The HEP.TrkX project: deep learning solutions for particle tracking

Steve Farrell,
on behalf of the HEP.TrkX collaboration

HSF meeting
Sep 26, 2018

Ernest Orlando Lawrence
Berkeley National Laboratory

Introduction

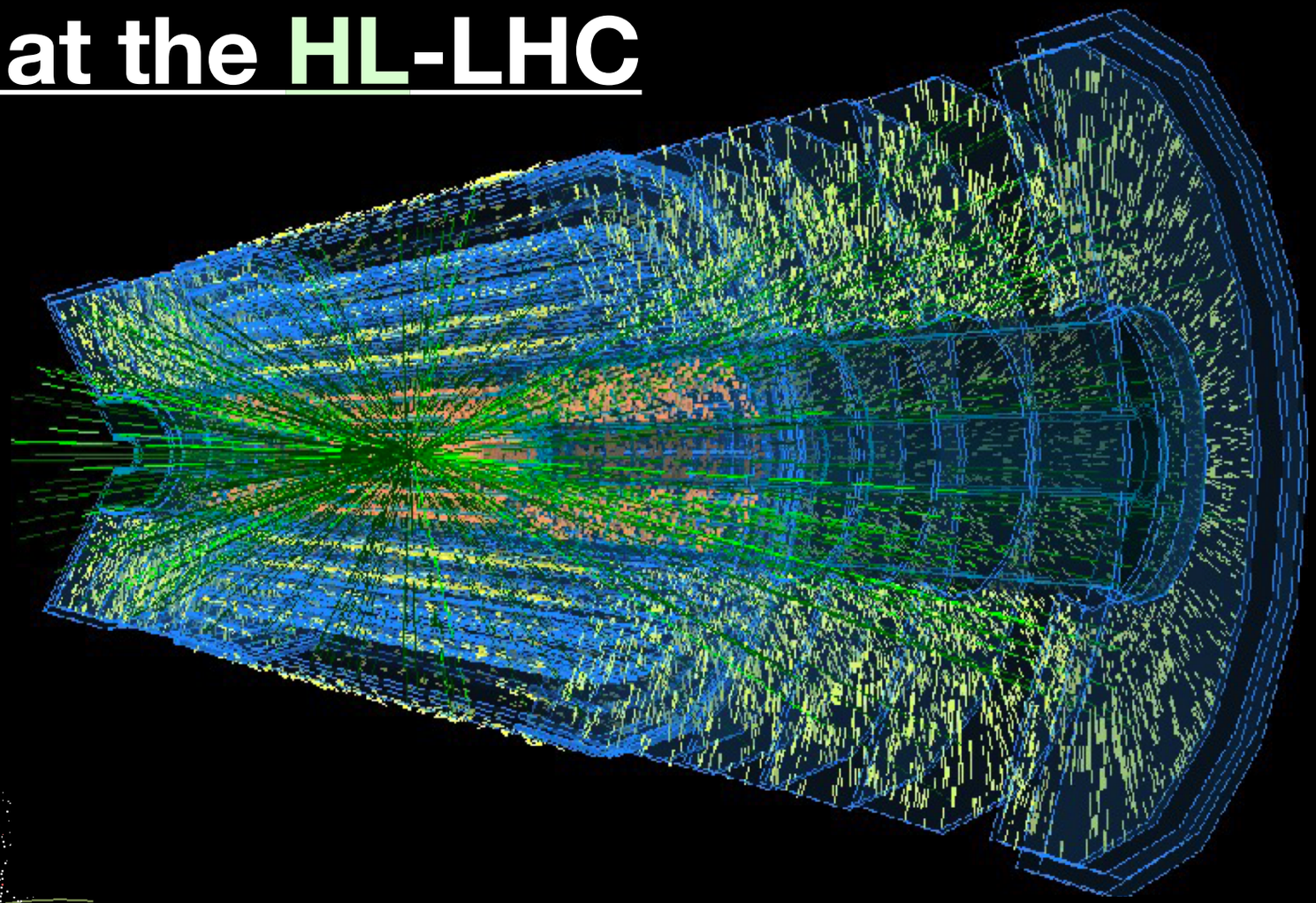
- **Deep Learning is working its way into everything, even in HEP**
  - Object identification, event selection, fast simulation, …
- **In track reconstruction, we face unprecedented challenges in the HL-LHC era**
  - Algorithmic challenges
  - Computing resource shortage
- **Can Deep Learning techniques be useful on these types of problems as well?**

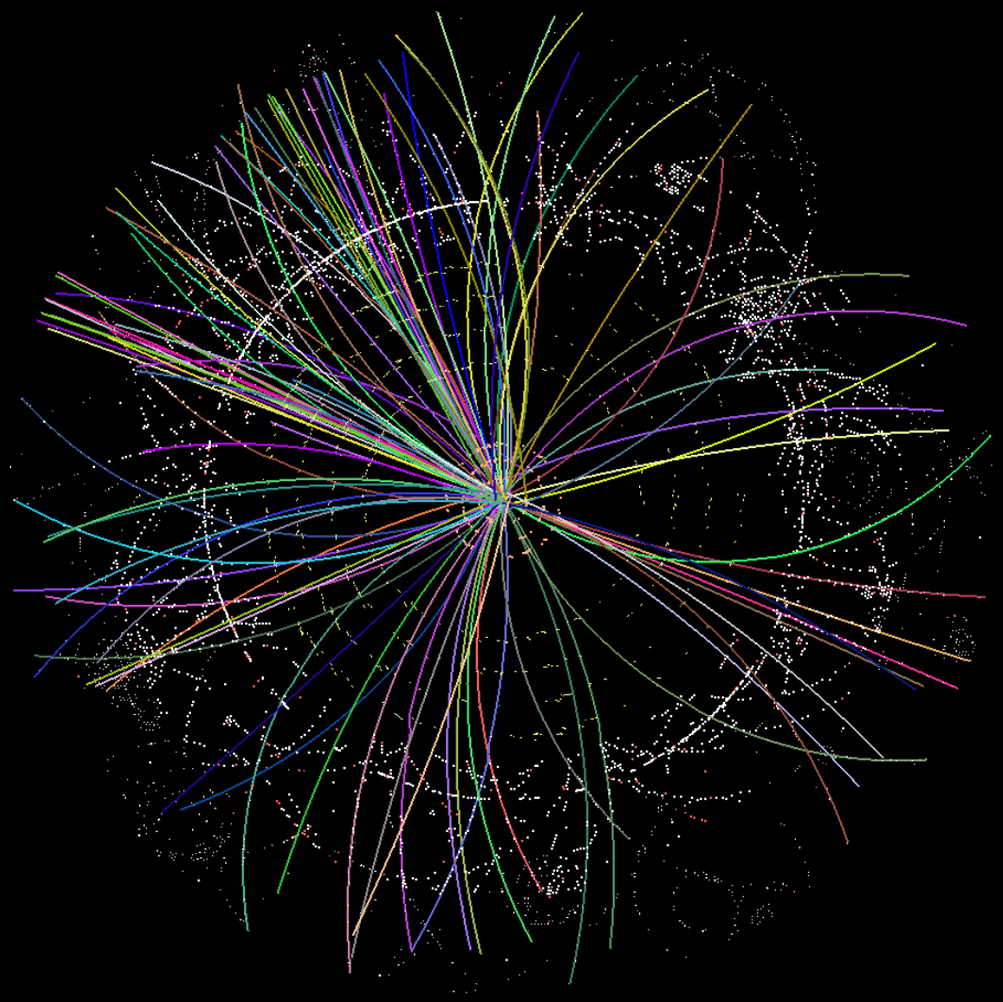
Outline

- The HL-LHC tracking problem
- Engaging broader community with TrackML challenges
- The HEP.TrkX project
- Image-based methods for track finding
- Track building from spacepoints and graphs

Particle tracking at the HL-LHC

Identify the particle trajectories in each collision event

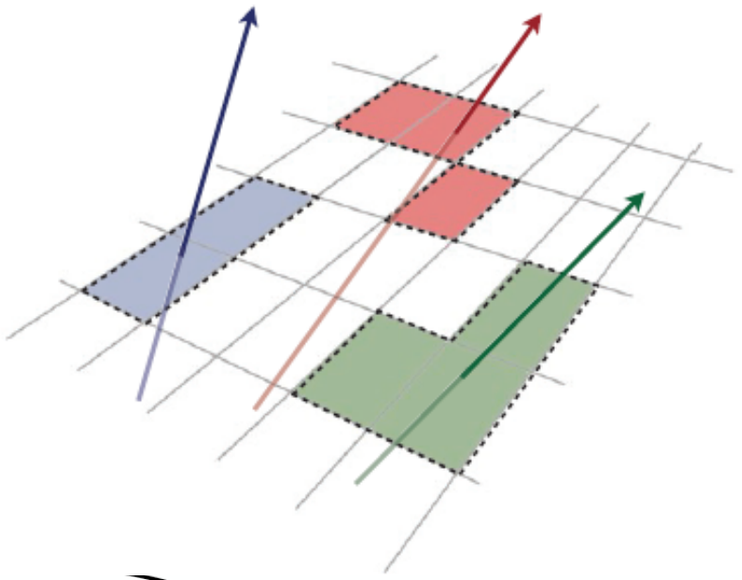


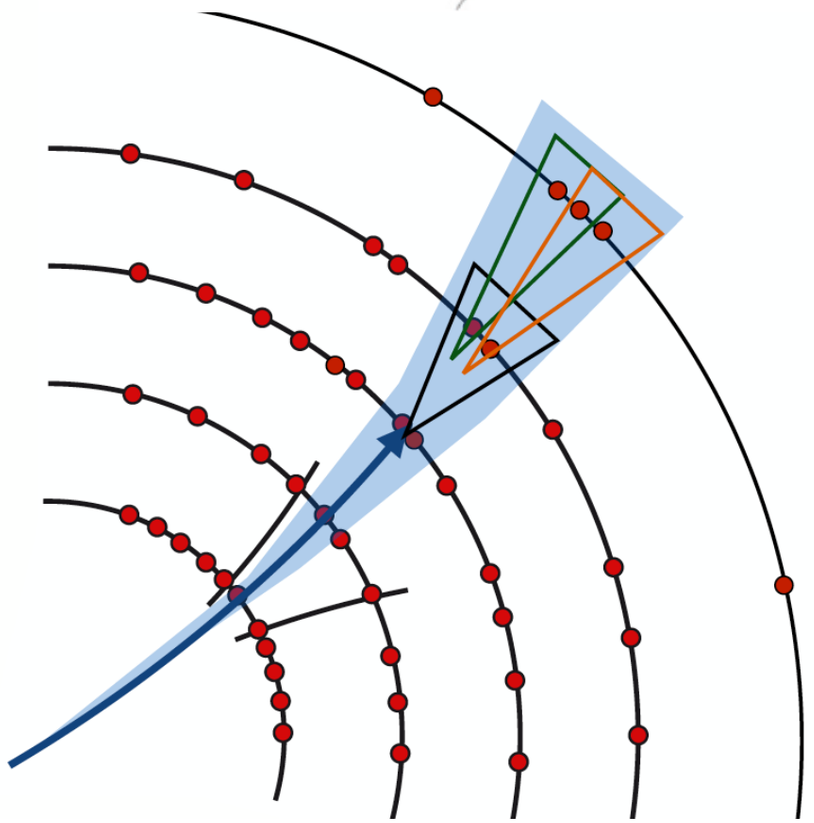


- **Record “hits” in high-granularity detectors**
  - 100M channels
- **Connect the dots to find tracks**
  - *O*(10k) tracks, *O*(100k) hits

The current situation

- **What do we do *today*?**
  1. Cluster detector signals to form hits
  2. Construct triplets of hits (“seeds”)
  3. Extend seeds to build tracks with a combinatorial Kalman Filter
  4. Fit the trajectories
- **What’s the problem?**
  - Scales poorly to HL-LHC conditions
  - Severe CPU shortage expected
- **What’s the solution?**
  - Machine learning…?





Credit: Andy Salzburger

The TrackML Challenge

https://sites.google.com/site/trackmlparticle/

- **Can the broader data science community help HEP develop (ML) solutions for particle tracking?**
  - Side benefit: can we standardize the tracking problem and dataset?
- **HEP challenges have been popular in the past**
  - Higgs ML Kaggle Challenge
  - Flavour of Physics Kaggle Challenge
- **A tracking challenge is a bit trickier in certain ways**
  - Complex data and physics metrics
  - Compute vs. physics performance tradeoffs

David Rousseau, Sabrina Amrouche, Paolo Calafiura, Victor Estrade, Steven Farrell, Cécile Germain, Vladimir Vava Gligorov, Tobias Golling, Heather Gray, Isabelle Guyon, Mikhail Hushchyn, Vincenzo Innocente, Moritz Kiehn, Edward Moyse, Andreas Salzburger, Andrey Ustyuzhanin, Jean-Roch Vlimant, Yetkin Yilmaz

The TrackML Challenge

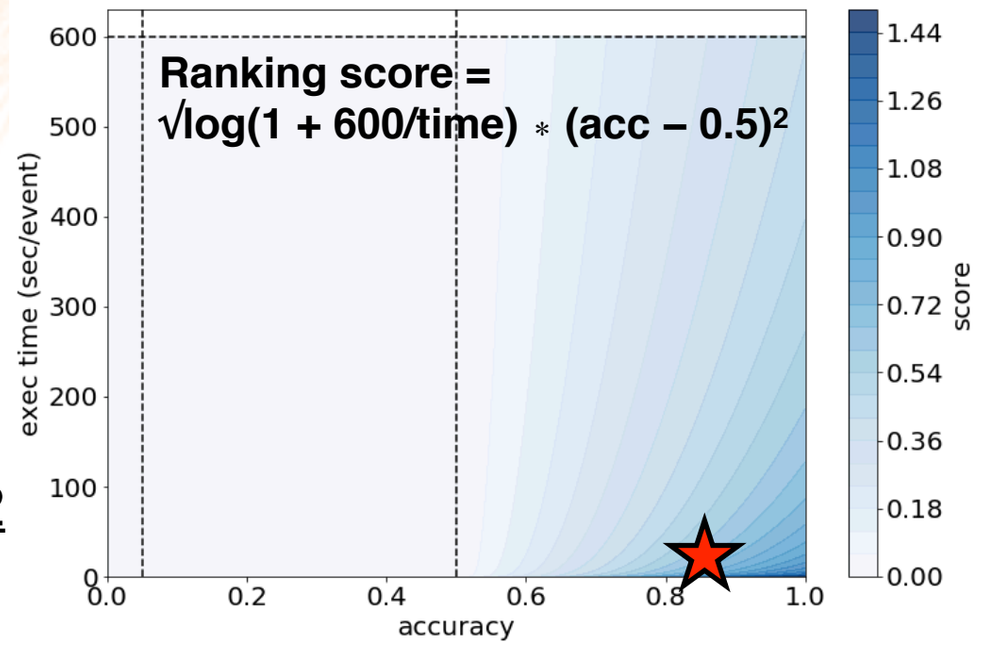
- **Accuracy phase: Kaggle featured competition May - Aug 2018**
  - Reconstruction efficiency scoring metric
  - $25k prize money
  - Final leaderboard:

#	△pub	Team Name	Kernel	Team Members	Score	Entries	Last
1	—	**Top Quarks**			0.92182	10	22d
2	—	**outrunner**			0.90302	9	21d
3	—	**Sergey Gorbunov**			0.89353	6	21d

656	776	5,837
Teams	Competitors	Entries

https://www.kaggle.com/c/trackml-particle-identification

- **Throughput phase: Codalab competition Sep - Oct 2018**
  - Scored by both reconstruction efficiency and throughput
  - Official NIPS competition!
  - Current leaders are at 6-15s for 85-90% accuracy



https://competitions.codalab.org/competitions/20112

The HEP TrkX Project

https://heptrkx.github.io/

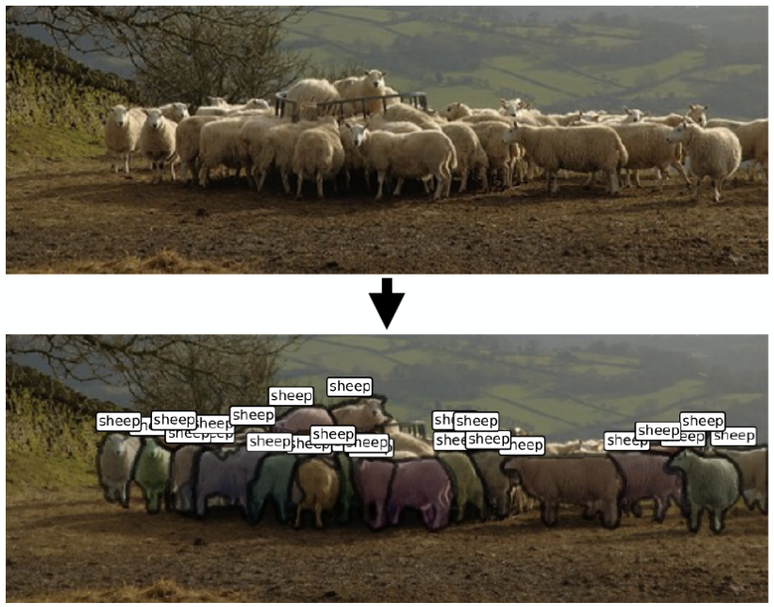
- **A pilot project to develop machine learning algorithms for HEP tracking**
  - Funded by DOE ASCR, COMP HEP; part of HEP-CCE

LBL: Steve Farrell, Mayur Mudigonda, Prabhat, Paolo Calafiura
Caltech: Dustin Anderson, Jean-Roch Vlimant, Josh Bendavid, Maria Spiropulu, Stephan Zheng
FNAL: Aristeidis Tsaris, Giuseppe Cerati, Jim Kowalkowski, Lindsey Gray, Panagiotis Spentzouris

- **We've been trying lots of ideas to see what sticks**
  - **Representations**: detector images, hit sequences, hit graphs
  - **Architectures**: CNNs, RNNs, Graph NNs
  - **Problems**: single/multi track building, vertex finding
  - **Datasets**: toy data, ACTS generic detector, TrackML dataset
- We're not the only players in town, see ML tracking session from CHEP: https://indico.cern.ch/event/587955/sessions/266675/#20180712

Some deep learning inspirations

Image segmentation



https://arxiv.org/abs/1604.02135

Our goal (more or less…):

