ inside neighbourhood $\mathcal{N}\big(\mathcal{I}(x_i)\big)$ for at least one influencer (and preferably *only* one influencer)
- We can achieve this by taking $L_u(T_a) = \sqrt[N]{\prod_j \frac{1}{N} \sum_i \Delta_{ij}^2}$, where $\Delta_{ij} = \left|\mathcal{U}(x_i) - \mathcal{I}(x_j)\right|$
- Consider loss L in simple example of two points in three different cases:



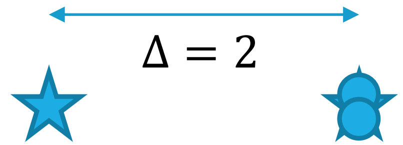
$L \propto (\Delta_{00}^2 + \Delta_{01}^2) \times (\Delta_{10}^2 + \Delta_{11}^2) = 4$

Case A



$L \propto (\Delta_{00}^2 + \Delta_{01}^2) \times (\Delta_{10}^2 + \Delta_{11}^2) = 16$

Case B



$L \propto (\Delta_{00}^2 + \Delta_{01}^2) \times (\Delta_{10}^2 + \Delta_{11}^2) = 0$

Case C

Note: Noise is given a class label NaN and handled like all other data points

● Position of **user-embeddings**
★ Position of **influencer-embeddings**

ATTRACTIVE INFLUENCER LOSS

- The attractive Influencer loss for track a is $L_a^+ = \sqrt[N]{\prod_j \frac{1}{N}\sum_i \Delta_{ij}^2}$, where $\Delta_{ij} = \left|\mathcal{U}(x_i) - \mathcal{I}(x_j)\right|$
- It has a minimum when all user embeddings $\mathcal{U}(x_i)$ are close to at least one influencer embedding $\mathcal{I}(x_j)$, therefore it **attracts** users to influencers of the same class
- The attractive Influencer loss is actually the **geometric mean across influencers** of the **arithmetic mean across users** of the distance between each positive pair across all n tracks, so we can rewrite it for numerical stability:

$$L_a^+ = \exp\left(\frac{1}{N}\sum_j \ln(\frac{1}{N}\sum_i \Delta_{ij}^2)\right), \qquad L^+ = \frac{1}{n}\sum_a L_a^+, \qquad y_{ij} = 1$$

- Looks pretty damn ugly! It’s a triple for-loop. Luckily, we can parallelise this on GPU

REPULSIVE INFLUENCER LOSS

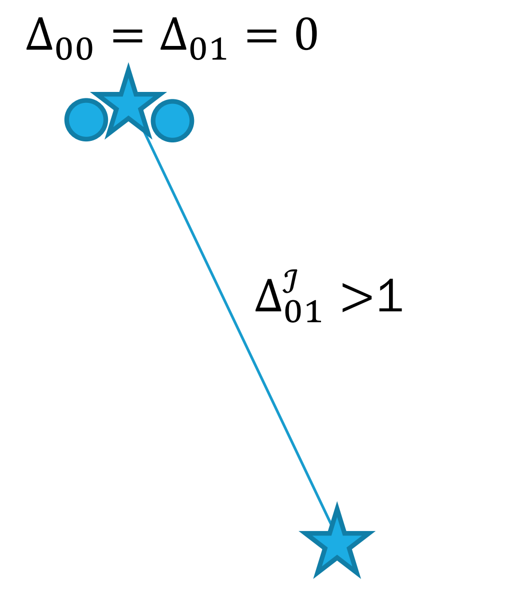
- Recall our desired loss function behaviour:

