Particle track reconstruction with the TrackNETv2

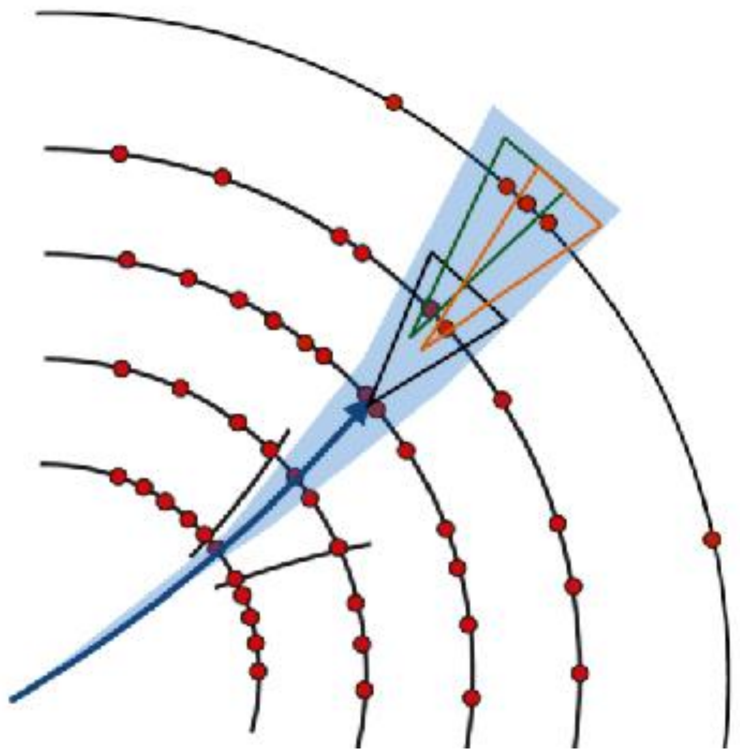
Pavel Goncharov, Gennady Ososkov, Dmitriy Baranov

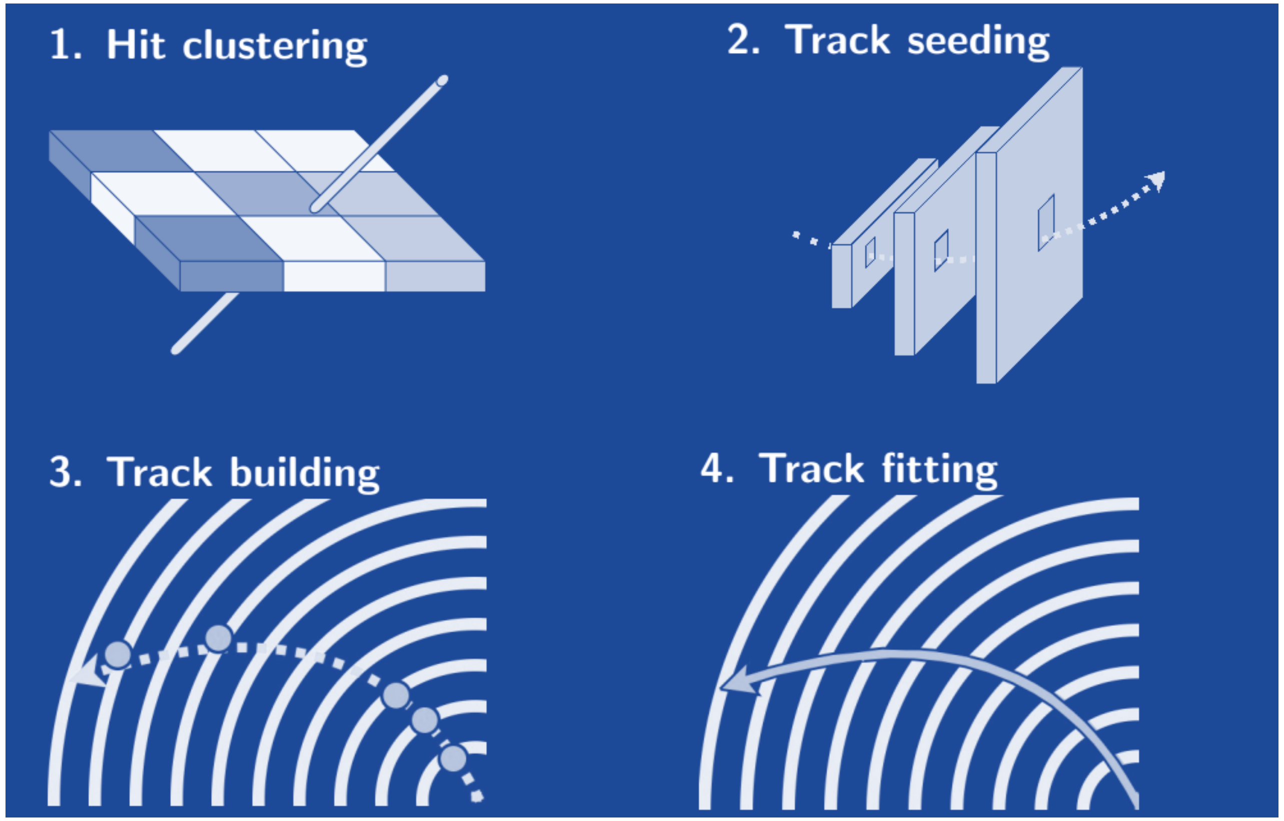
Sukhoi State Technical University of Gomel