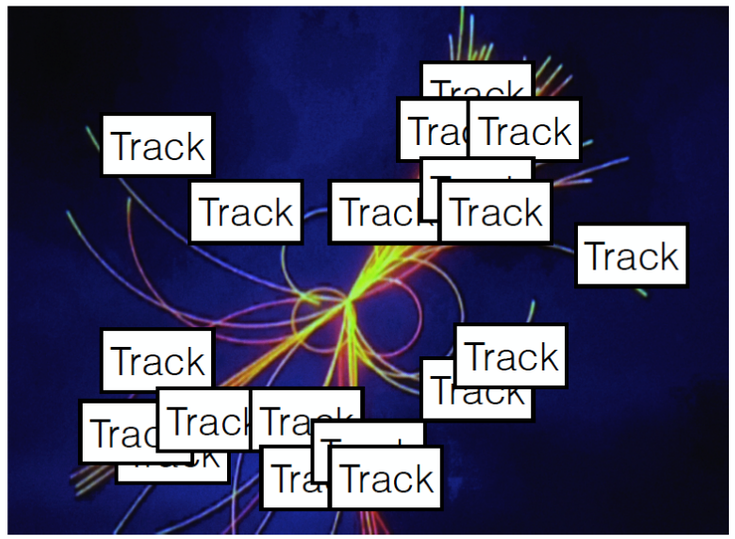
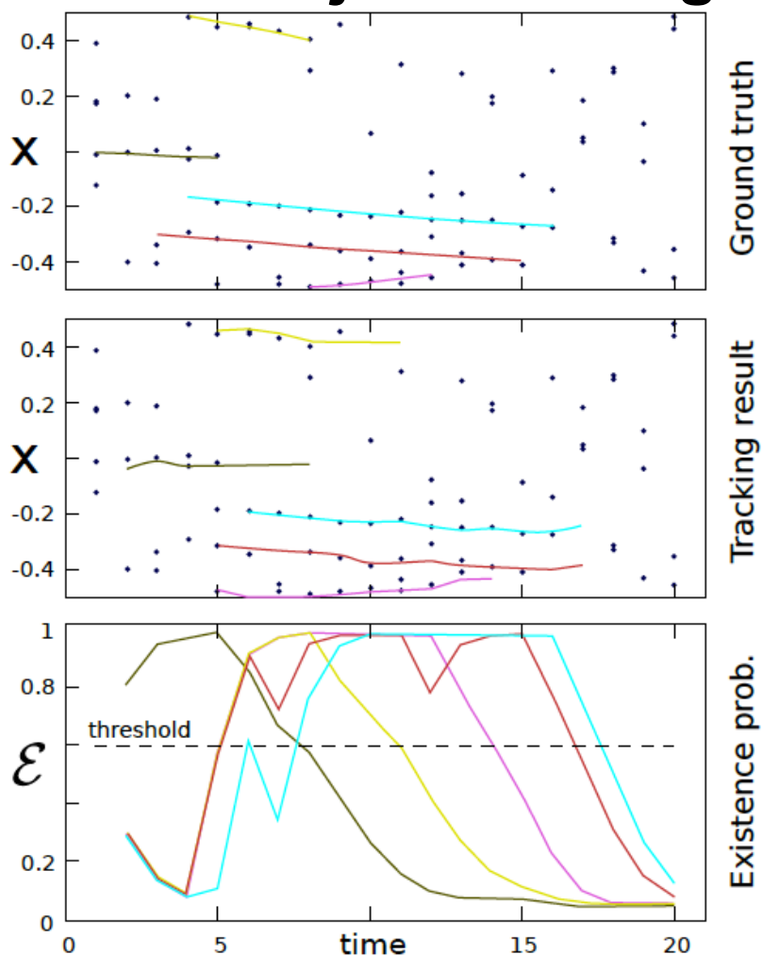
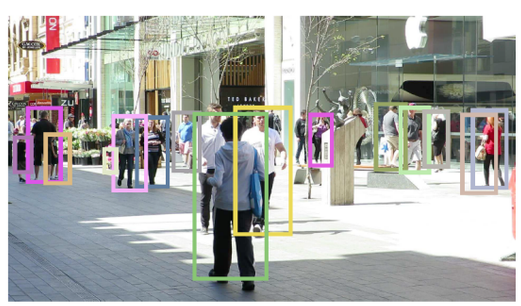
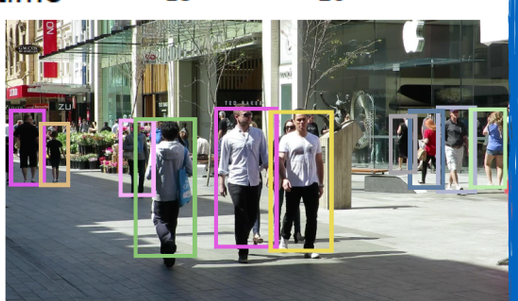


Photo by Pier Marco Tacca/Getty Images

Online object tracking

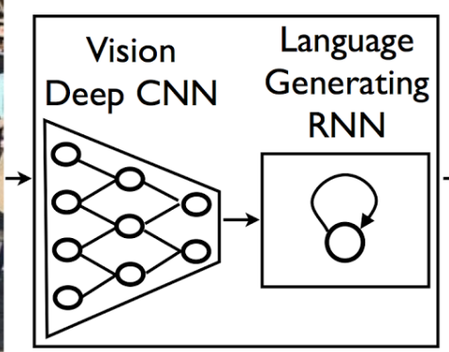






https://arxiv.org/abs/1604.03635

Image captioning

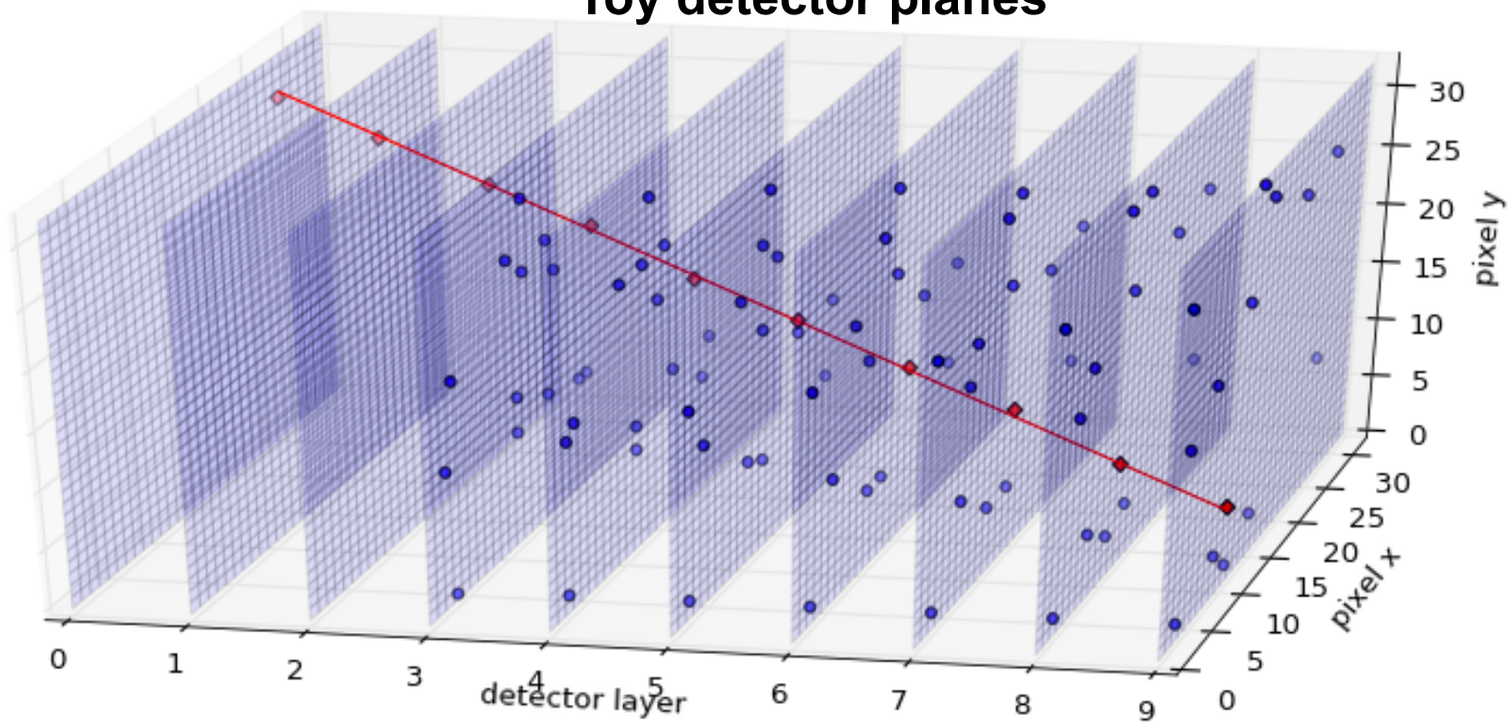


A group of people shopping at an outdoor market.

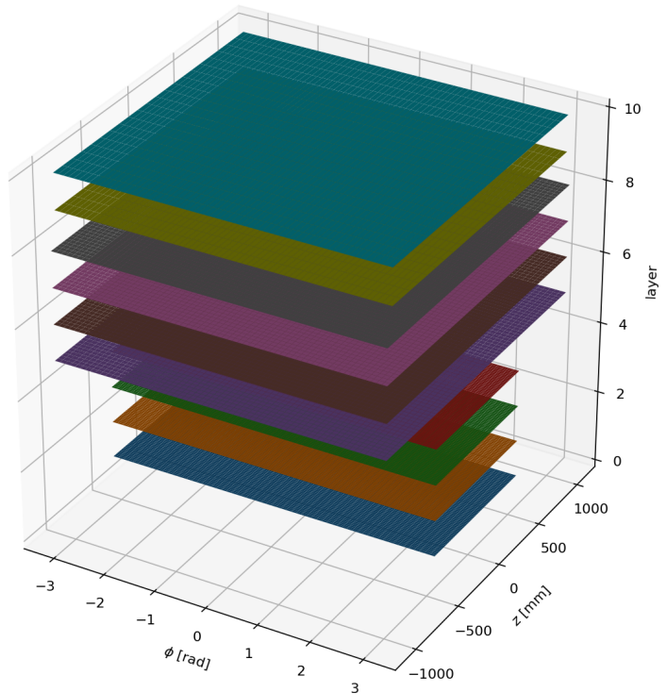
There are many vegetables at the fruit stand.

Image representations

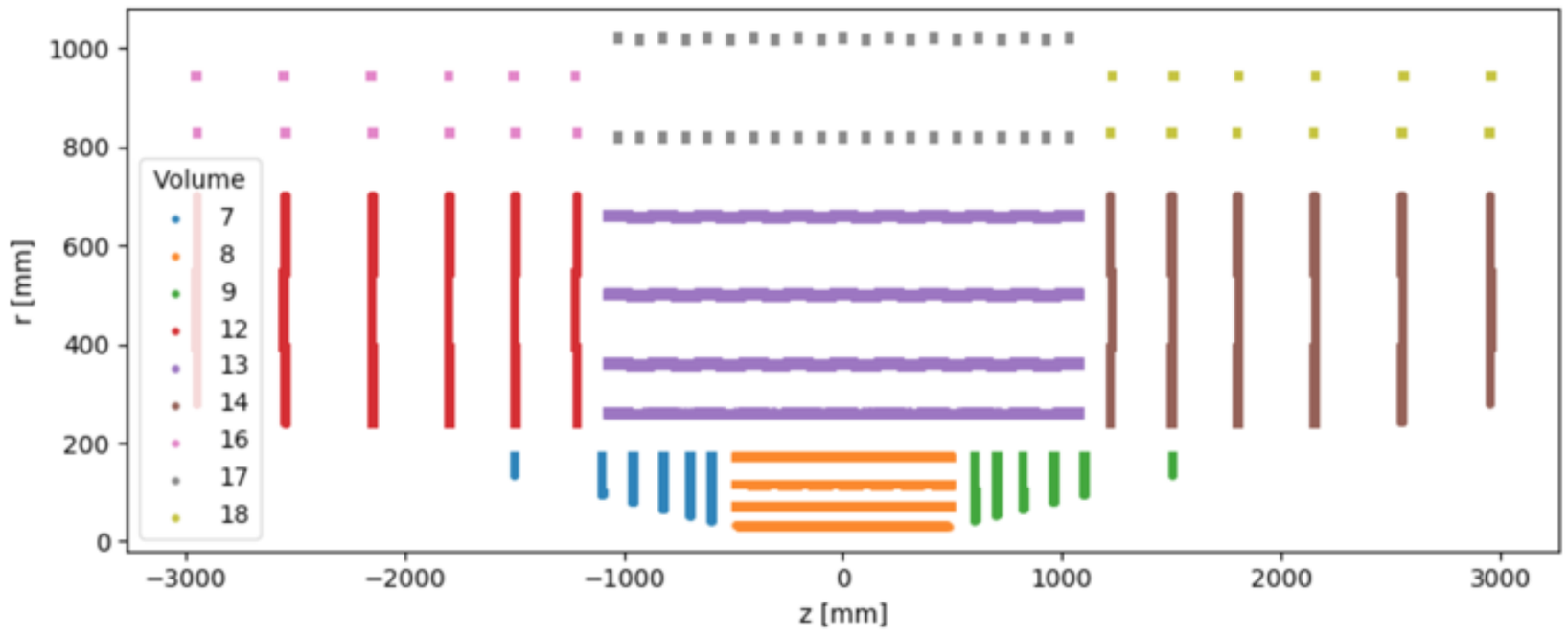
Toy detector planes



ACTS unrolled barrel



ACTS detector geometry



Z and phi projections

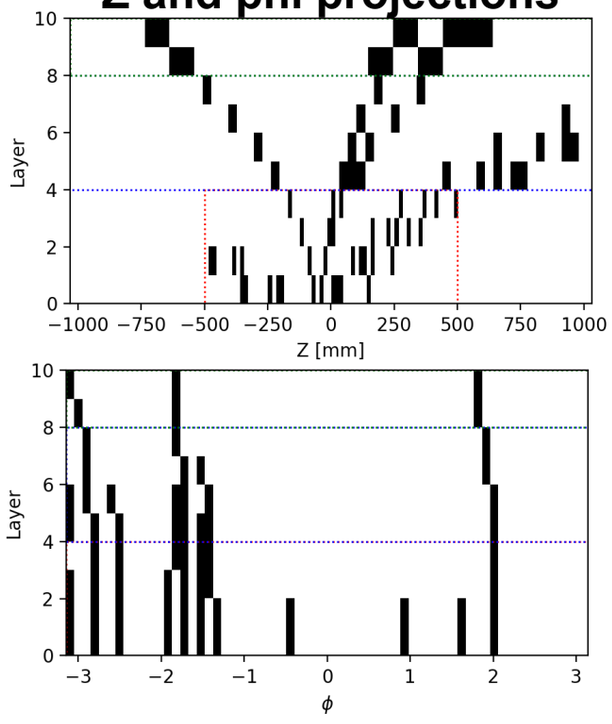
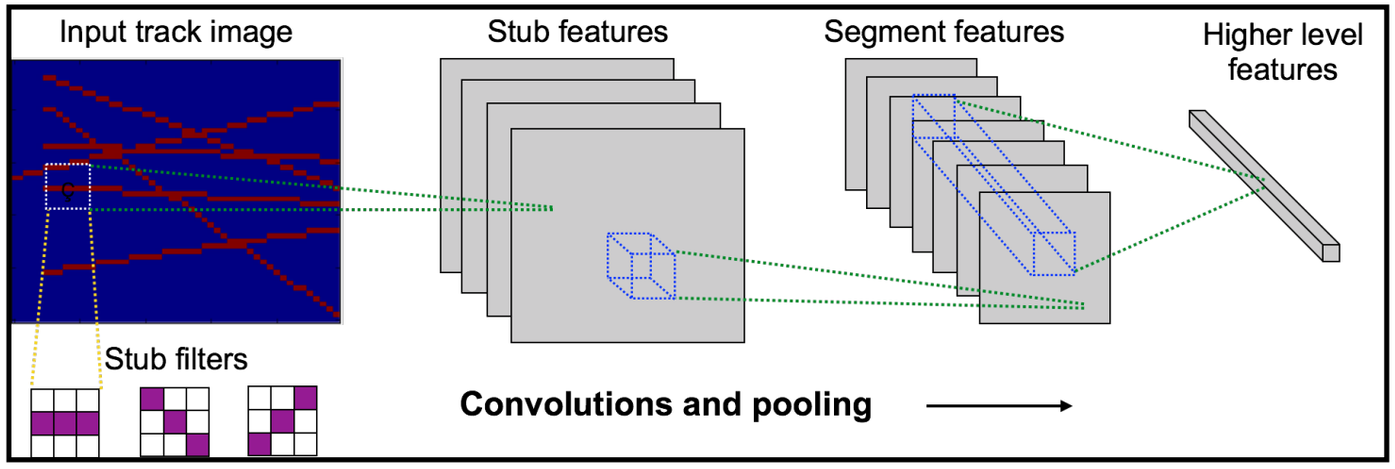
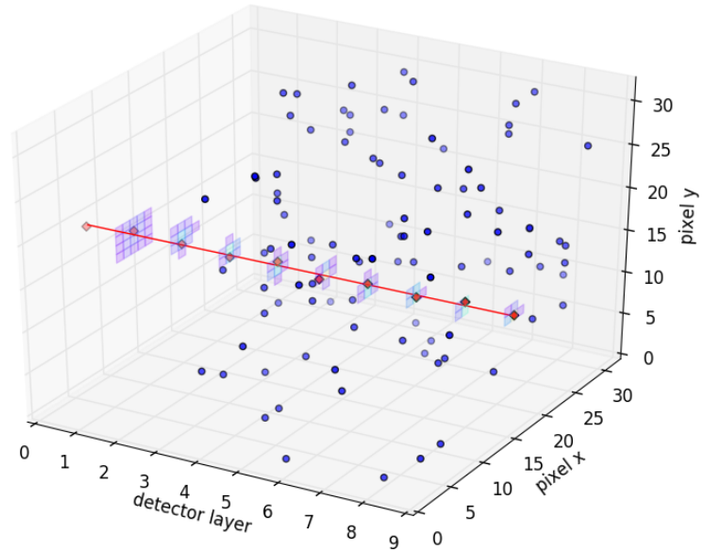