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

- The attractive loss gives all hits close to at least one influencer
- To constrain this neighbourhood to contain **exactly one** influencer, we must punish influencers for being close to one another:

$$L^{-} = \text{mean}_{ij}(\max\left(0, 1 - \Delta_{ij}^{\mathcal{I}}\right))$$

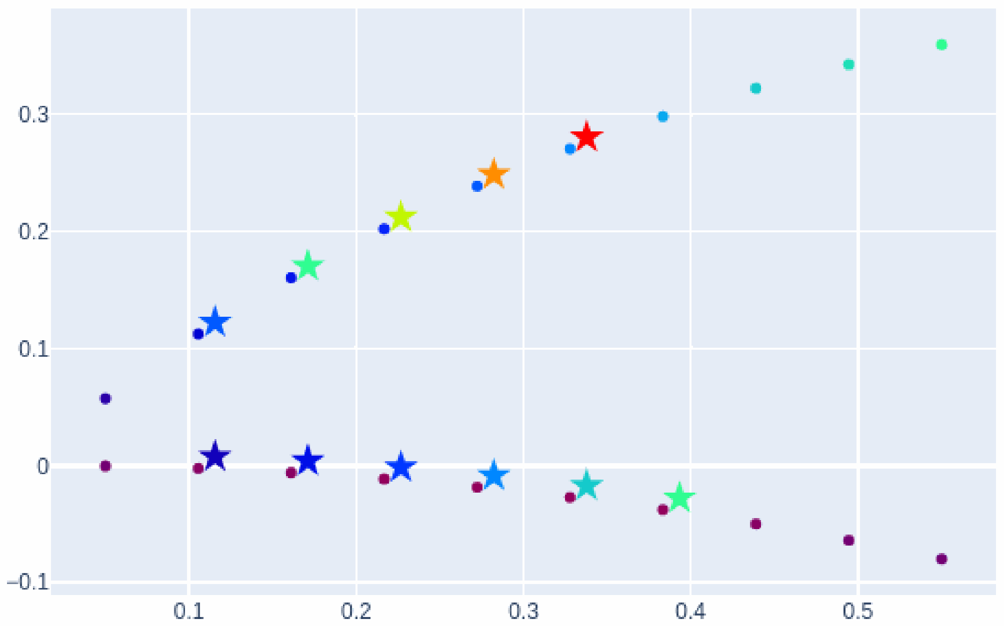
- This is a simple repulsive hinge loss, which has a maximum at $\Delta_{ij}^{\mathcal{I}}$=0, and a minimum at $\Delta_{ij}^{\mathcal{I}} \geq 1$
- As it is linear, it turns out to **not be strong enough** to overcome the attractive influencer loss, leading to high duplicate rates



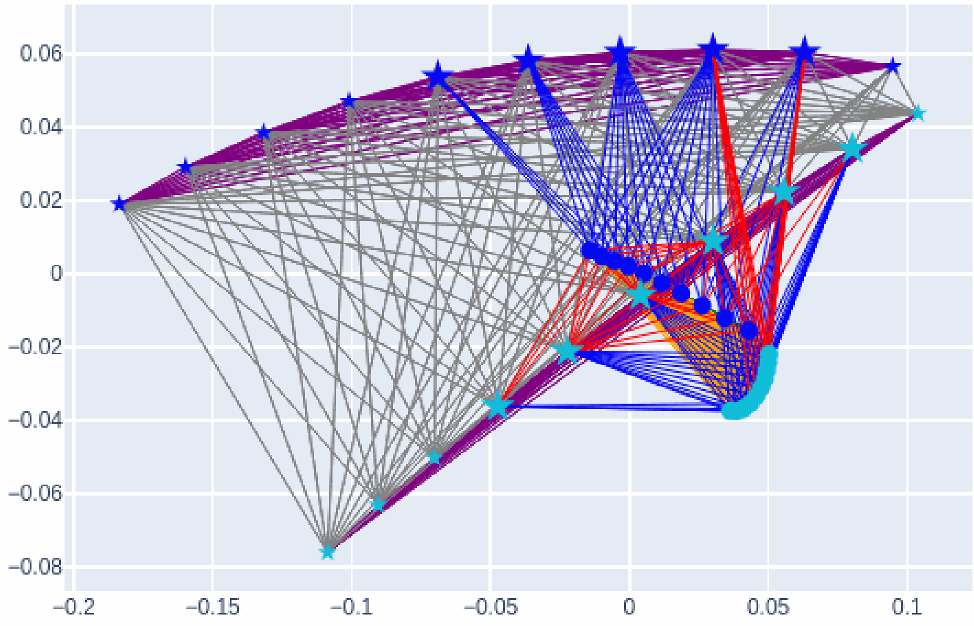
The total Influencer Loss is at a minimum in this case