kaliostrogoblin3@gmail.com

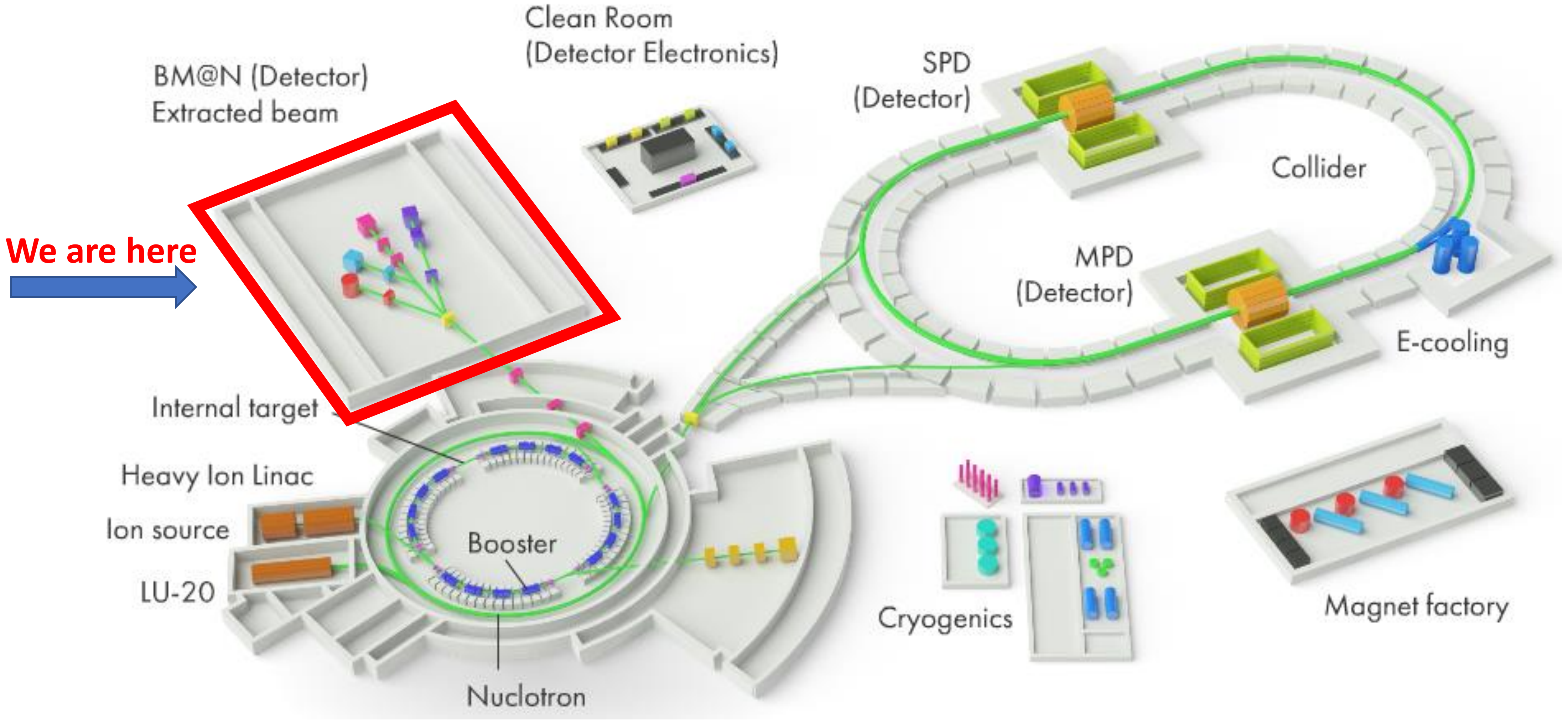
What is tracking?

Tracking or track finding is a process of **reconstruction the particle's trajectories** in high-energy physics detector by connecting the points – hits – that each particle leaves passing through detector's planes. Tracking includes track seeding and track building phases.