Image-based methods (more in backup) https://heptrkx.github.io/

- **Image segmentation or "captioning" on toy data with LSTMs and CNNs:**

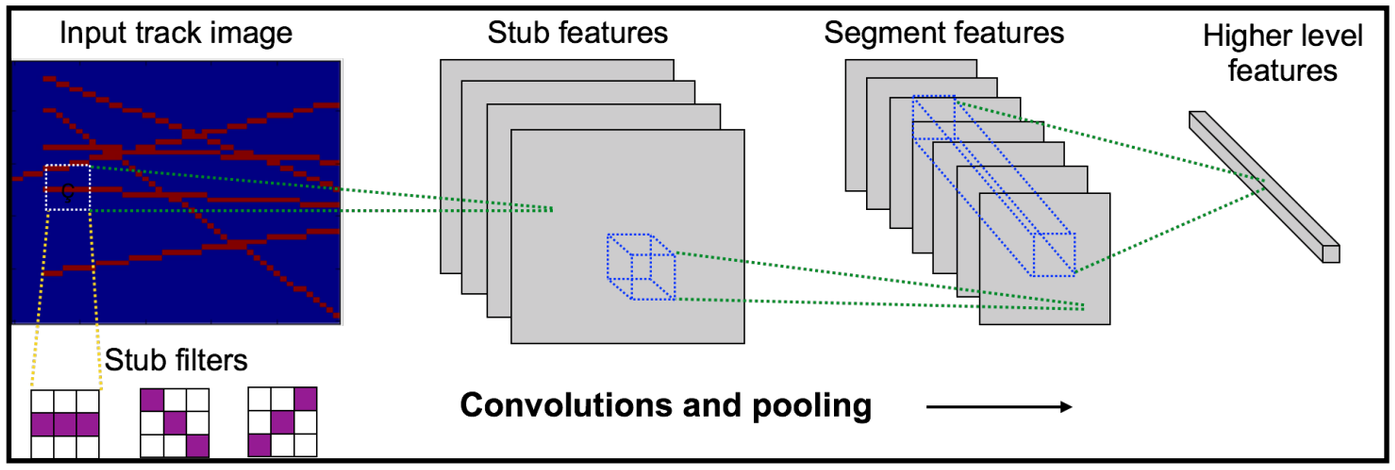


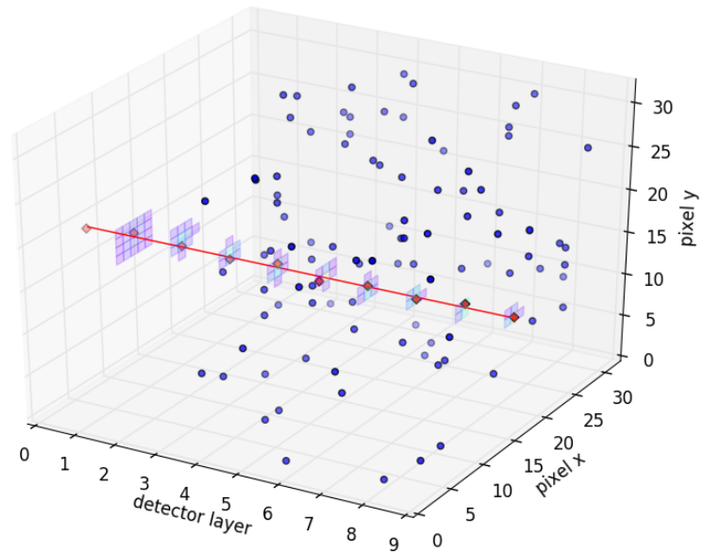


From CTD 2017

Image-based methods (more in backup) https://heptrkx.github.io/

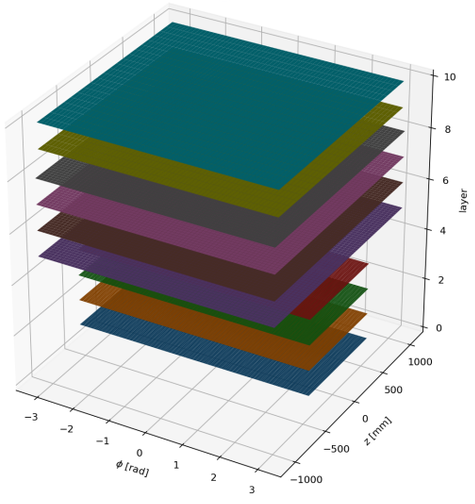
- **Image segmentation or "captioning" on toy data with LSTMs and CNNs:**

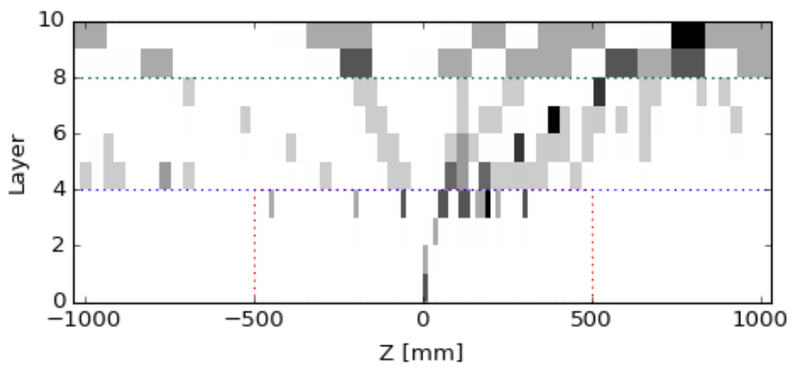


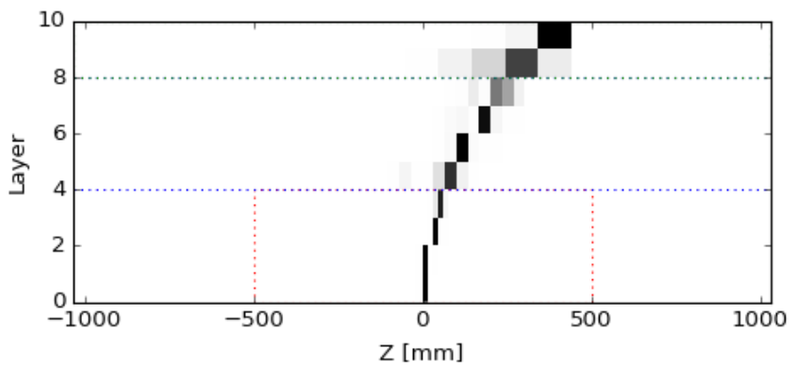


From CTD 2017

- **Segmentation with LSTMs on ACTS data:**



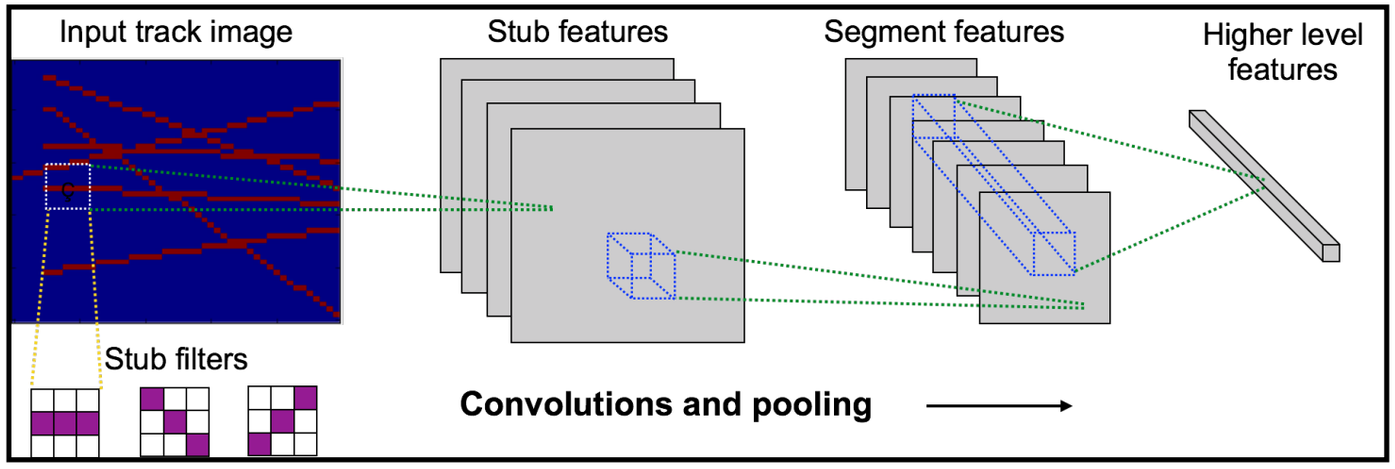


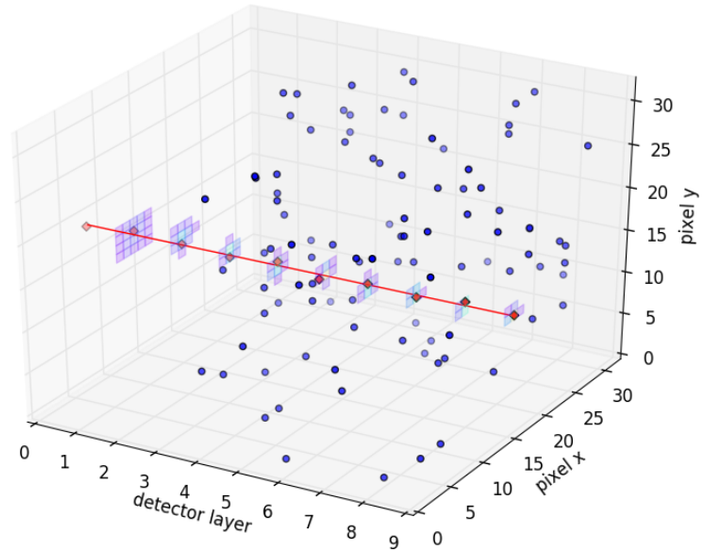


From ACAT 2017

Image-based methods (more in backup) https://heptrkx.github.io/

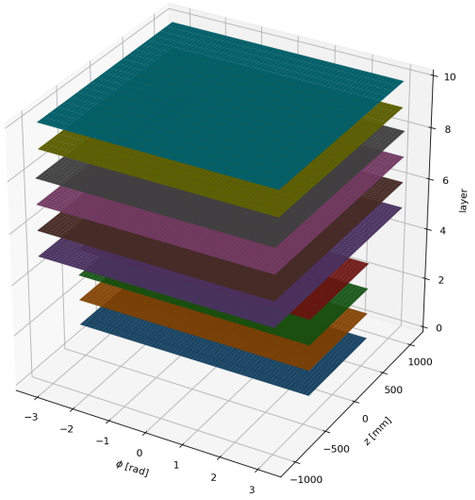
- **Image segmentation or "captioning" on toy data with LSTMs and CNNs:**

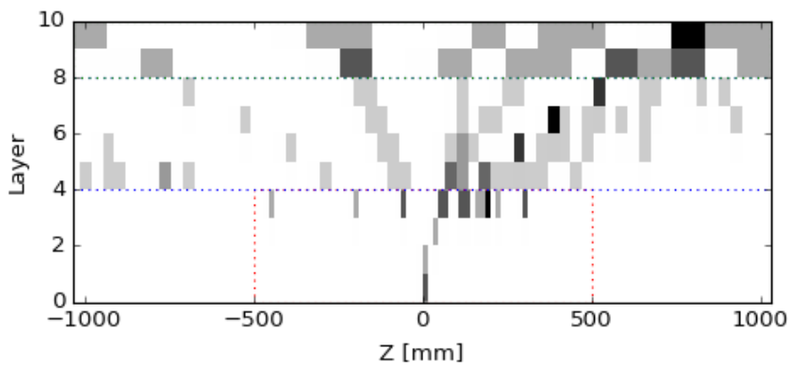


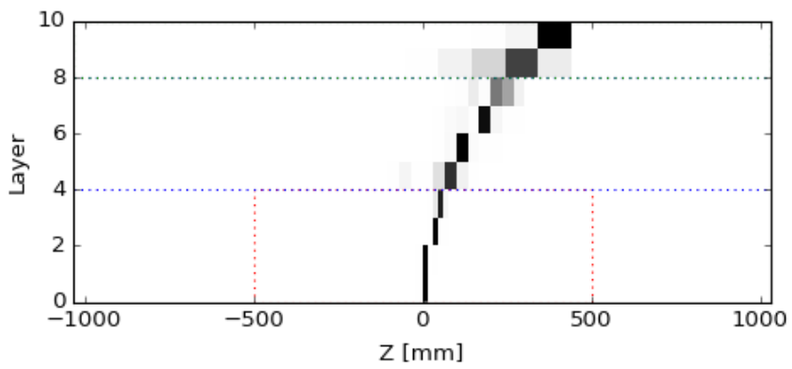


From CTD 2017

- **Segmentation with LSTMs on ACTS data:**



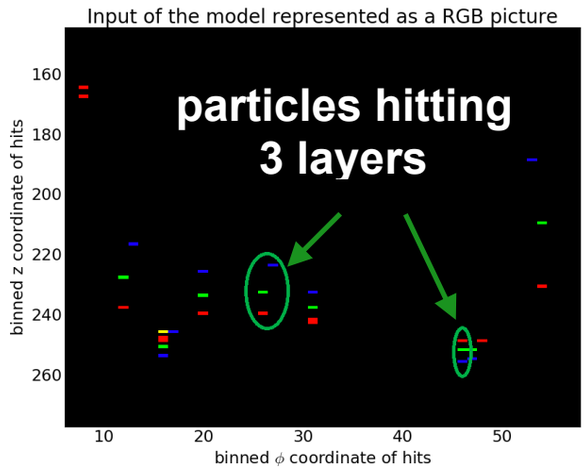


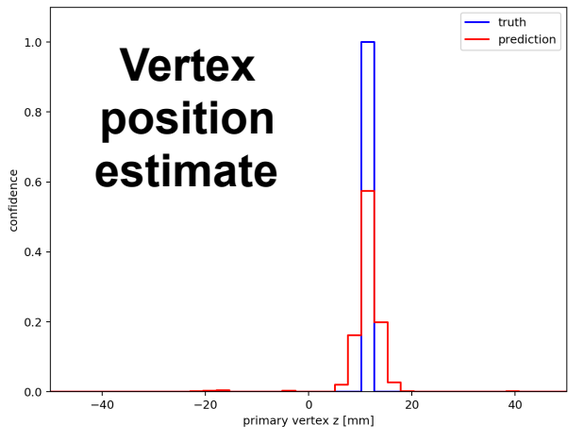


From ACAT 2017

- **Vertex-finding with CNNs**

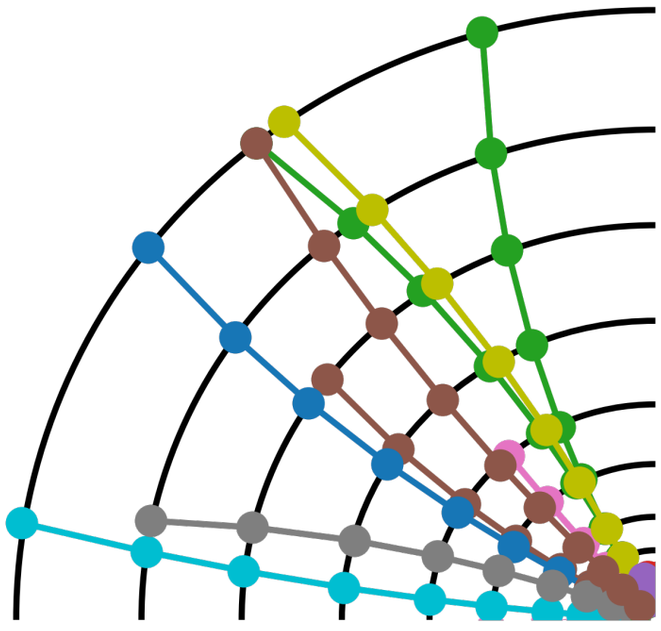
From ACAT 2017

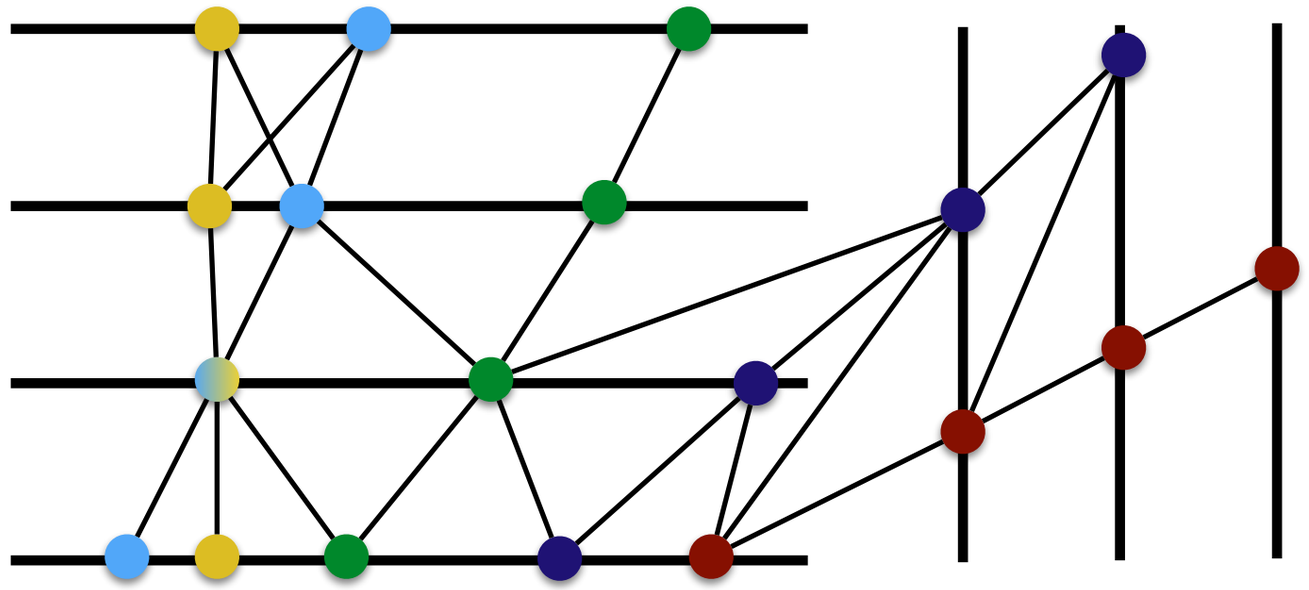




Moving from images to points

- **Image-based methods face challenges scaling up to realistic HL-LHC conditions**
  - High dimensionality and sparsity
  - Irregular detector geometry
- **Instead of forcing the data into an image, we can use spacepoint representations (sequences, graphs)**
  - New challenges for designing models
  - But now we can exploit the structure of the data with full precision