A TRAINING MONTAGE

REAL SPACE



EMBEDDING SPACE

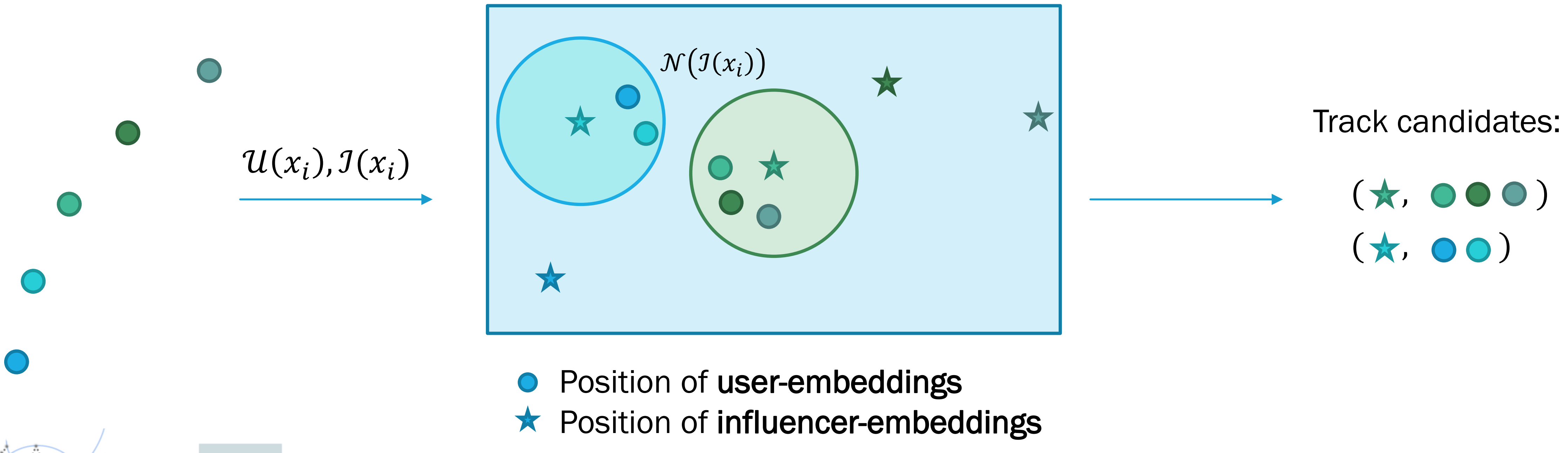


- We can see the Influencer Loss working on two tracks above, across training epochs
- In **Real Space**, we show only Users (circles) and Influencers (stars) when they are associated with an Influencer or User (respectively)
- The color in **Real Space** is a projection in 1D of the location in **Embedding Space**
- In **Embedding Space**, we should edges created, and connected Influencers are large stars, unconnected Influencers are small stars