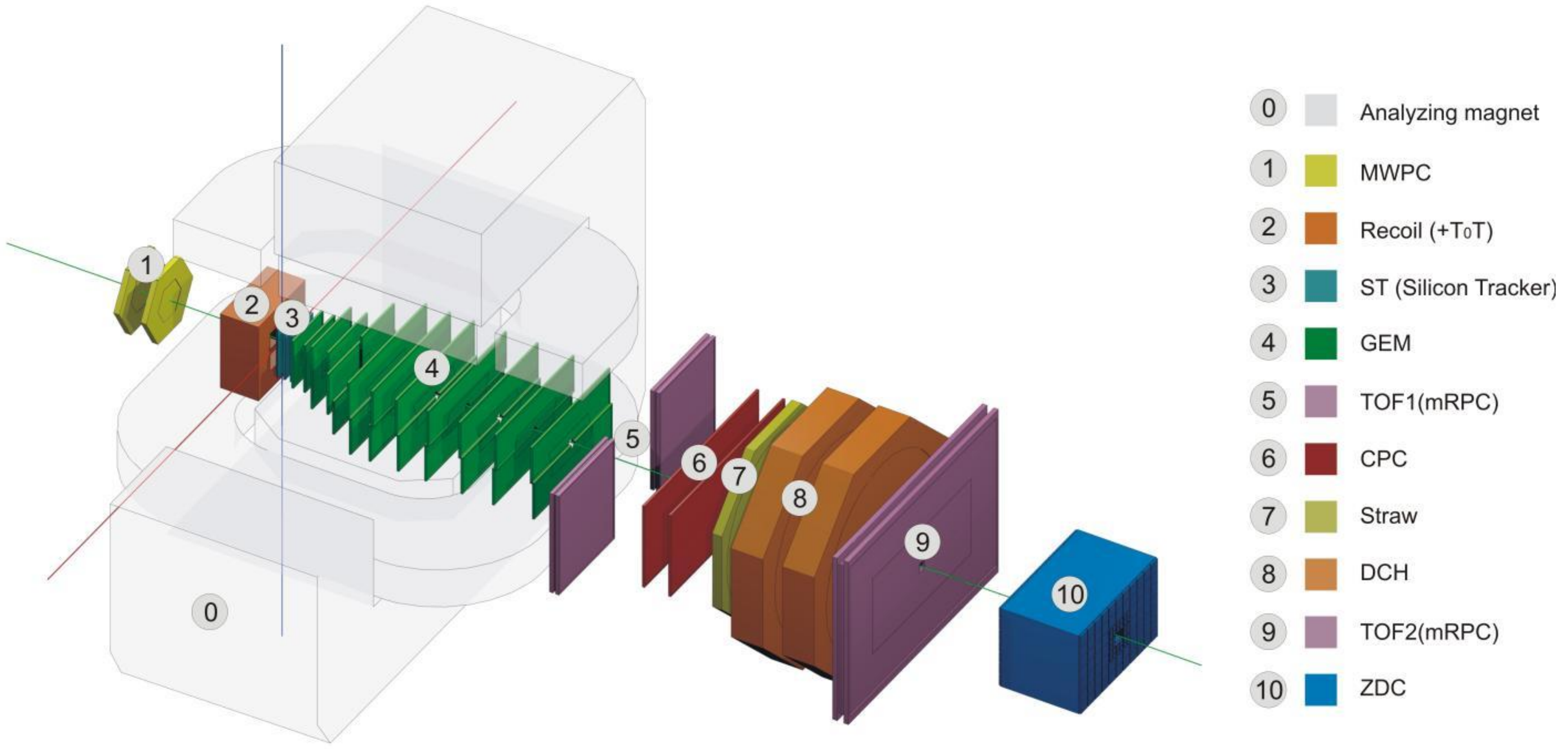


NICA-MPD-SPD-BM@N



General view of the NICA complex with the experiments MPD, SPD, BM@N

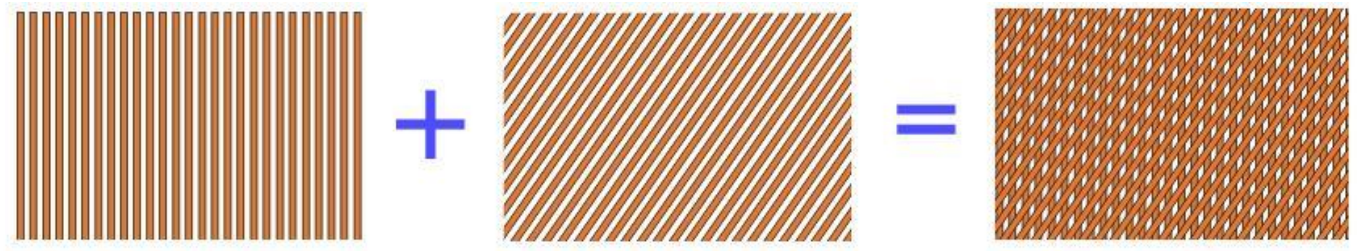
Baryonic Matter at Nuclotron (BM@N)



- Our problem is to reconstruct tracks registered by the GEM vertex detector with 6 GEM-stations (**RUN 6, spring 2017**) inside the magnet.
- All data for further study was simulated in the BmnRoot framework with LAQGSM generator.

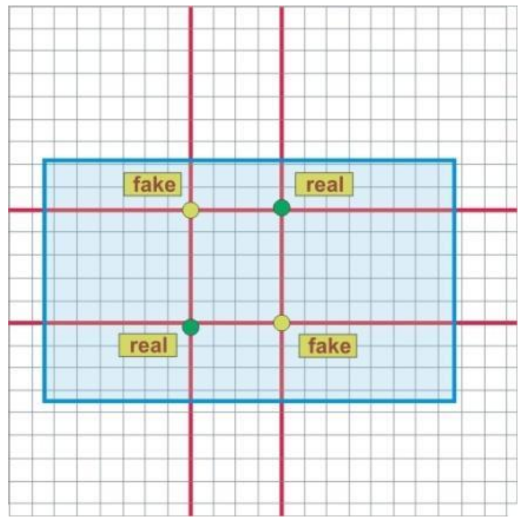
Problems of microstrip gaseous chambers

The general schema of construction of any GEM-station

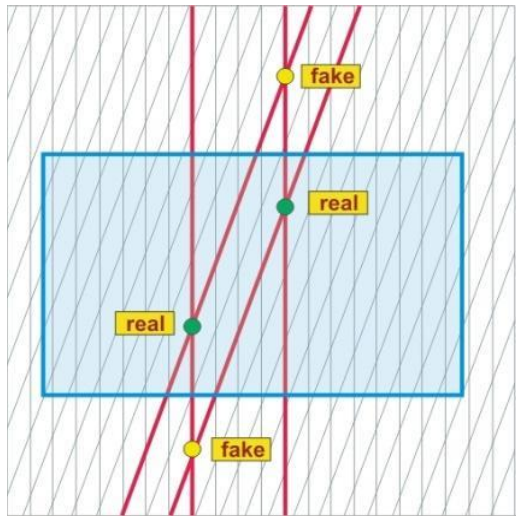


Layer of vertical strips Layer of inclined strips Complete readout plane

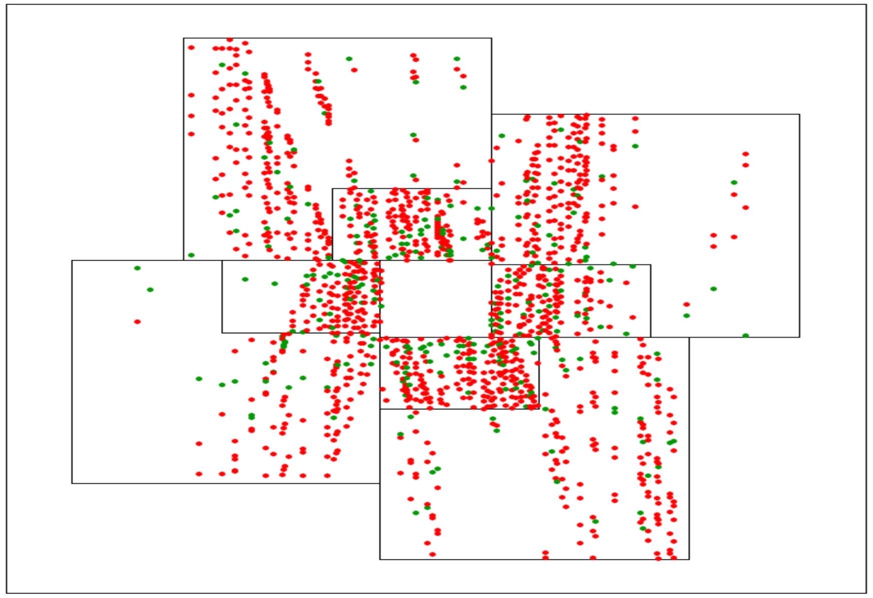
The main shortcoming is the appearance of **fake hits caused by extra spurious strip crossings. For n real hits one gains n^2- n fakes**



We can significantly reduce the number of observed fakes by adding a stereo-angle between layers of strips (15˚)



Although small angle between layers removes a lot of fakes, pretty much of them are still left



However **too high reducing of the angle increases the Y-coordinate error**

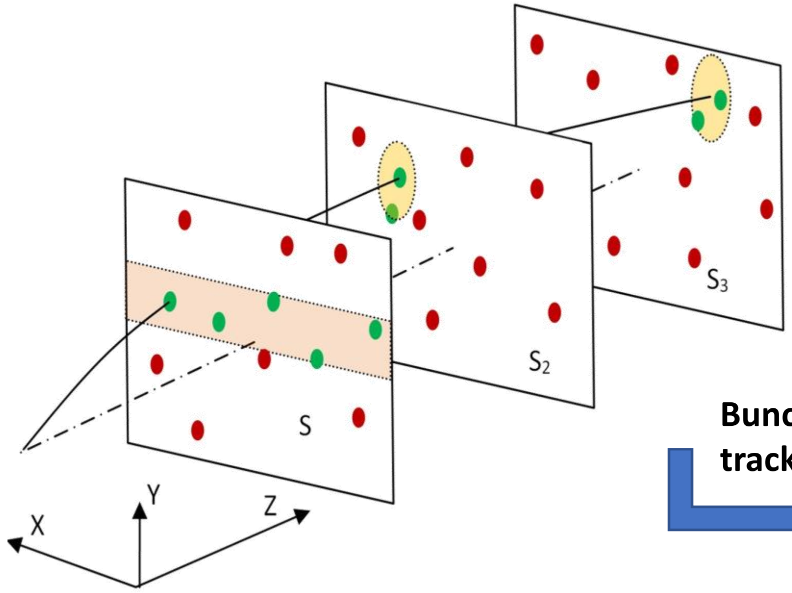
Initial Attempts. Two-step tracking

http://ceur-ws.org/Vol-2023/37-45-paper-6.pdf

1. **Preprocessing by directed K-d tree search** to find all possible track-candidates as clusters joining all hits from adjacent GEM stations lying on a **smooth curve**.
2. **Deep recurrent network** trained on the big simulated dataset with 82 677 real tracks and 695 887 ghosts **classifies track-candidates in two groups: true tracks and ghosts**.