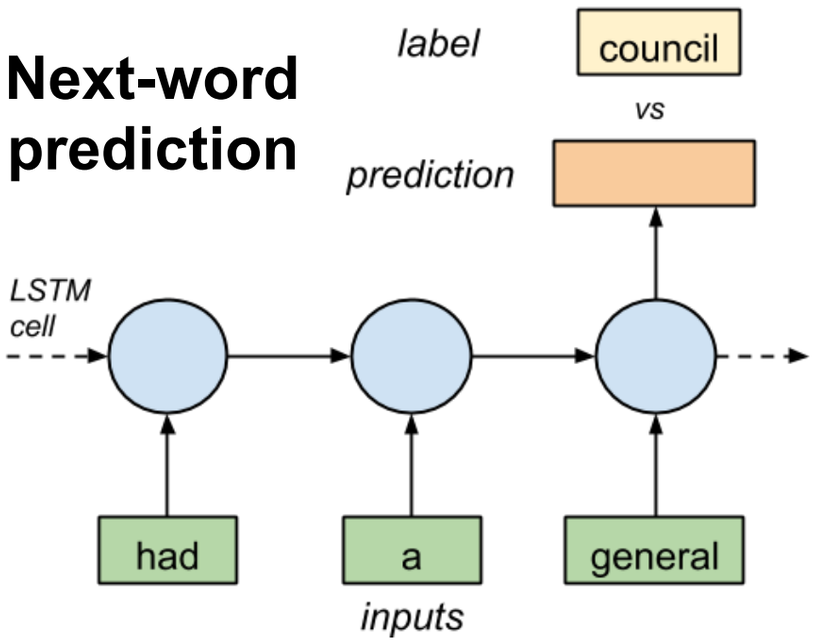


RNNs for track building

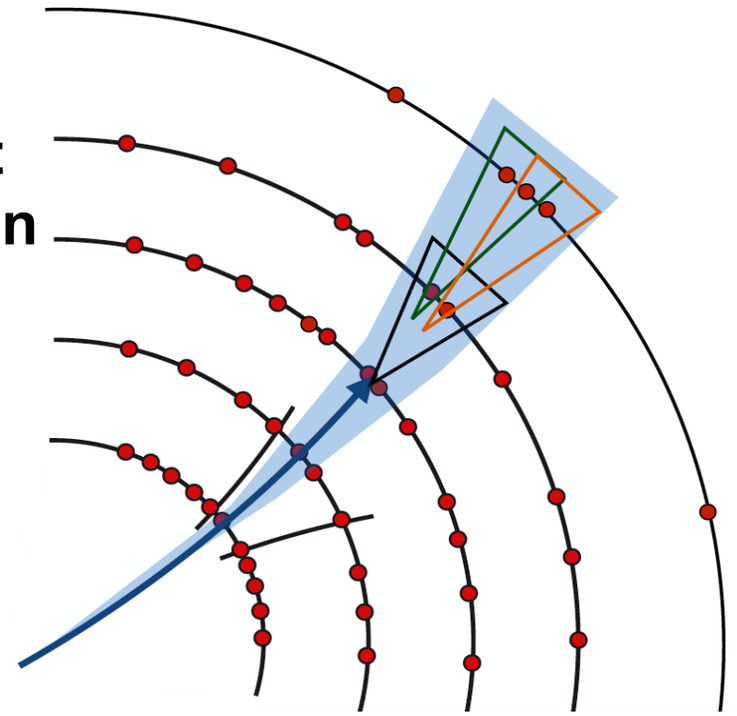
- **Recurrent Neural Networks (e.g. LSTMs) are effective for sequence data problems**
  - In NLP, language translation, sentiment analysis
  - In biology, DNA function predictions
- **Can they be used in track building?**
  - Next-step predictions (like Kalman Filter)
  - Track scoring, fitting

No time to show everything. See backup slides for more RNN applications!

Next-word prediction



Next-hit prediction

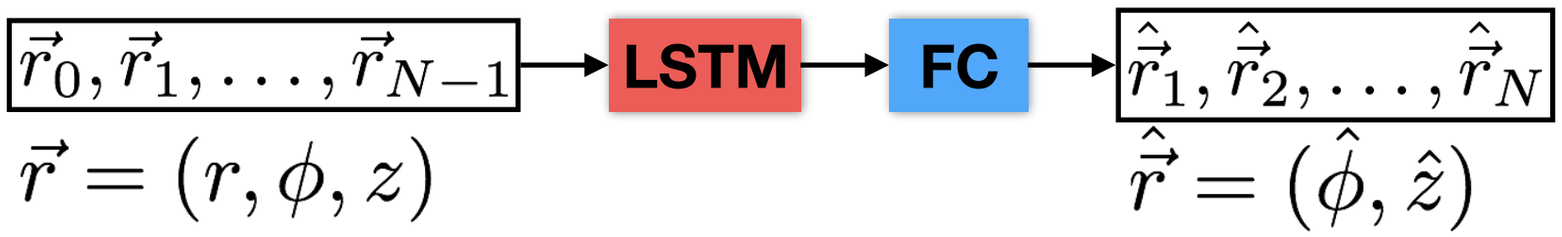


https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537

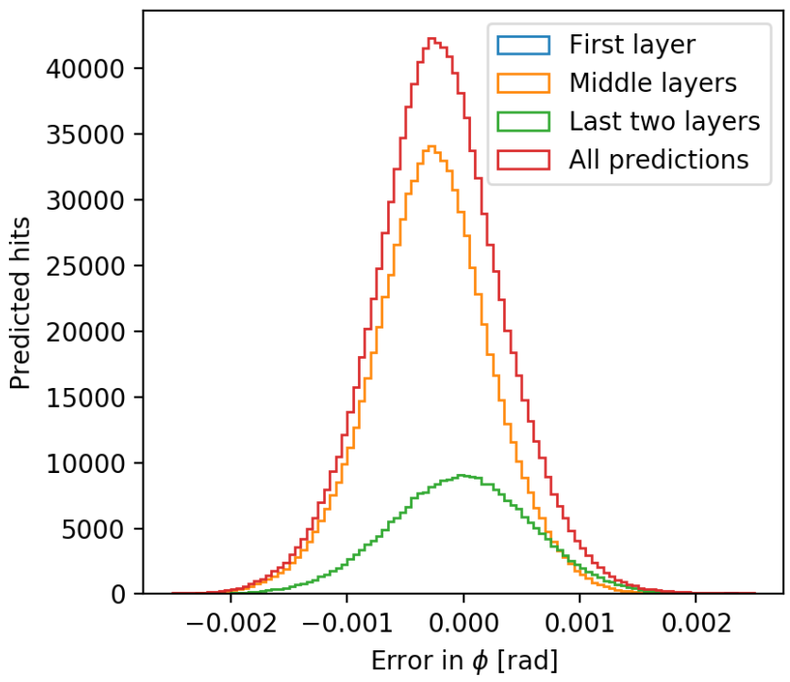
Credit: Andy Salzburger

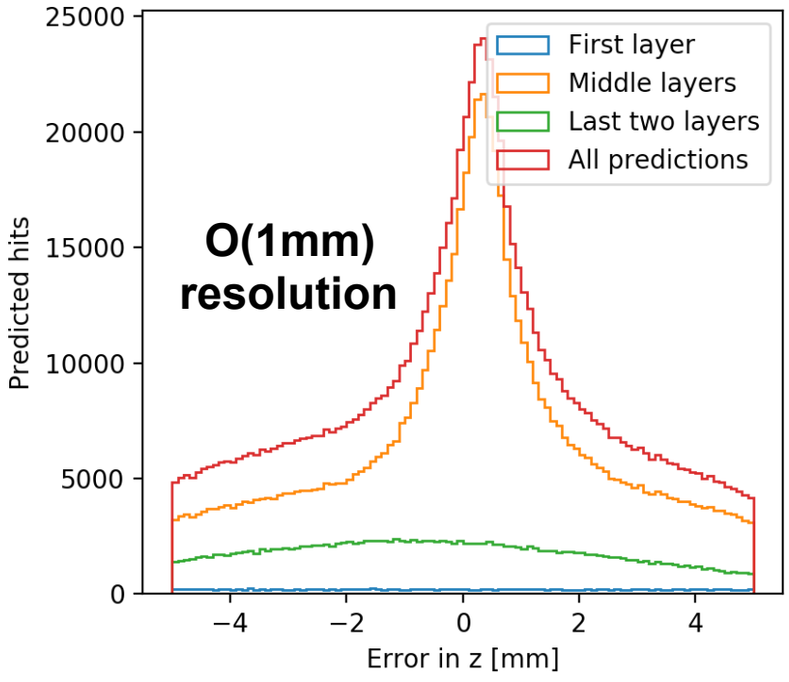
RNN hit predictor model

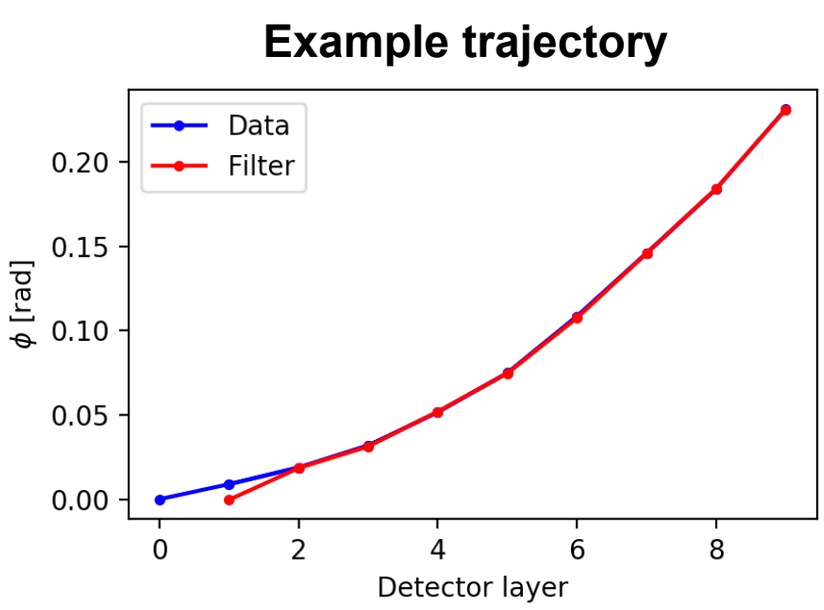
- **Given a sequence of hits, predict coordinates of the next hit**
  - Regression problem with mean-squared-error loss



- **Results on ACTS barrel tracks:**

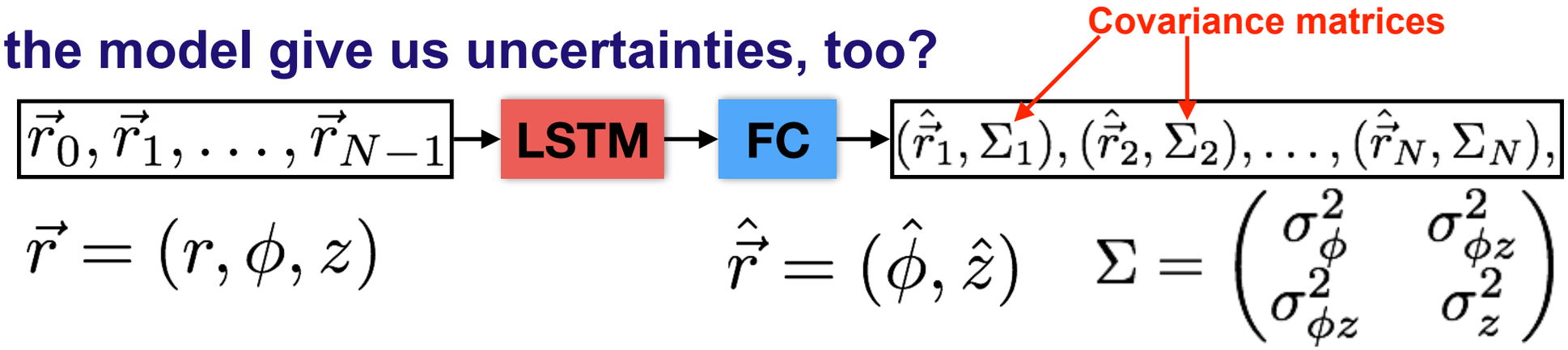






RNN Gaussian hit predictor

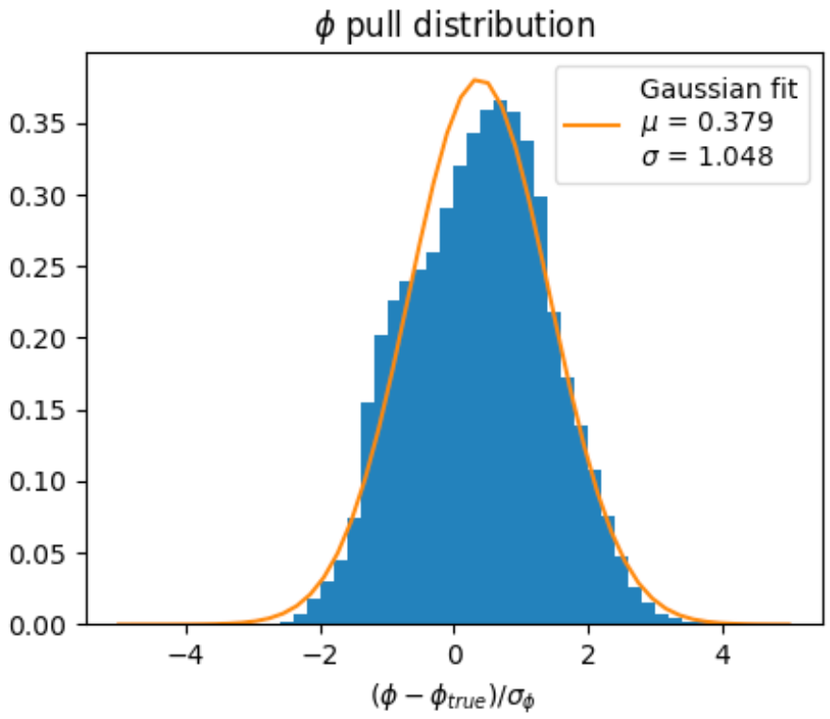
- **Can the model give us uncertainties, too?**

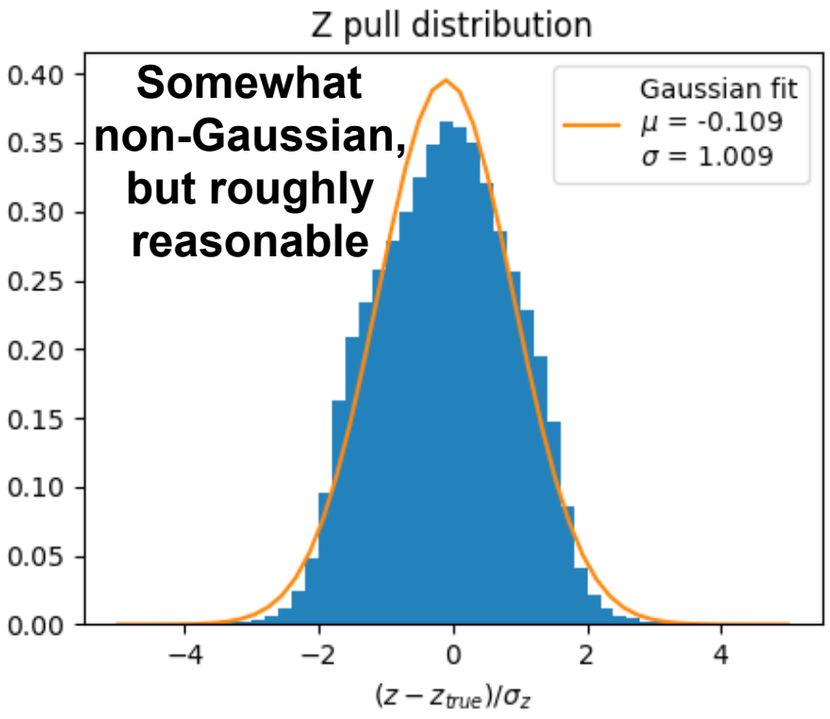


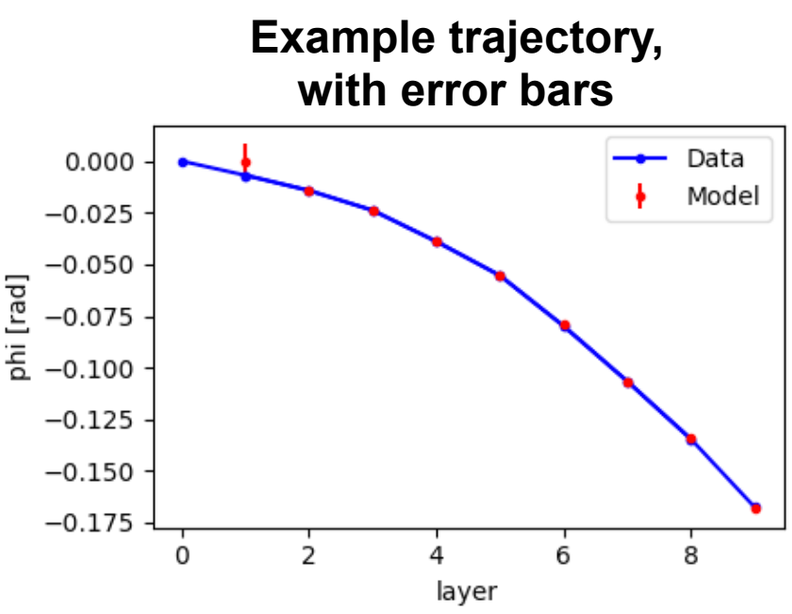
- **Change to Gaussian log-likelihood maximization:**

$$L(x, y) = \log|\Sigma| + (y - f(x))^{\mathrm{T}} \Sigma^{-1} (y - f(x))$$

- **Now we can plot pull distributions:**

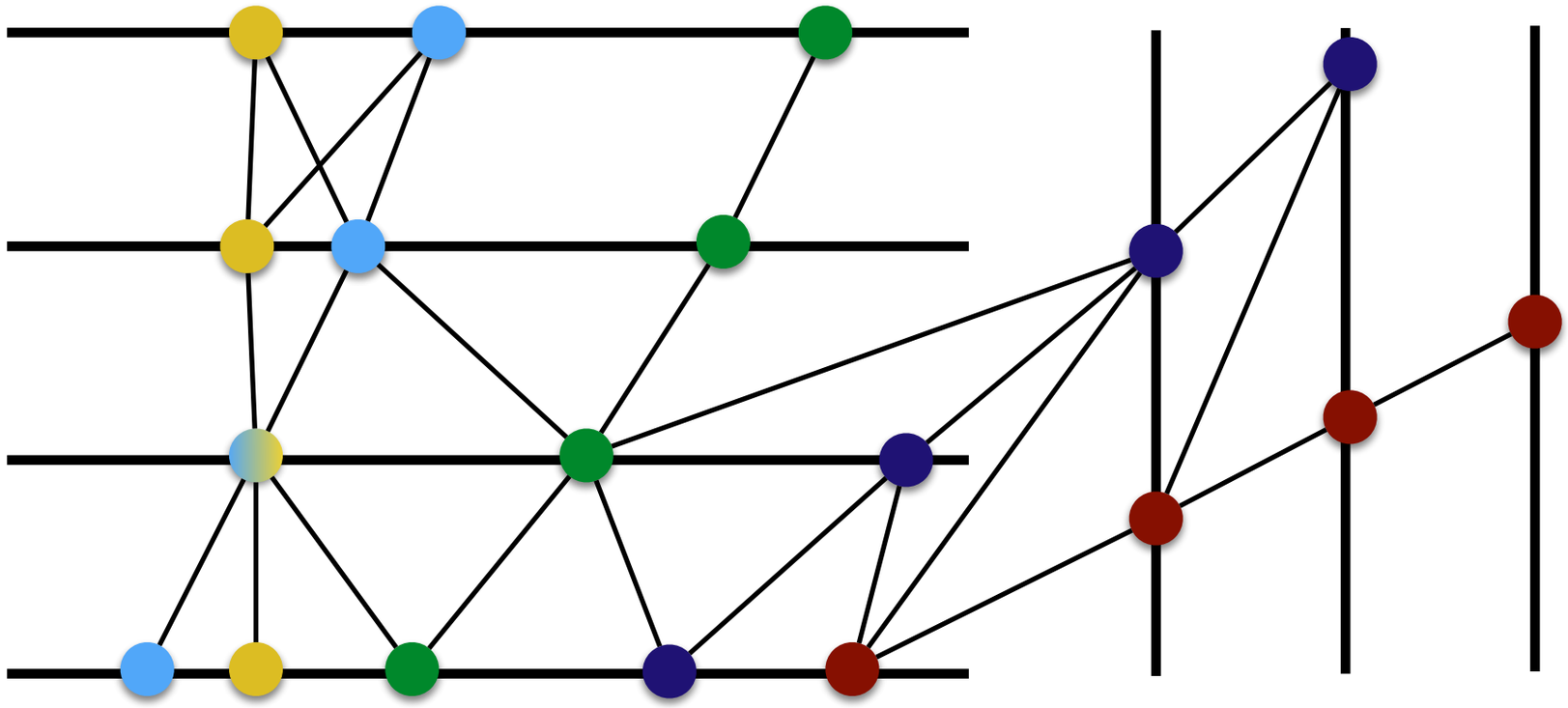






See related work from BM@N experiment: https://indico.cern.ch/event/587955/contributions/2937545/