INFLUENCER INFERENCE

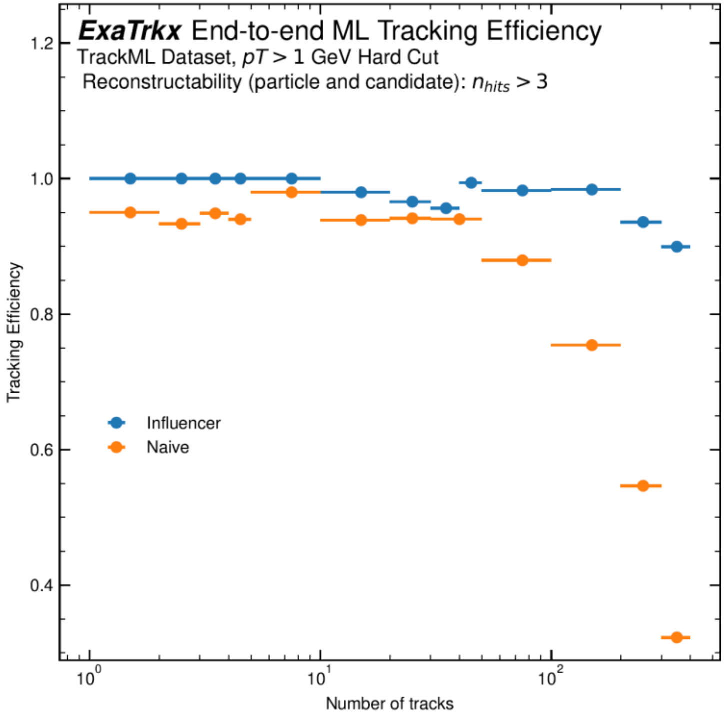
To construct track candidates,

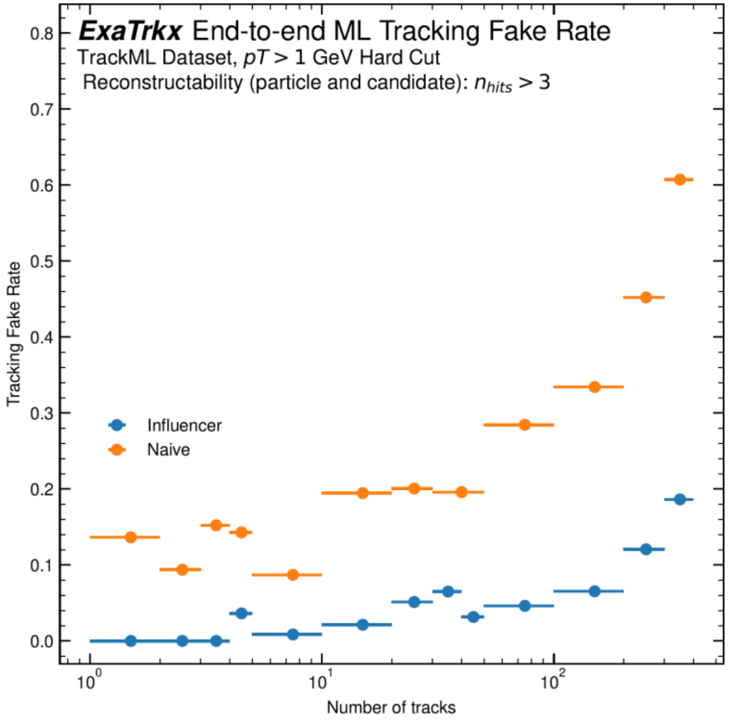
1. Embed hits with $\mathcal{U}(x_i), \mathcal{I}(x_i)$ into $\mathbb{R}^M$
2. Perform fixed-radius nearest neighbour (FRNN) search, with $\mathcal{U}(x_i)$ as database, $\mathcal{I}(x_i)$ as query
3. All non-empty Influencer neighbourhoods are track candidates of user hits $\{x_i \mid \mathcal{U}(x_i) \in \mathcal{N}(\mathcal{I}(x_i))\}$, each represented by an influencer hit. No further processing is required



PHYSICS PERFORMANCE

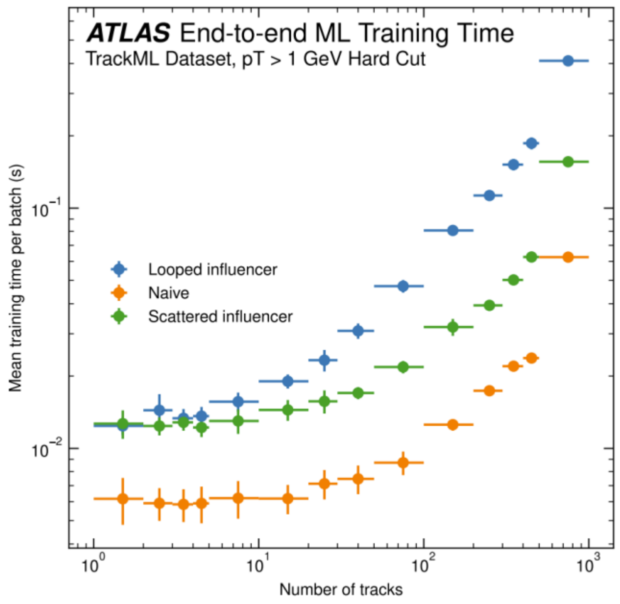
- Comparison of track reconstruction of naïve condensation and influencer condensation event size
- Influencer loss is able to condense tracks much more efficiently, and with far fewer fake track candidates