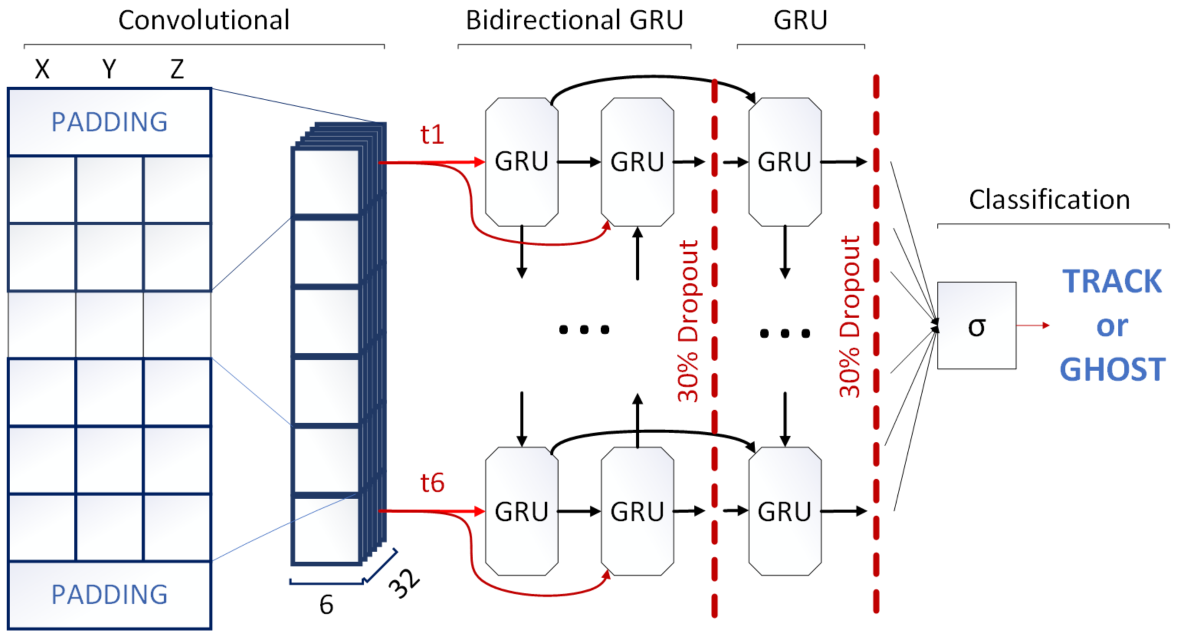
very imbalanced dataset

1) Directed K-d Tree Search



Bunch of track-candidates

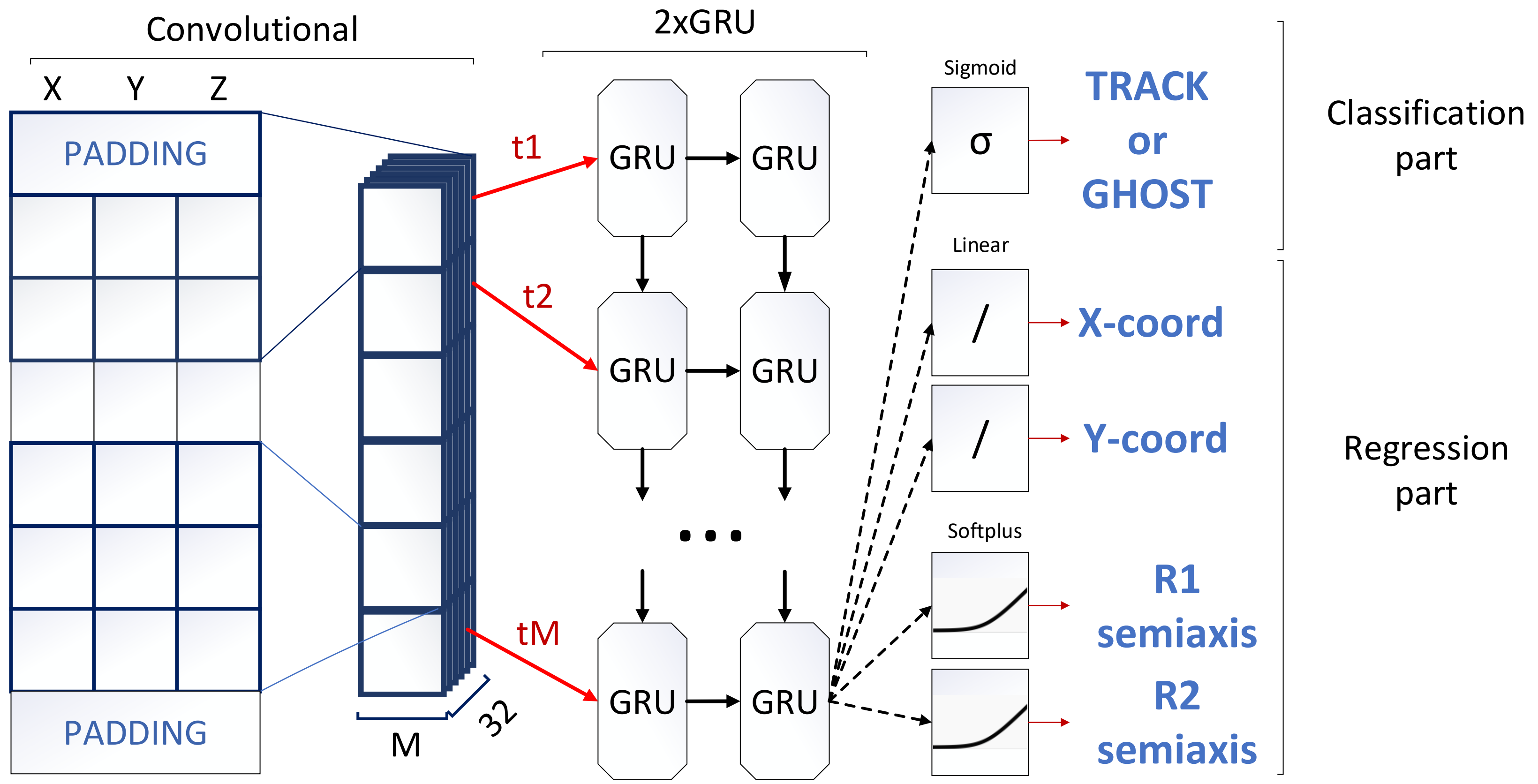
2) Deep Recurrent Neural Network Classifier