Graph formulation



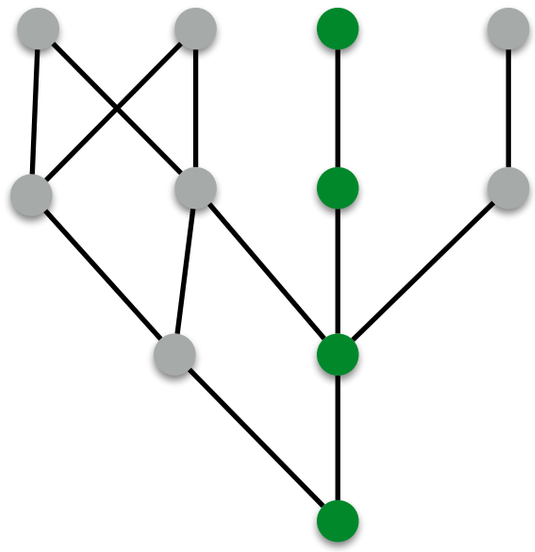
- **What if we structure our data as a *graph* of connected hits?**
  - Connect plausibly-related hits using geometric constraints or some pre-processing algorithm (e.g. Hough)
- **What kinds of models can we apply to this representation?**
  - Traditional architectures clearly don't work
  - but there's a growing sub-field of ML called *Geometric Deep Learning*

http://geometricdeeplearning.com/

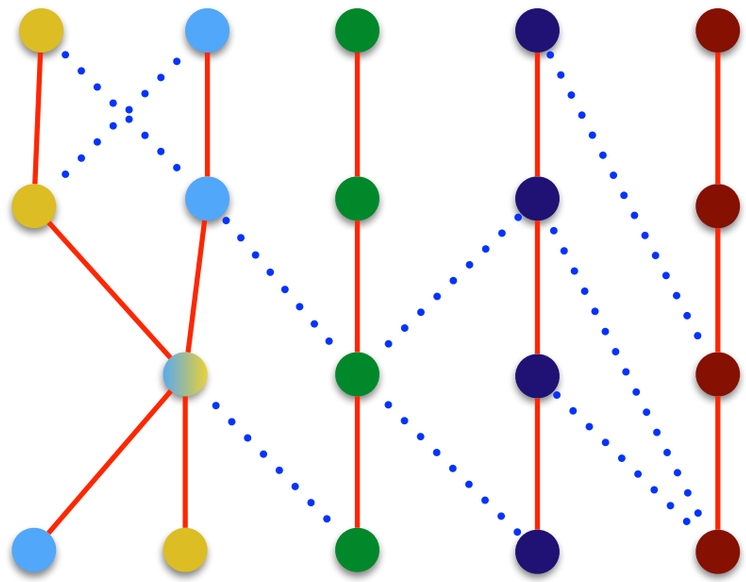
Graph neural networks - potential tracking apps

- **Hit classification**
  - Construct graph of possible hits from a seed
  - Binary-classify the hits
- **Segment classification**
  - Unseeded graph of connected hits
  - Binary classification of hit-pair segments (“Connecting the dots”)
  - Similar to cellular automaton, Hopfield networks
- **Our architecture**
  - A *Message-Passing Graph Neural Network*
  - Design inspired by literature but customized for our purposes

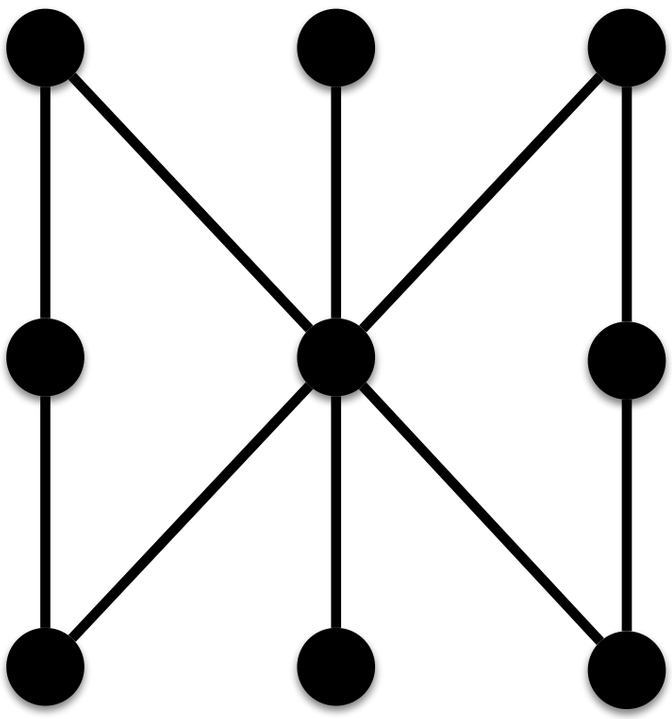
Hit classification



Segment classification



Graph network architecture



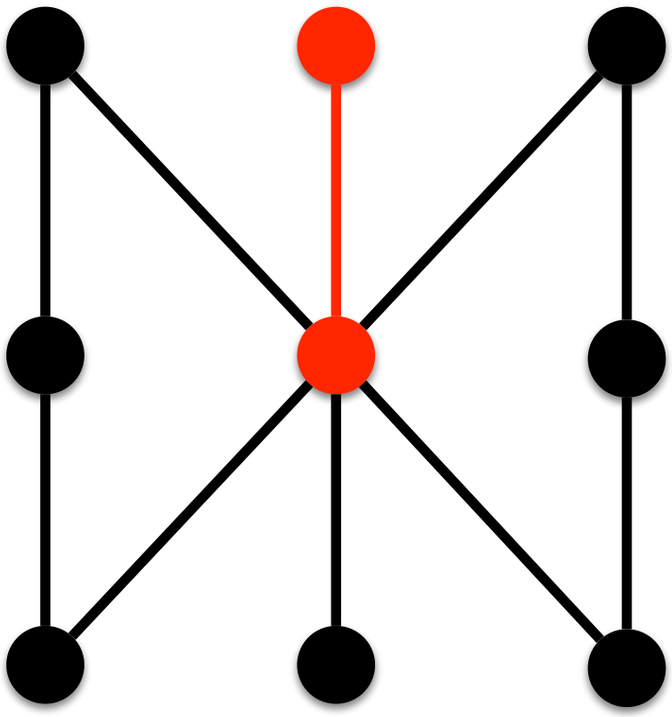
Two main components operate on the graph locally:

X ⟶ (N x D) node feature matrix

R_i ⟶ (N x E) association matrix of nodes to input edges

R_o ⟶ (N x E) association matrix of nodes to output edges

Graph network architecture

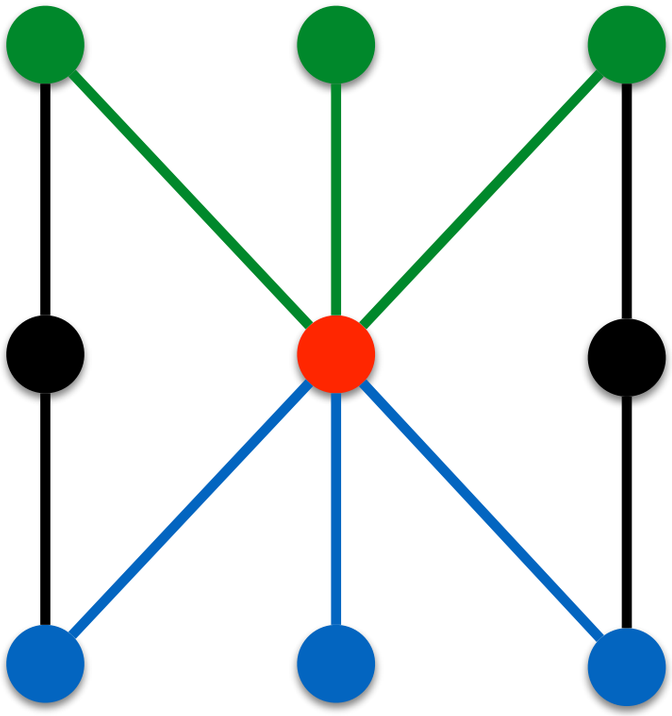


Two main components operate on the graph locally:

- ***Edge network*** uses the **node features** to compute **edge weights**

$$w = f_{\text{edge}}(R_i^T X, R_o^T X) \longrightarrow$$ (E) edge weight array

Graph network architecture

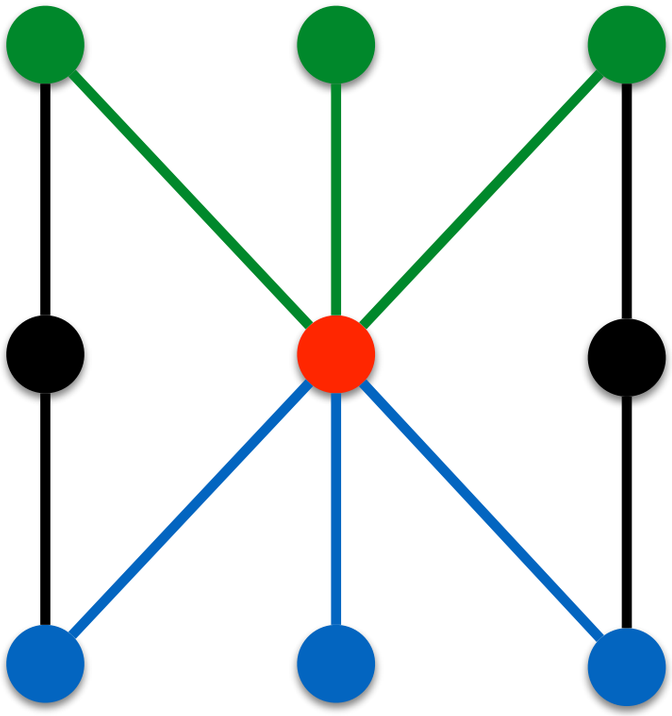


Two main components operate on the graph locally:

- ***Edge network*** uses the **node features** to compute **edge weights**
- ***Node network*** aggregates **forward** and **backward** node features with the edge weights and updates **node features**

$$X' = f_{\text{node}}\Big((R_i \odot w)R_o^T X, (R_o \odot w)R_i^T X, X\Big) \longrightarrow \text{(N x D) node features}$$

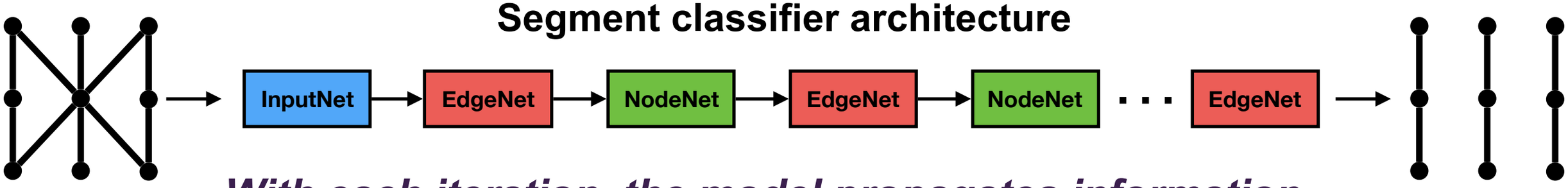
Graph network architecture



Two main components operate on the graph locally:

- ***Edge network*** uses the **node features** to compute **edge weights**
- ***Node network*** aggregates **forward** and **backward** node features with the edge weights and updates **node features**

Segment classifier architecture



With each iteration, the model propagates information through the graph, strengthens important connections, and weakens useless ones.

Architecture details

- Inputs:

$X \longrightarrow$ (N x D) node feature matrix

$R_i \longrightarrow$ (N x E) association matrix of nodes to input edges

$R_o \longrightarrow$ (N x E) association matrix of nodes to output edges

- The edge network is a 2-layer MLP with tanh and sigmoid activations:

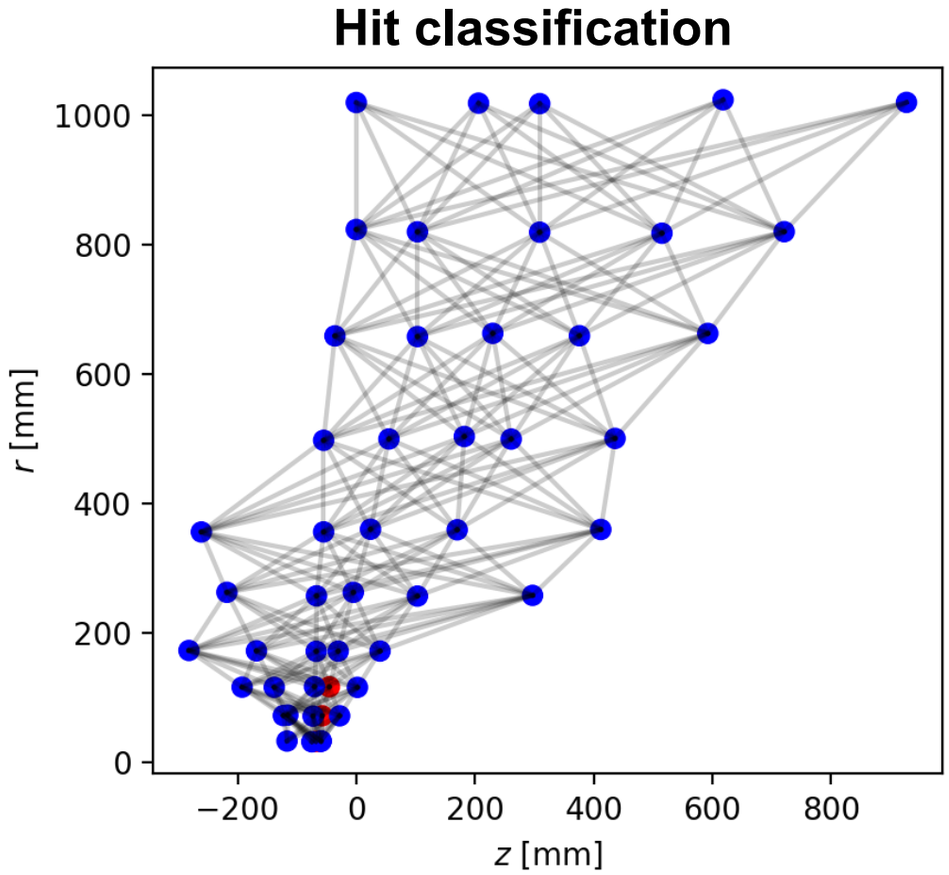
$$w = f_{\text{edge}}(R_i^T X, R_o^T X) \longrightarrow \text{(E) edge weight array}$$

- The node network is a 2-layer MLP with tanh activations:

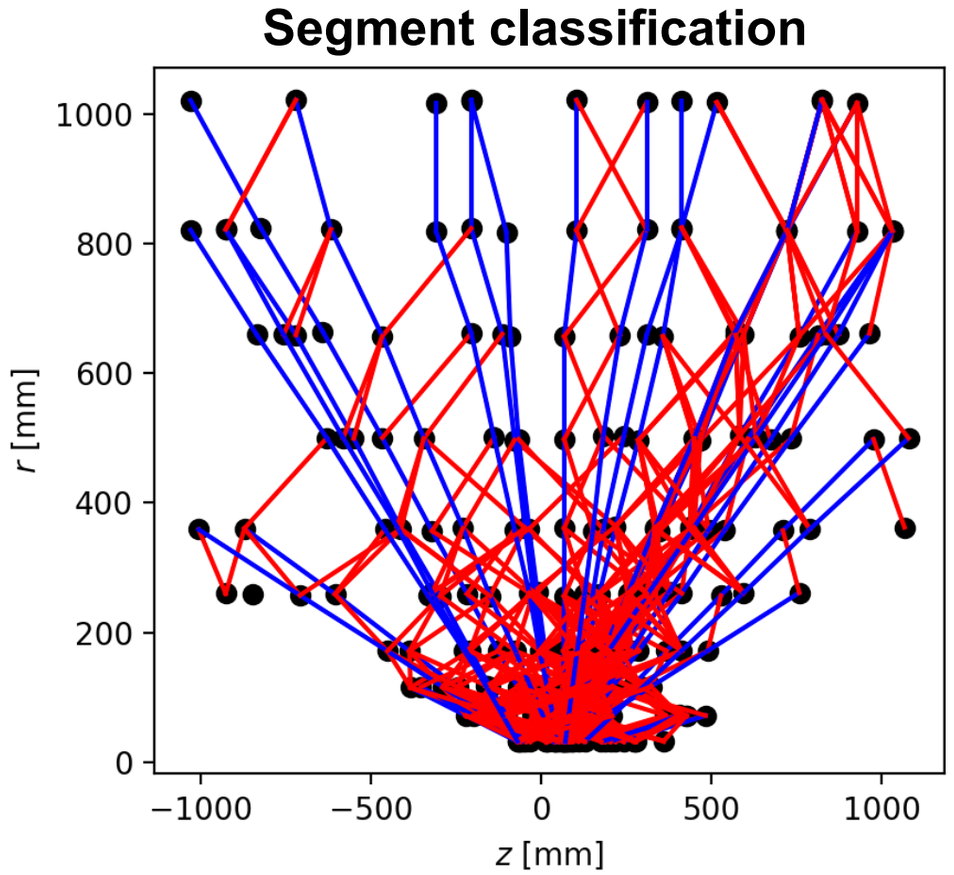
$$X' = f_{\text{node}}\Big((R_i \odot w)R_o^T X, (R_o \odot w)R_i^T X, X\Big) \longrightarrow \text{(N x D) node features}$$