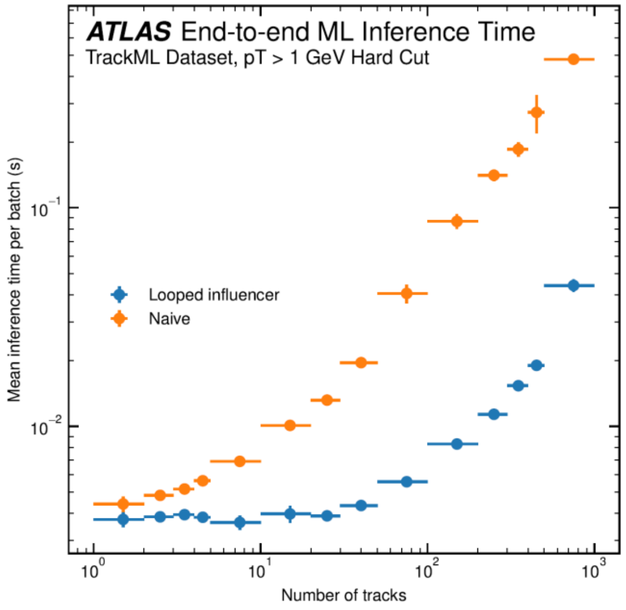


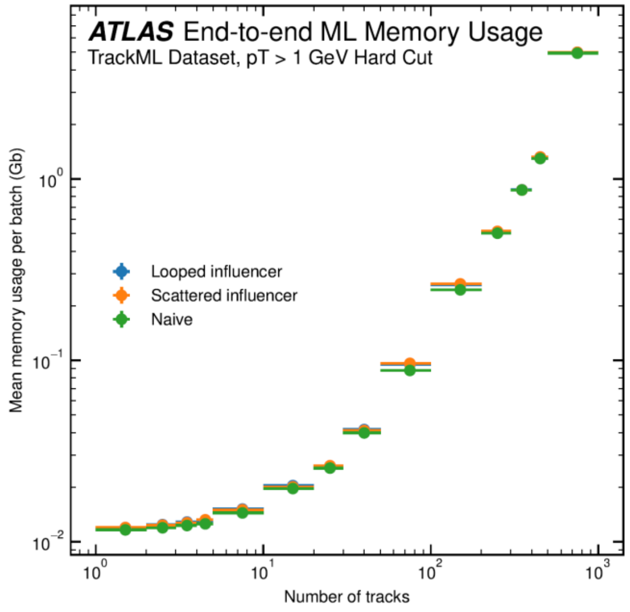


COMPUTATIONAL PERFORMANCE

- Influencer loss is currently an expensive calculation and a slow function to minimize, compared with the Naïve Hinge Loss
- Can be sped-up with a careful use of scatter-aggregations on the GPU
- However, this cost is only incurred during training and is amortized in inference
- The Naïve model's greedy condensation creates tracks sequentially, while the Influencer condensation occurs in parallel and on a significantly more sparse neighbourhood structure (c.f. the training montage to see this at work)







CONCLUSION

- Graph neural networks and transformers are a proven technique for tracking, given sufficient pre-and-post processing
- To perform tracking in a single step, we need to assign all hits in each track to a representative point
- We can do this with the Influencer Loss function
- Track finding inference with a fully trained Influencer network *much* faster than regular object condensation, and gives similar or better physics performance

Next steps

- Since an Influencer point represents a whole track, we should be able to regress track-level features on it
- Understand why this is not working out of the box!
- Reduce the duplicate rate produced in the Influencer condensation approach, possibly with stronger repulsive loss function