Initial Attempts. TrackNETv1

https://www.epj-conferences.org/articles/epjconf/pdf/2019/06/epjconf_ayss18_05001.pdf

We introduced the regression part consisting of four neurons, two of which **predict the point of the center of ellipse on the next coordinate plane**, where to search for track-candidate continuation and another two – **define the semiaxis of that ellipse.**



TrackNETv1 custom loss

$$L = \underbrace{\boxed{\max(\lambda_1, 1 - p)\, FL(p, p')}}_{\text{Classification error}} + p\left(\underbrace{\boxed{\lambda_2 \sqrt{\left(\frac{x - x'}{R1}\right)^2 + \left(\frac{y - y'}{R2}\right)^2}}}_{\text{Point in ellipse loss}} + \underbrace{\boxed{\lambda_3 R1R2}}_{\text{Minimizes the ellipse size}}\right)$$

- *p'* – the probability of track/ghost was predicted by deep RNN
- *p* – the label that indicates whether or not the set of points belongs to true track
- *x', y'* – the center of ellipse, predicted by network
- *x, y* – the next point of the true track segment
- *R1, R2* – semiaxis of the ellipse
- $\max(\lambda_1, 1 - p), p$ - coefficients that weights classification and regression parts, e.g. we **don't need to search for the continuation of track candidate if it is a ghost**
- λ_{1-3} – weights for each part of equation

$$\mathrm{FL(p, p')} = \begin{cases} -\alpha\,(1 - \mathrm{p'})^{\gamma} \log(p') & \text{if } \mathrm{p} = 1 \\ -(1 - \alpha)\, p'^{\gamma} \log(1 - p') & \text{otherwise} \end{cases}$$

FL is a **balanced focal loss** with a weighting factor $\alpha \in [0, 1]$ – common method for addressing class imbalance. We set $\alpha = 0.95$**,** The focusing parameter γ (we set it to 2) smoothly adjusts the rate at which easy examples are down-weighted.

TrackNETv1 thinking out loud

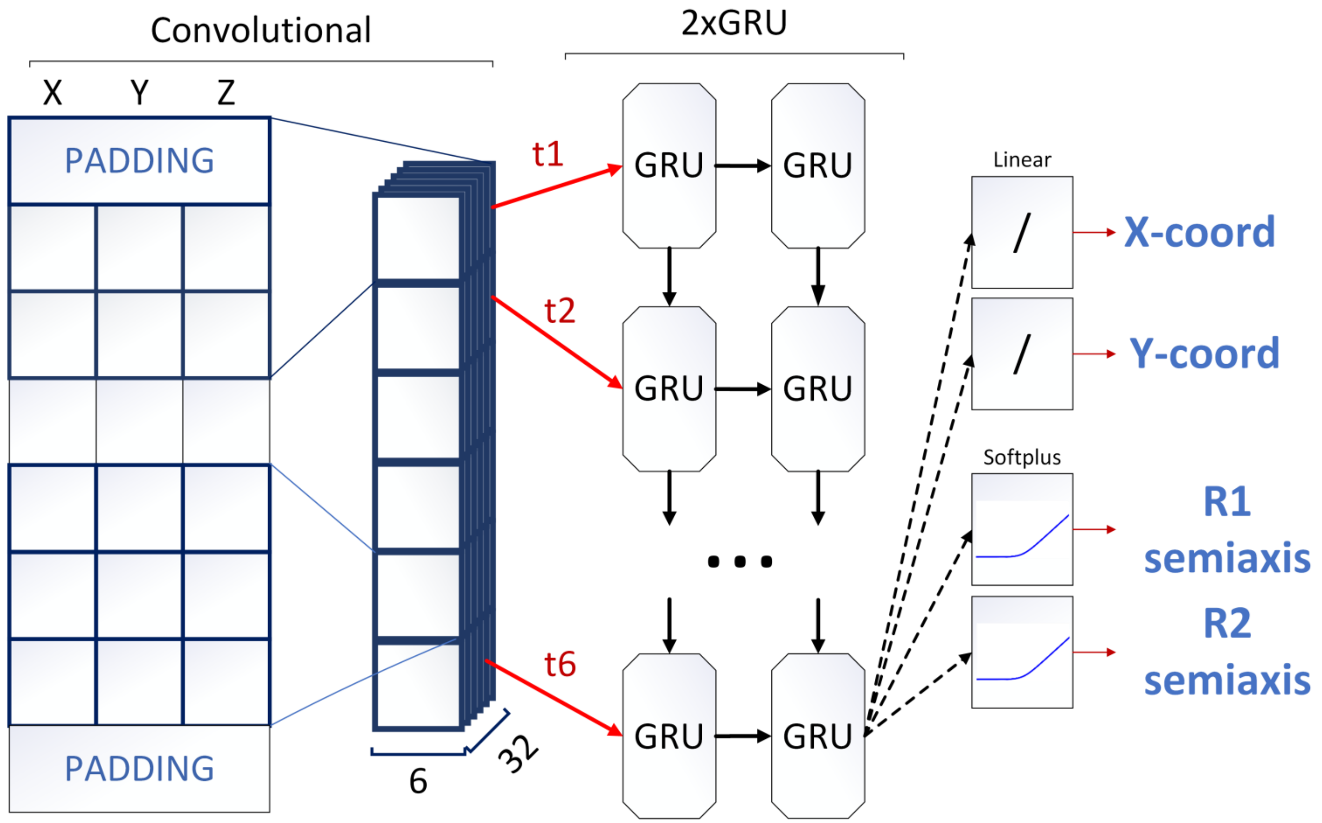
- The **TrackNETv1 requires a labeled dataset with true and fake tracks**, which may be obtained with the help of the first stage of the two-step approach, i.e. directed spatial track-candidates search. **Preparing such a dataset takes a lot of efforts and time**.
- **We cannot use a single model to solve the tracking problems up to the hilt**, because of the cohabitation of the classification part and the regression part, in which one of each have to turn off depending on the inputs' length. Meaning that for seeds with two points we can't set any label of true track appearance and for seeds with the length of a number of stations we don't know the next hit.
- While dealing with great data imbalance, we invented the special loss function, which is fully controlled by the set of **five hyperparameters needed for careful tuning**.