- Outputs
  - ***Hit classifier***: binary classifier layer scores each node
  - ***Segment classifier***: final edge network application scores each edge

Constructing the graph



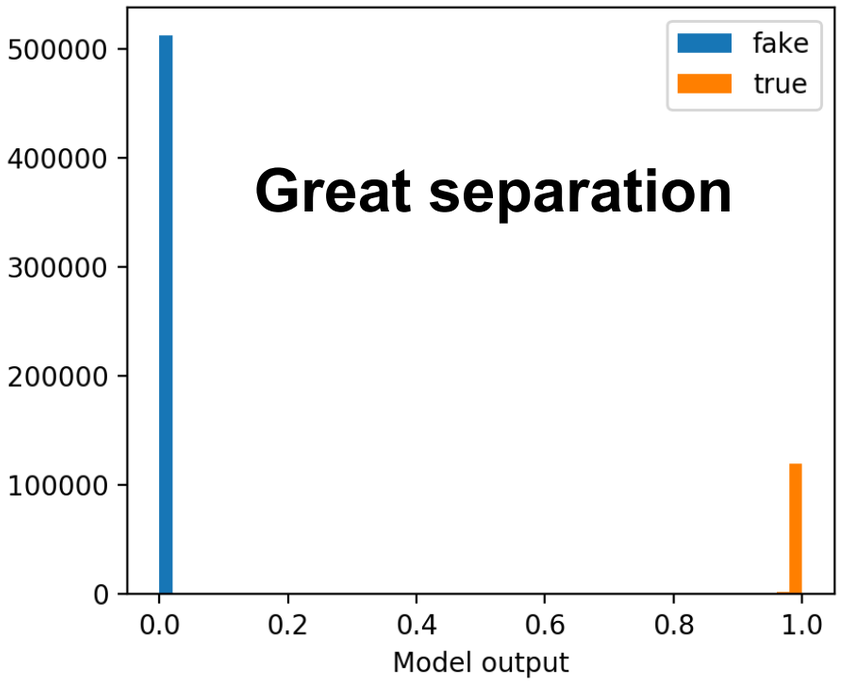
- Select hits in neighborhood of true track
- Label the seeds
- Target is the true binary labels for every hit

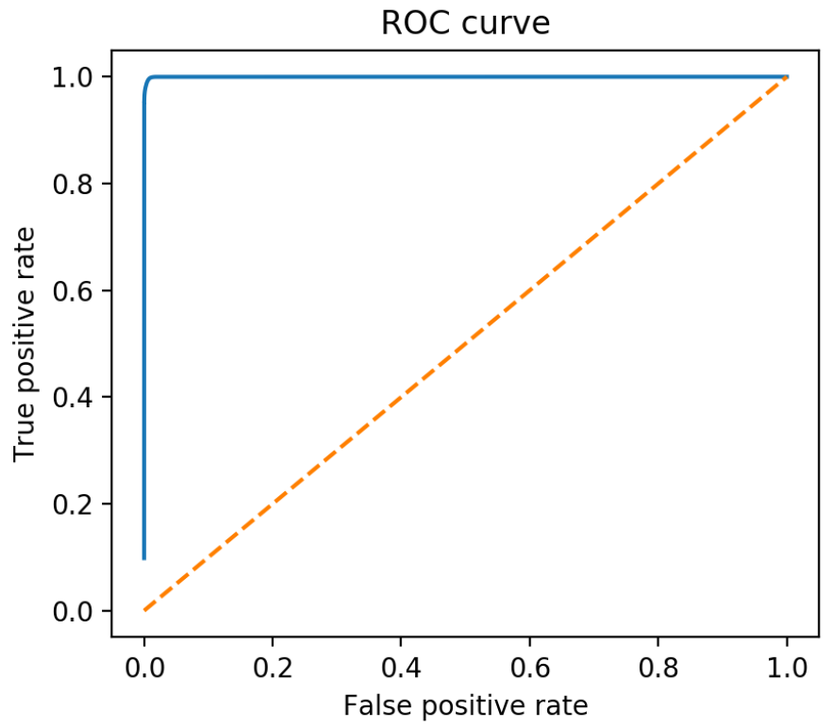


- Connect hits on adjacent layers using crude geometric constraints
  - delta(phi) < pi/4
  - delta(z) < 300mm

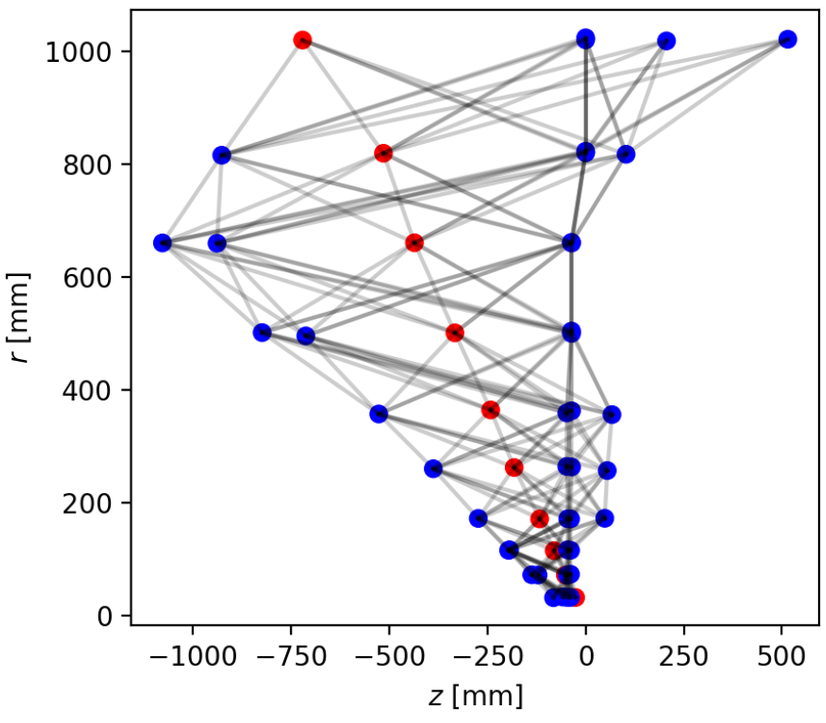
QCD data with pileup μ=10, pt>1GeV, barrel only, and duplicate hits removed

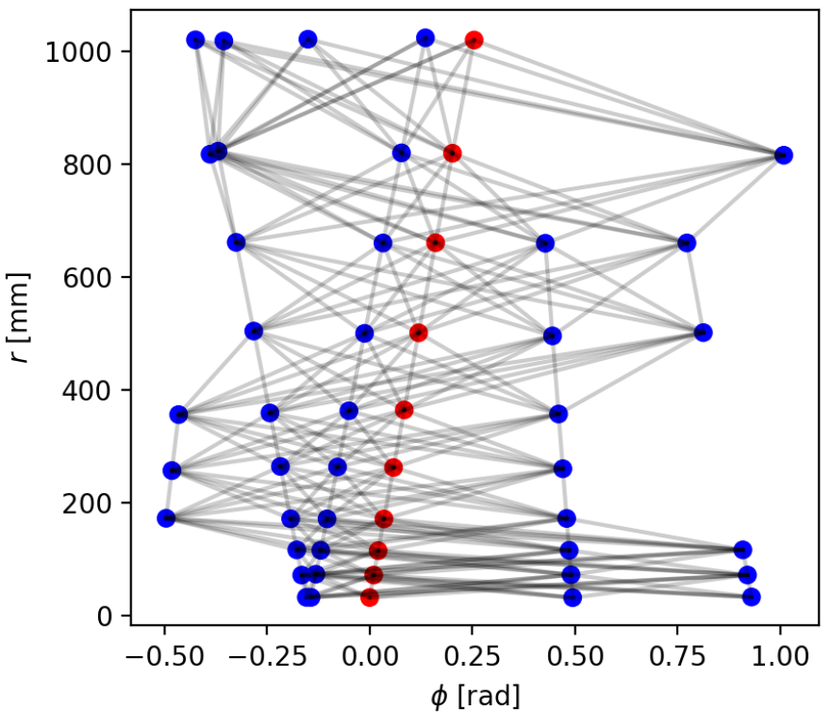
Hit classification model results





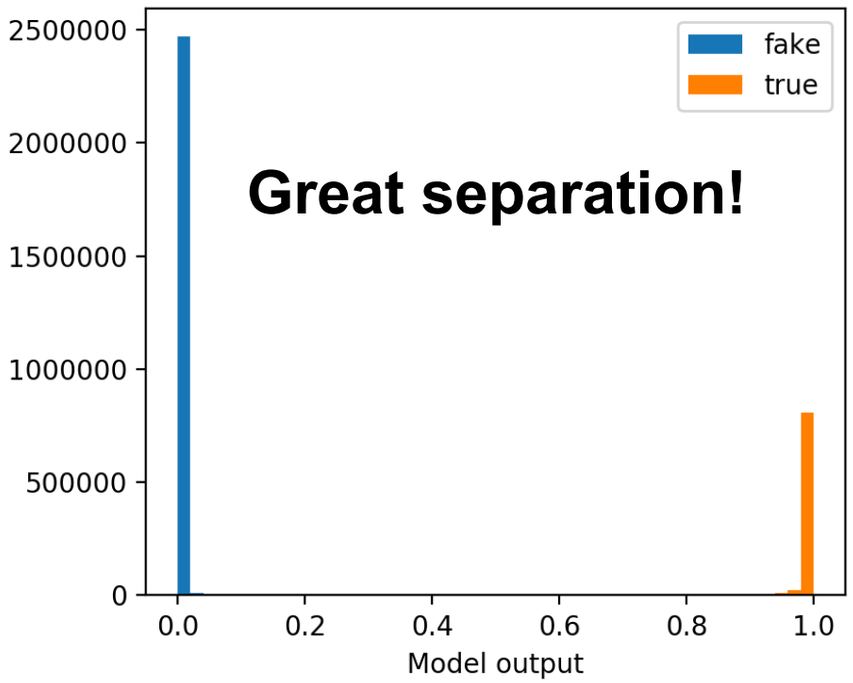
- Model settings
  - 7 graph iterations
  - 26k parameters
- It works really well!
- Good purity and efficiency

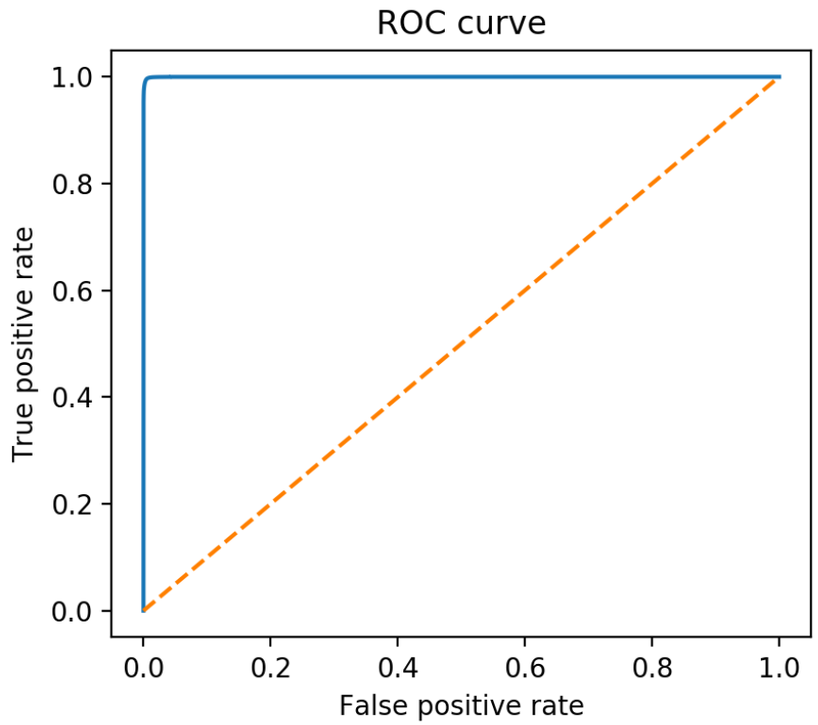





```
Test set metrics
Accuracy:   0.9942
Purity:     0.9918
Efficiency: 0.9793
```

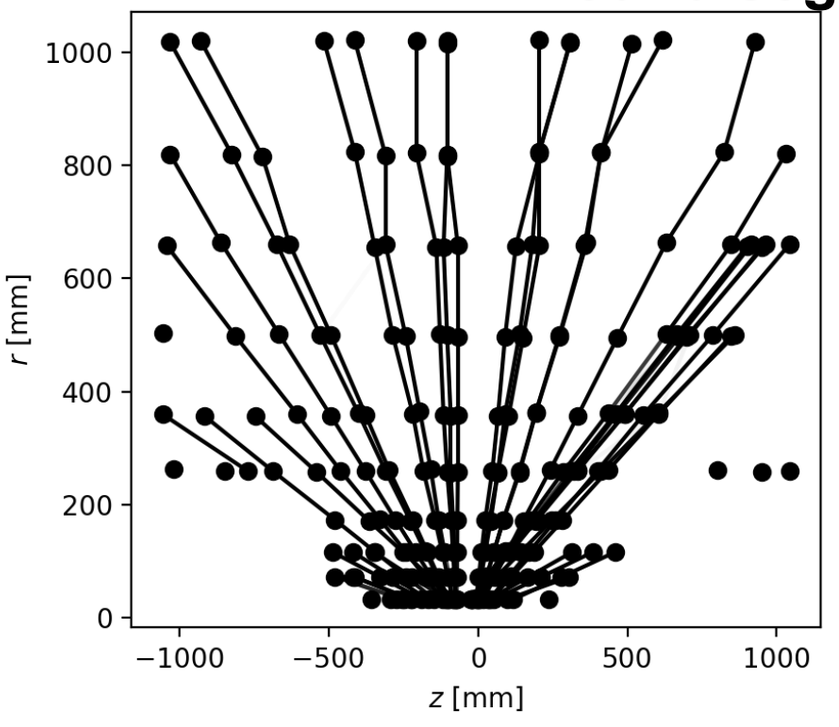
Segment classification model results

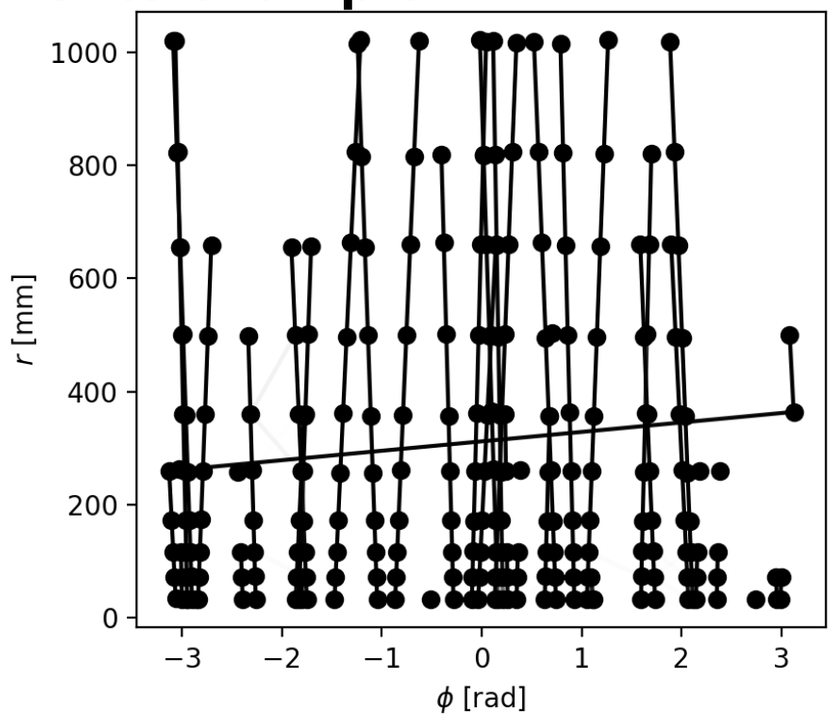




- Model settings
  - Only 4 graph iterations
  - *Only 6881 parameters!*
- It works really well!

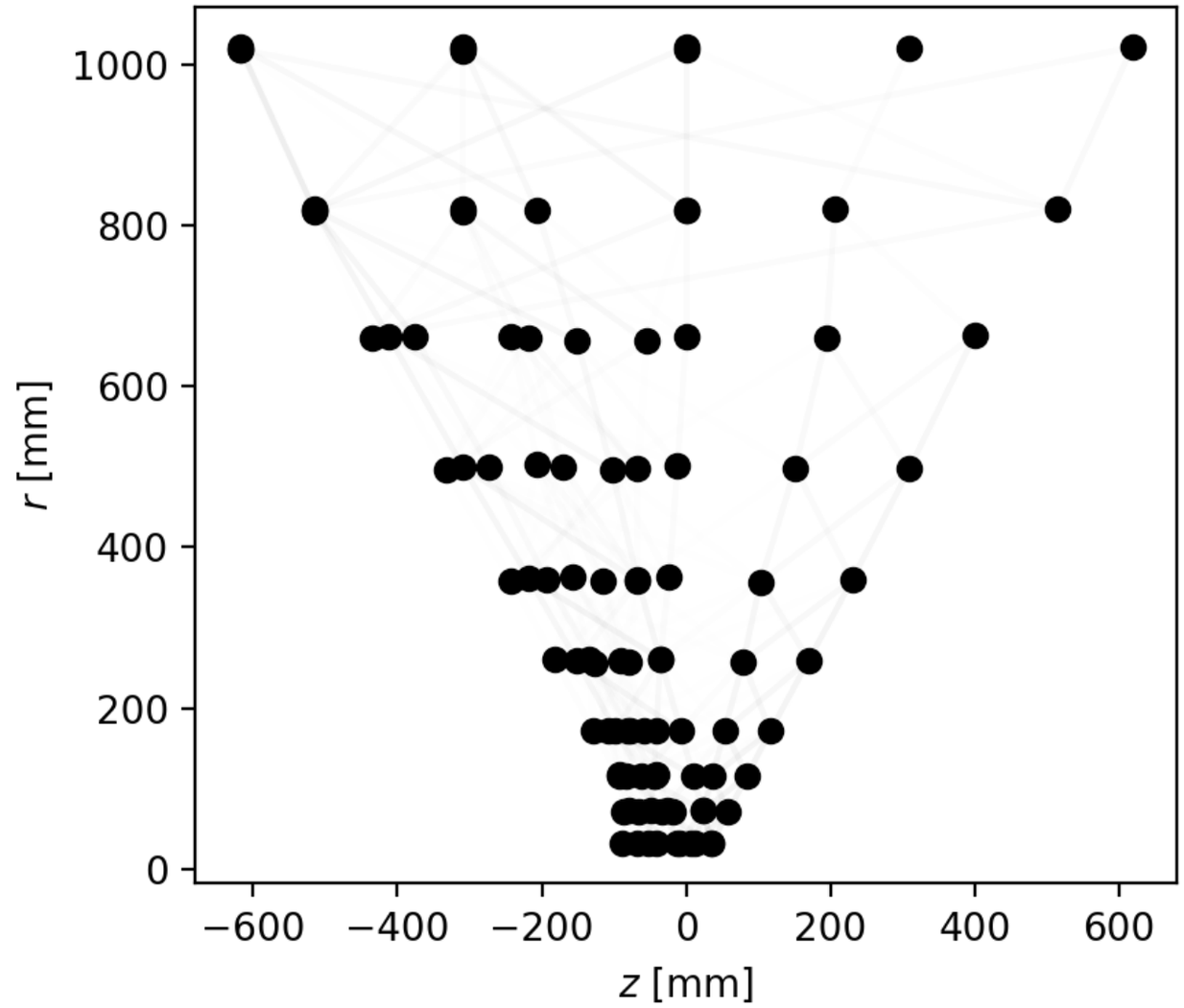
Found segments example

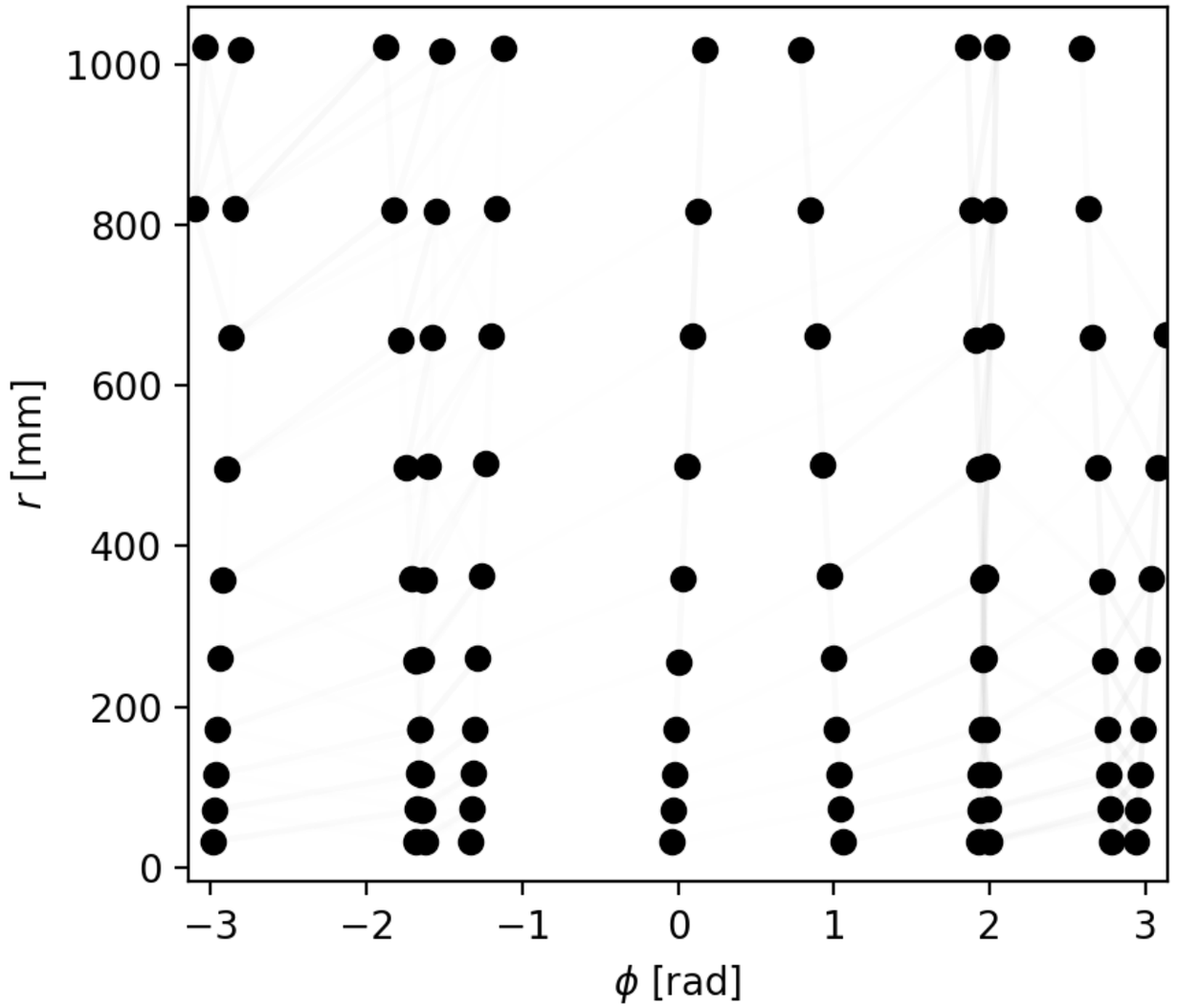





```
Test set metrics
Accuracy:   0.9952
Purity:     0.9945
Efficiency: 0.9870
```