Taking into account the statements above, we found that we can drop the classification part at all because the ellipse prediction comprises the track smoothness criterion by itself. By removing the classification part we open the opportunity **to train a single model end-to-end** using **only true tracks**, which can be accurately extracted from Monte-Carlo simulation. Also, by removing the classification part we decrease the number of the loss hyperparameters to be optimized to two lambdas. The loss' simplification **brings more stability to the training process**.

TrackNETv2

https://github.com/Kaliostrogoblin/TrackNet_v2/tree/laqgsm_data

Model Architecture



Loss

$$L = \lambda_1 \sqrt{\left(\frac{x - x'}{R1}\right)^2 + \left(\frac{y - y'}{R2}\right)^2} + \lambda_2 R1 R2$$

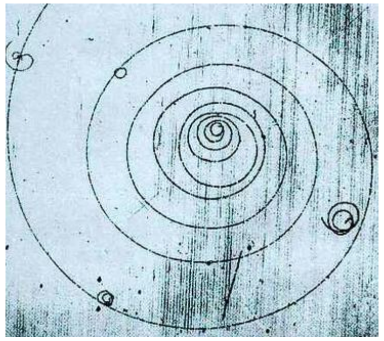
- *x', y'* – the center of ellipse, predicted by network
- *x, y* – the next point of the true track segment
- *R1, R2* – semiaxis of the ellipse
- λ_{1-2} – weights for each part of equation

We set up λ_1 with 0.9 and λ_2 with 0.1 values, respectively.

Training data preparation

To prepare the dataset, we were guided by the events of C+C interactions, specific for BM@N run 2017

1) Simulate 550k events with the energy of 4GeV using LAQGSM generator
2) Remove tracks containing less than 3 hits
3) Drop spinning tracks (more than 1 hit per station)
4) Label hits by seeking for the corresponding Monte-Carlo point
5) Take only true tracks for training from the 8 files with 50K events per file
6) For evaluation phase – 150K events



Eventually, we have three data sets for:

- **Training** - 1 405 556 tracks
- **Validation** – 351 388 tracks
- **Evaluation** – 150K events with fakes presence

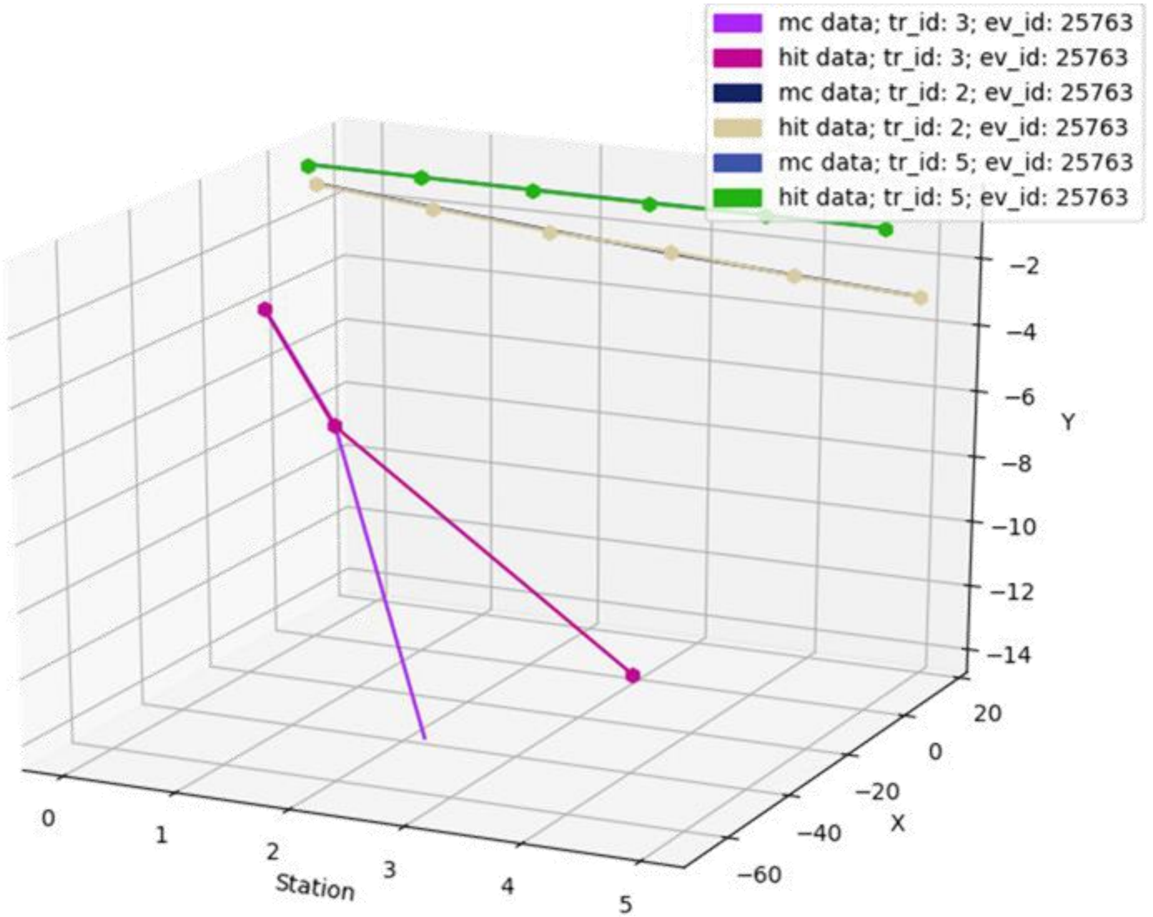
Problems we faced during data preparing

1) Events with an anomalous number of hits per station

Hits per station statistics (50K events)	
mean	22.0760
std	103.7415
min	1.0000
25%	4.0000
50%	10.0000
75%	20.0000
max	9507.0000

Solution: no work required

2) Corrupted events, where the nearest hit is too far from corresponding Monte-Carlo point



Solution: drop events where the distance between some MC point and corresponding hit is > 1

3) Nearest neighbor search is too slow when a spatial index is applied

Solution: vectorizing brute-force search

Training setup

We trained separately three models depending on the presence of the vertex data:

1. model, which uses data with **fixed vertex** (0, 0, -21.9);
2. network with inputs, where **each coordinate of the vertex in each sample is randomly generated**:
   - X and Y coordinates were sampled from the normal distribution $[\mathrm{X}, \mathrm{Y}] \sim N(0.0, 2.5)$
   - Z – from the uniform distribution $Z \sim U(-21.0, -21.9)$
3. and one, that has **no prior knowledge about vertex location**.

All three models were trained

- for **50 epochs**
- with **Adam** optimizer
- with **gradient clipping on 1**
- and **batch size = 256.**

Validation

As we have one single model, we can validate it by checking **for how much of the tracks** from validation set (351 388) **model can correctly predict the continuation** starting from the two points to the last hit of the track. **For that purpose we have used only tracks with the length equals to the number of stations.**

We measured:

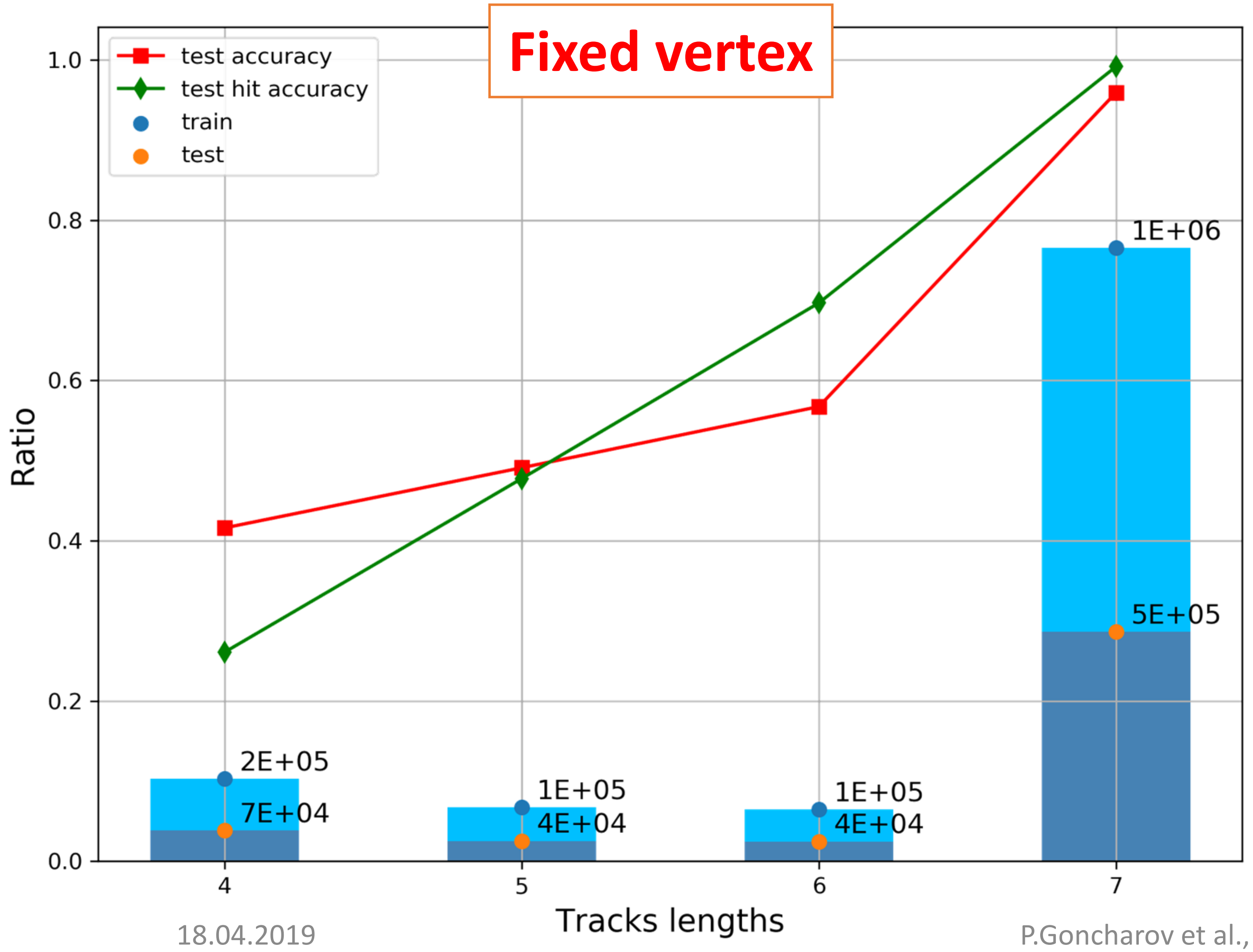
- **accuracy**, i.e. fraction of the tracks, which model can reconstruct without being mistaken;
- **hit accuracy** – the proportion of the hits predicted correctly;
- **processing speed** – how much time elapsed before the model serves all tracks

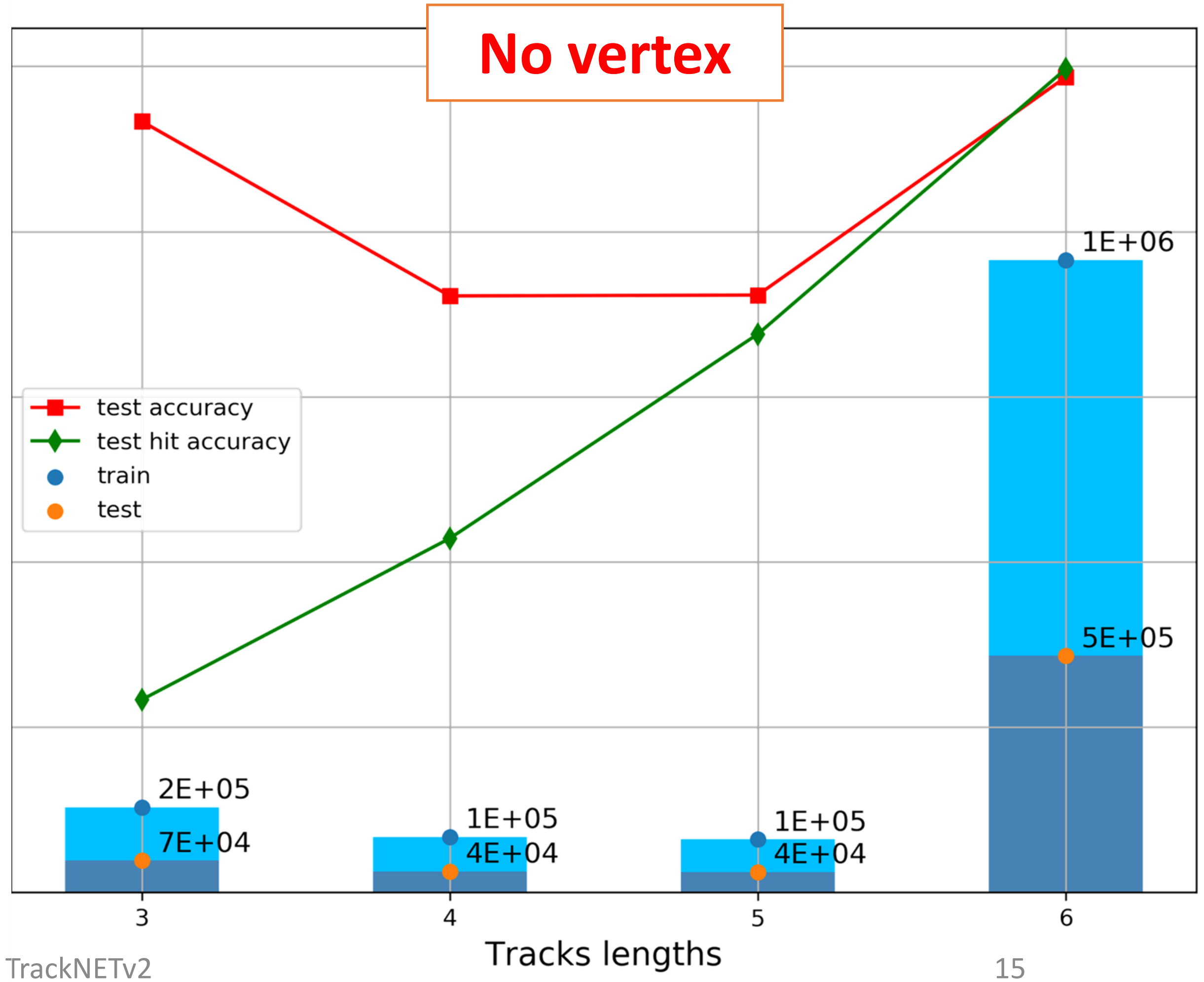
	Fixed vertex	**Sampled vertex**	**No vertex**
Accuracy	0.9593	0.9620	**0.9870**
Hit accuracy	0.9918	0.9923	**0.9967**
Processing speed (tracks/sec)	12128.59	12916.72	**16903.92**

Note: processing speed were measured on Intel Core i3-4005U @1.70 GHz with batch size = 512

Data imbalance disclosure

We observed that our **models performs poorly on «short» tracks with low energy**, eventually, we found out that **the fraction of the «long» tracks is much higher** than of the «short» ones.





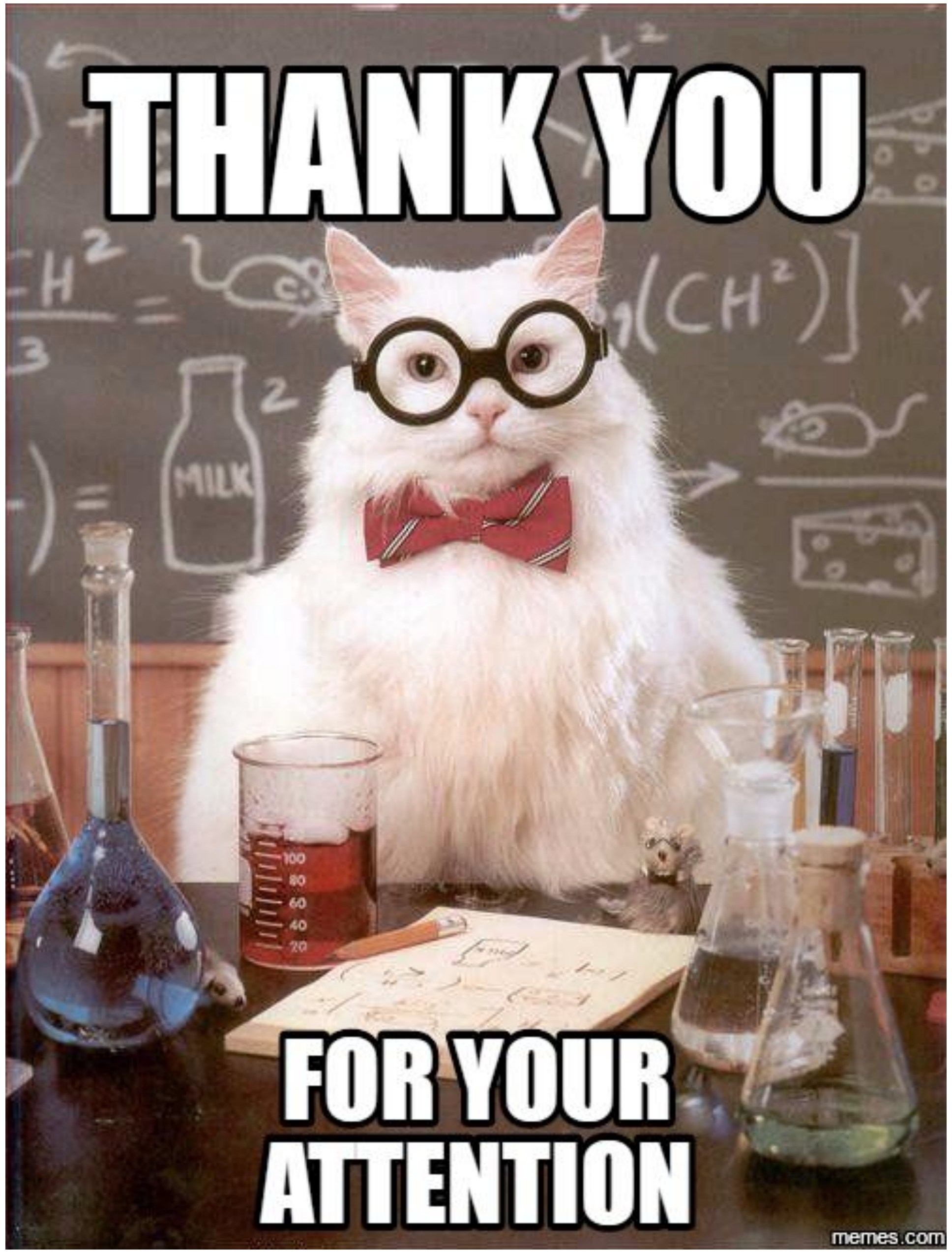
Conclusion and Outlook

We have improved the first version of the TrackNET and have presented the TrackNETv2, which is

- fully end-to-end trainable;
- does not require the tremendous fake tracks preparing via directed search;
- has a more stable training process
- and fewer parameters for tuning due to the lack of the classification part;
- memory cheaper because does not consist of three parts;
- can process around 16K tracks/sec $\approx$ 727 events/sec on a simple laptop

Now we are going to

- improve the efficiency for the tracks with low energy;
- try out convolutional structure instead of the recurrent;
- add more features to the input of the TrackNETv2;
- port the model and the code to a C++ package;
- expand the model's powers to solve tracking in a collider environment.


THANK YOU
FOR YOUR ATTENTION
memes.com