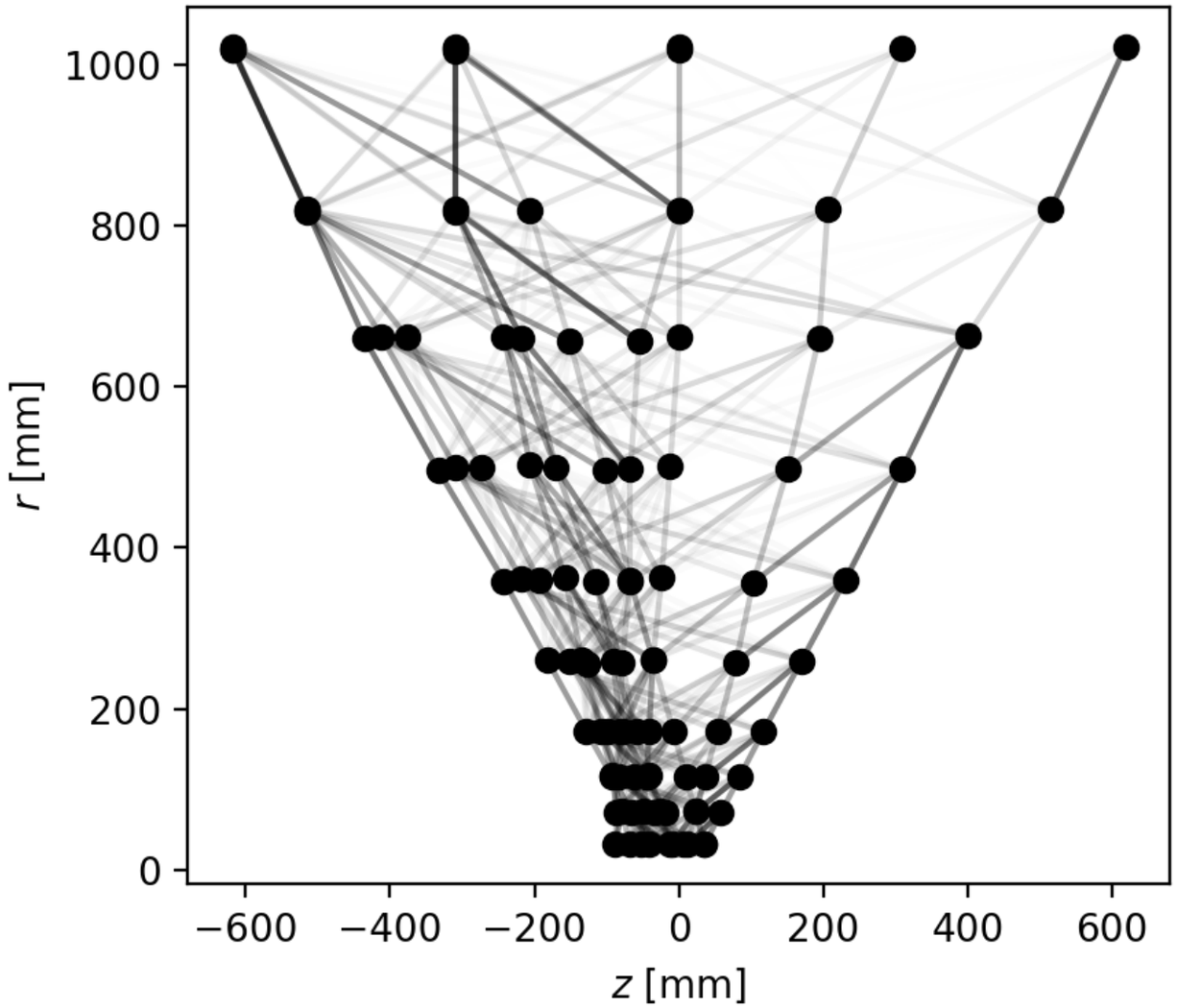
Evolution of graph edges 1st iteration

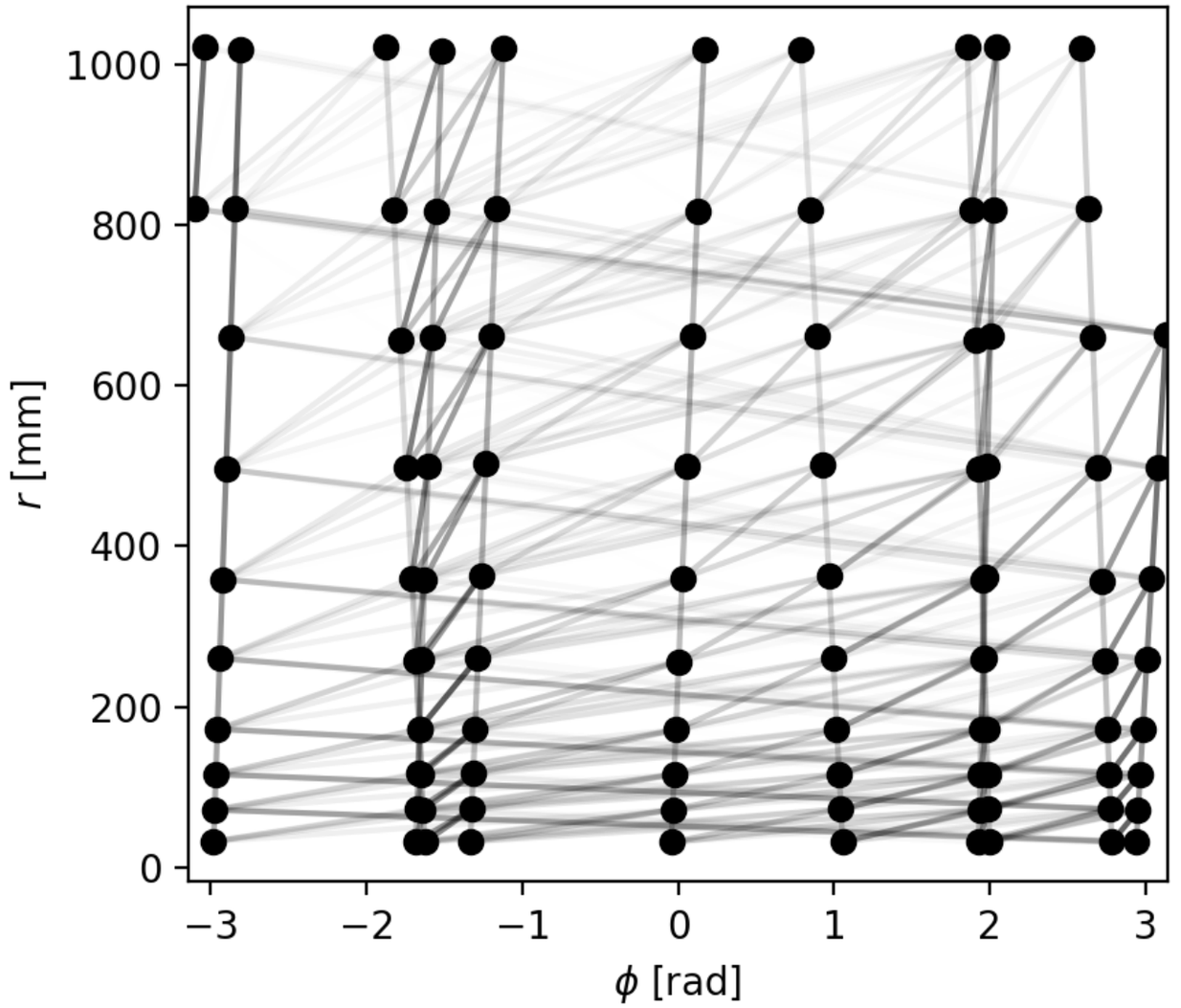




10-track sample for visualization

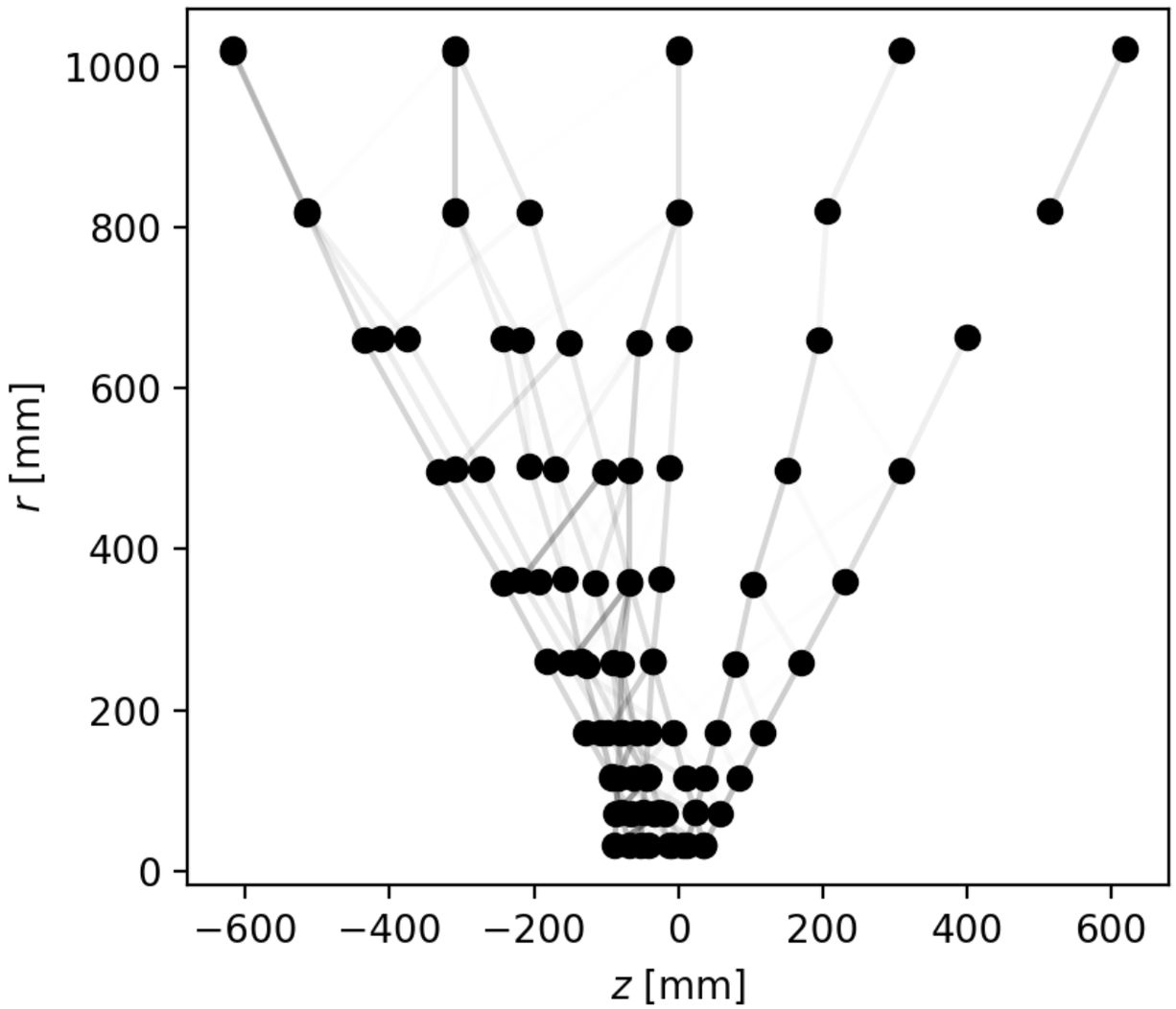
Evolution of graph edges 2nd iteration

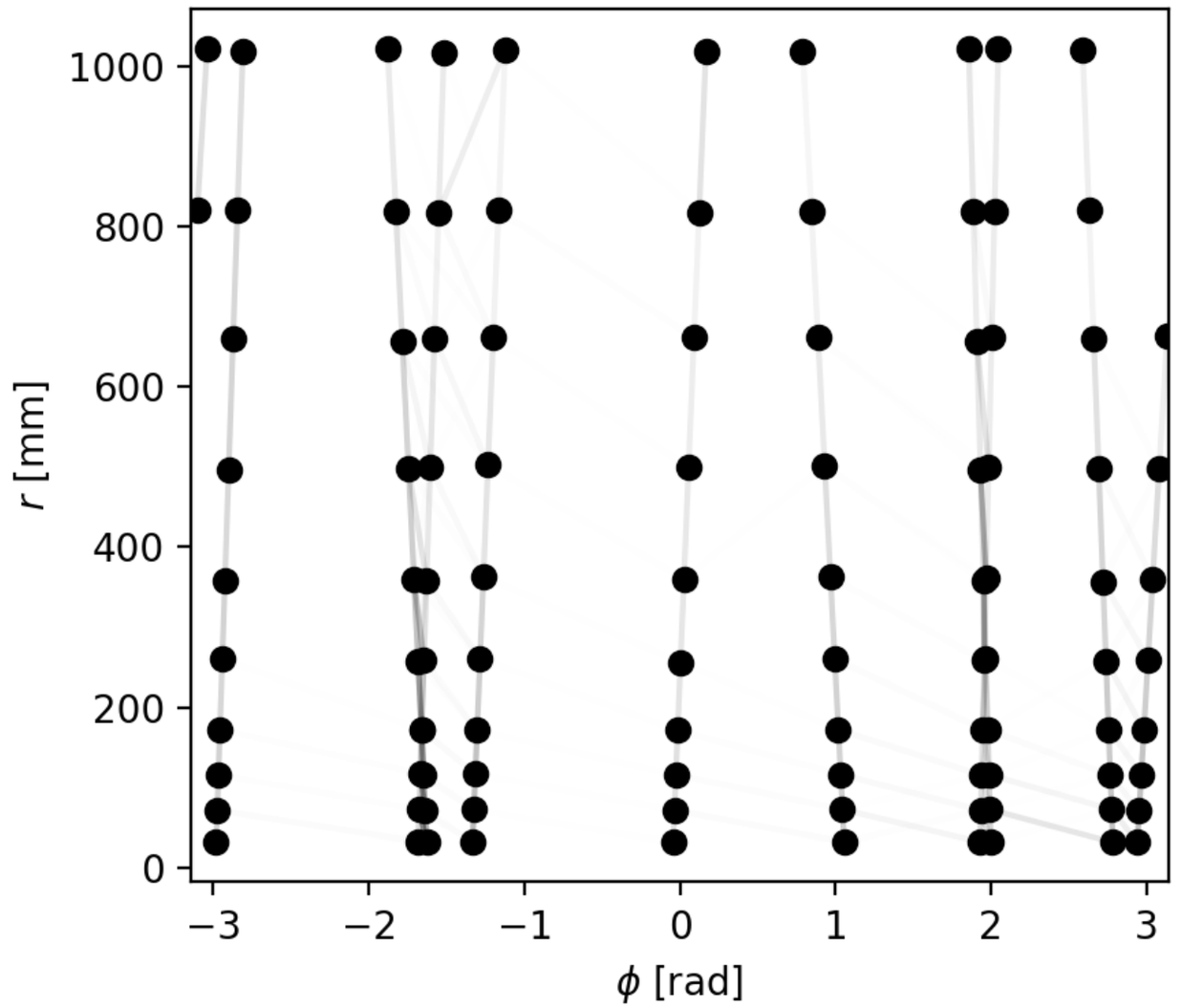




10-track sample for visualization

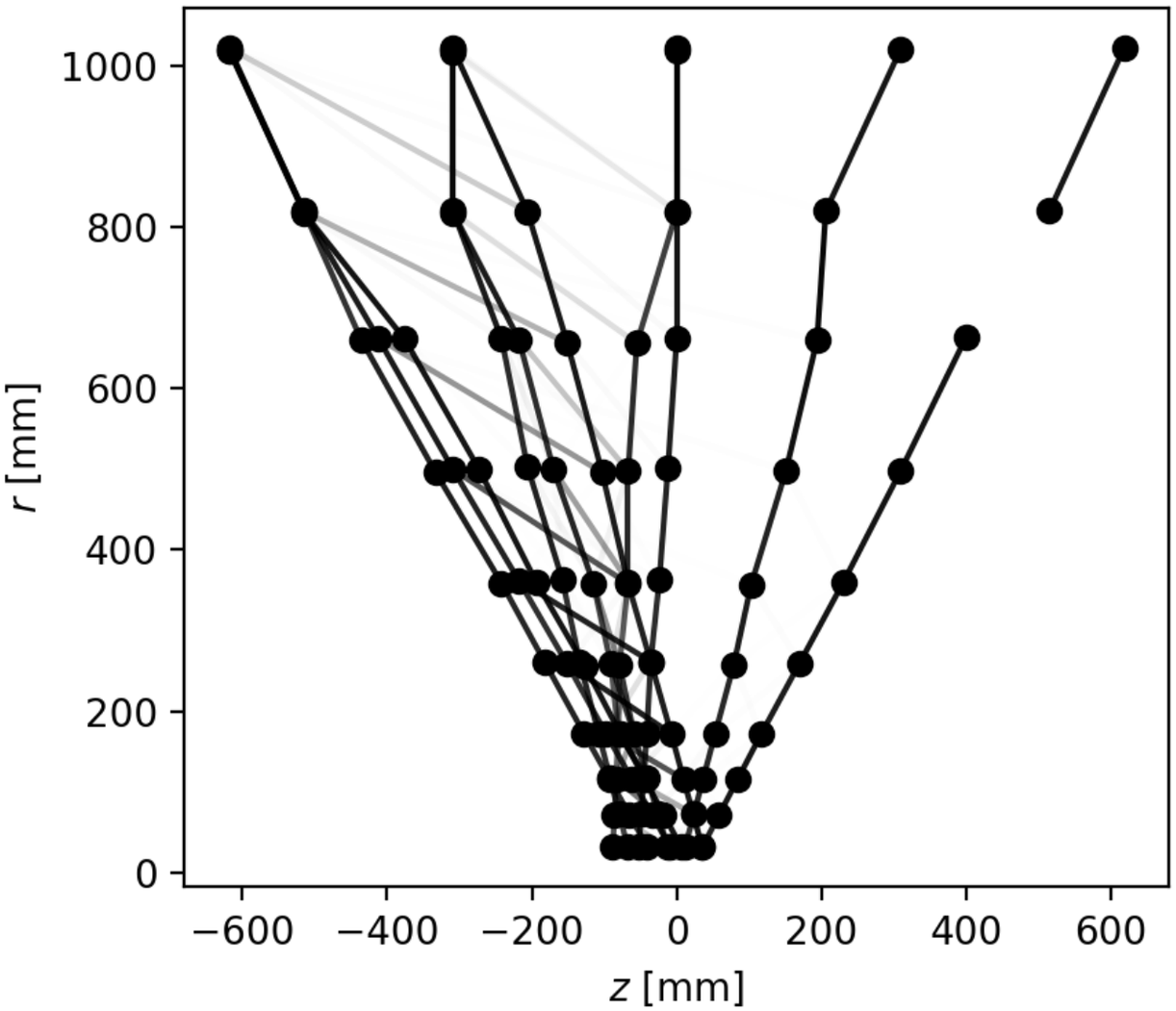
Evolution of graph edges 3rd iteration

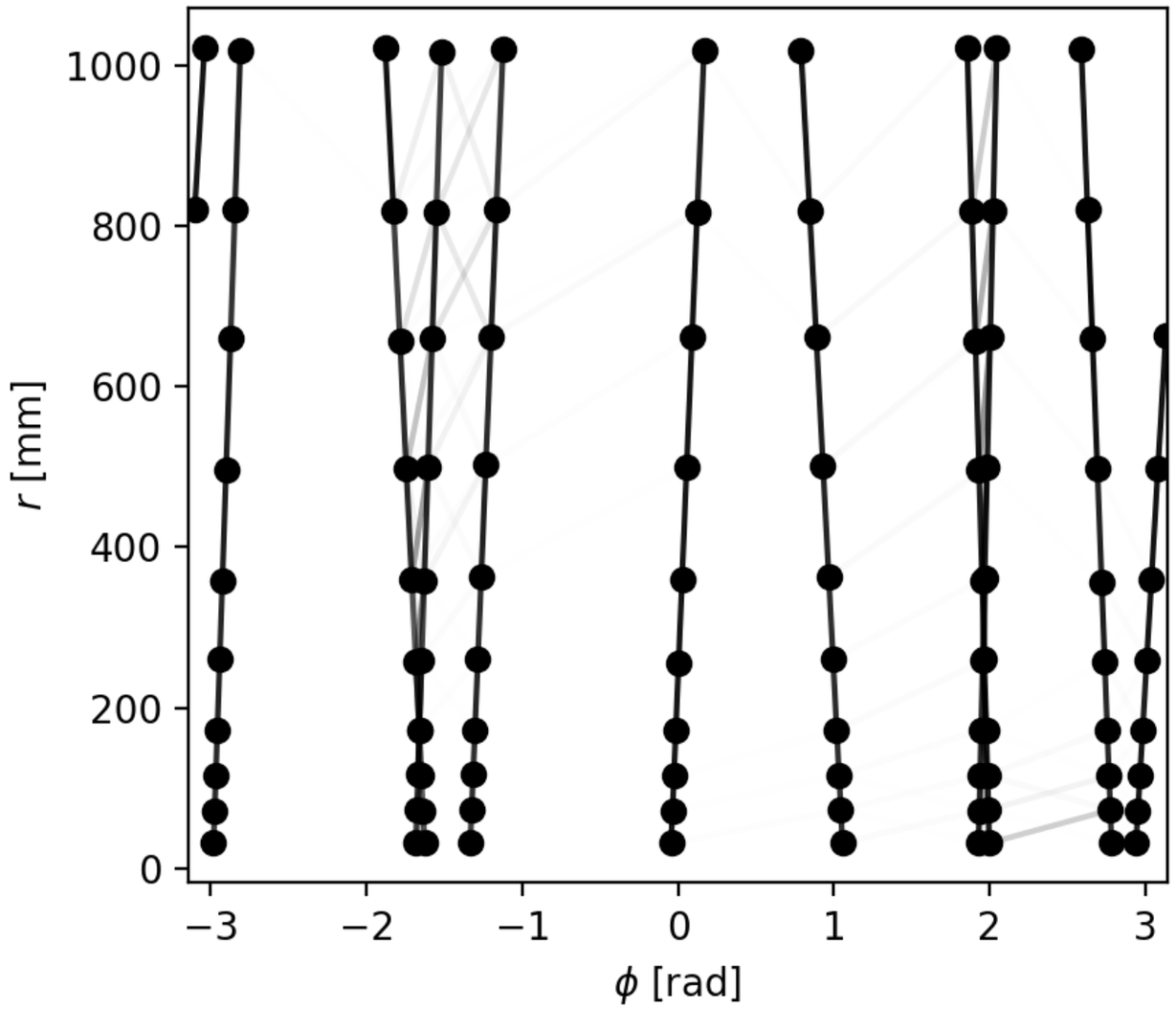




10-track sample for visualization

Evolution of graph edges 4th iteration

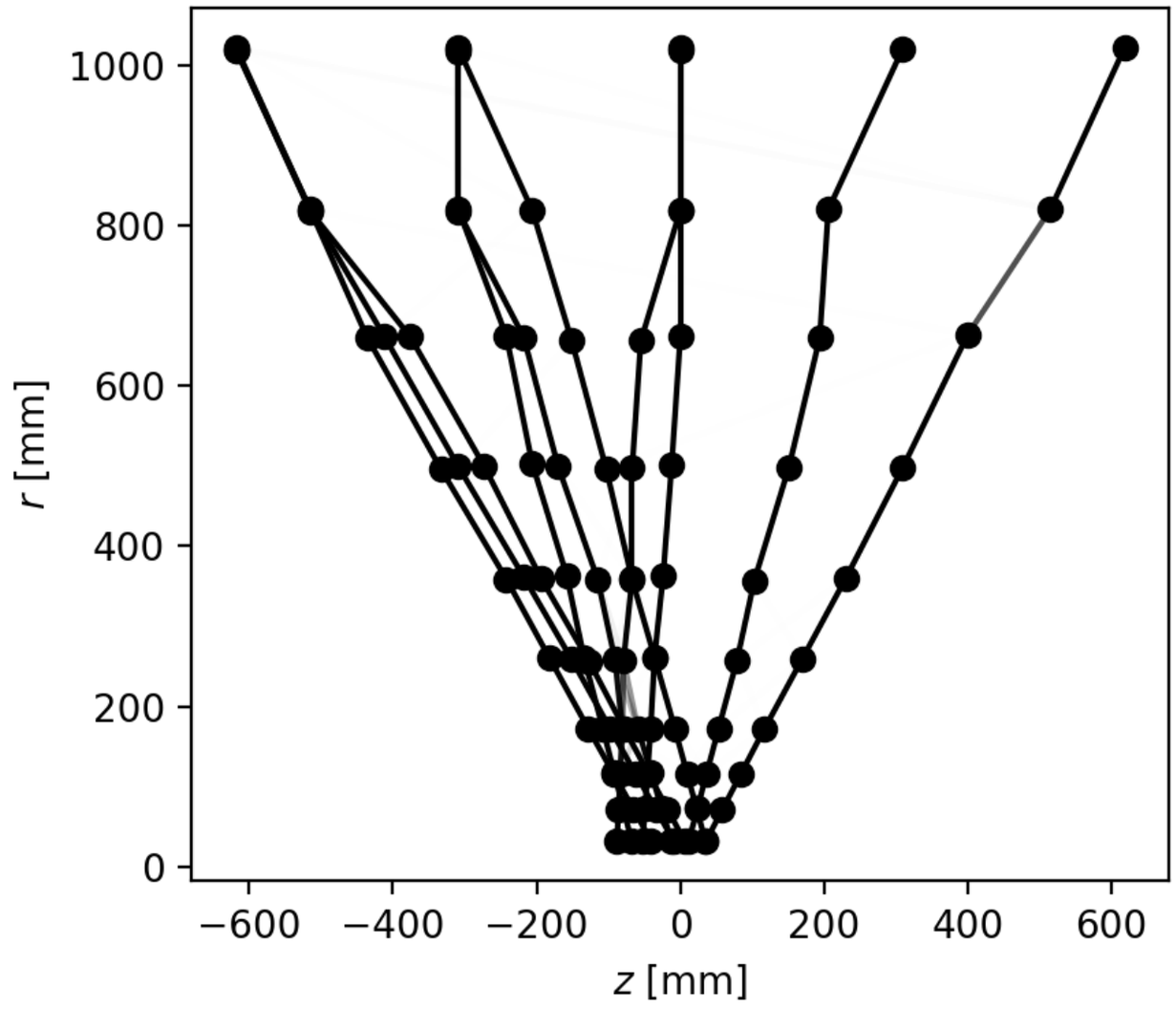


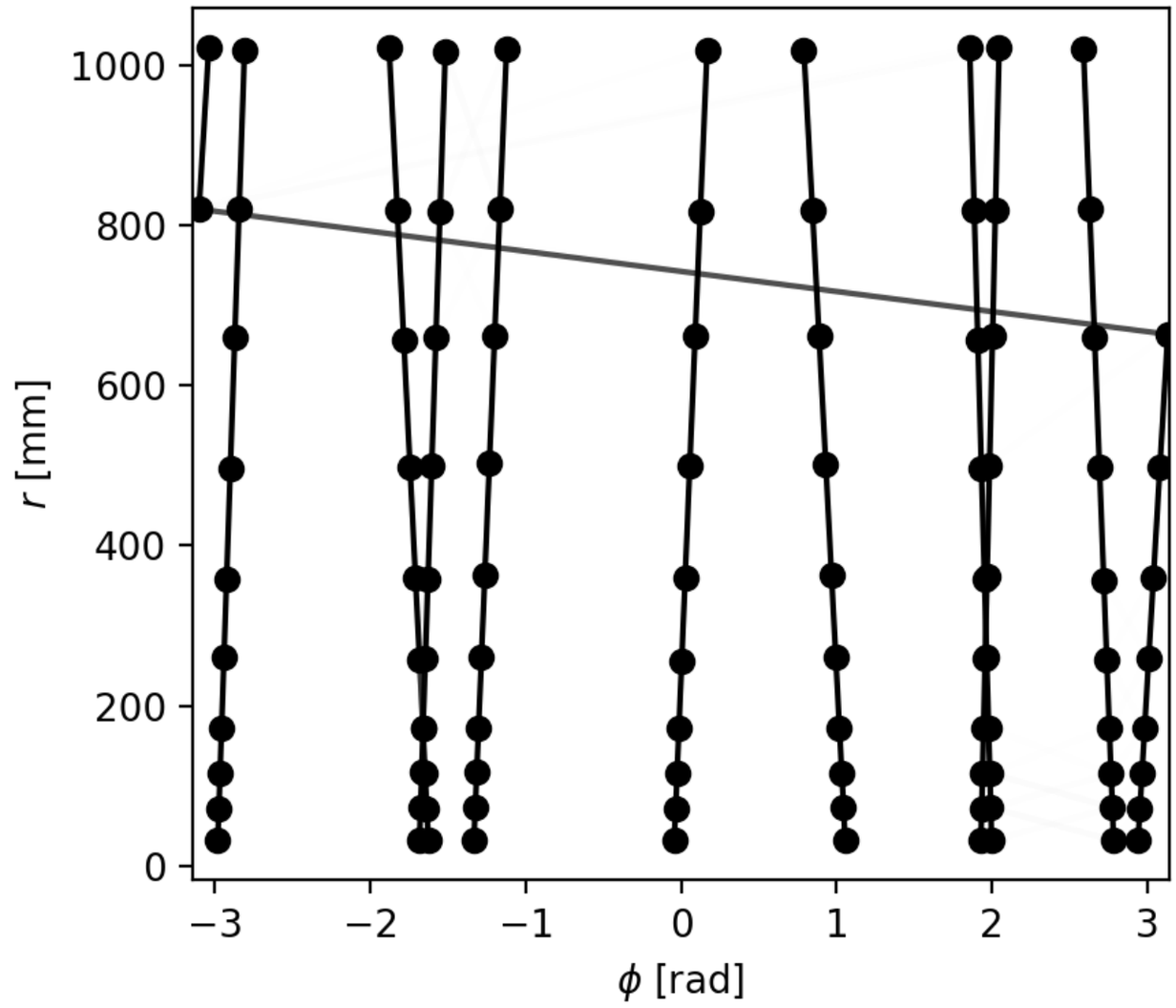


10-track sample for visualization

Evolution of graph edges

Final





10-track sample for visualization

Graph Neural Network next steps

- **Finishing/improving the pipeline**
  - Improved graph construction methods (segment criteria)
  - Going from segments to final tracks (e.g. connected clustering)
- **Scaling up data complexity**
  - Already moved to TrackML data (μ=200), working on performance
  - Tuning architecture and pipeline (see above)
  - Adding support for endcaps, double-hits, etc.
- **Investigate fast hardware implementations**
  - Collaborating with the HLS4ML team to port the GNN model to FPGAs

https://hls-fpga-machine-learning.github.io/hls4ml/

Conclusions

- **Particle tracking is an exciting and challenging area for ML/DL**
- **The TrackML challenges are successfully engaging the broader community**
  - And giving us a valuable standard benchmark dataset
- **HEP.TrkX has made some headway in this area**
  - We've studied a lot of ideas (many never shown)
  - We've gathered valuable expertise in applying ML to tracking
- **The Graph Neural Network approach seems particularly promising**
  - I think we've at least found the right *representation* of the data
- **Work is ongoing to demonstrate the full capability of the models**

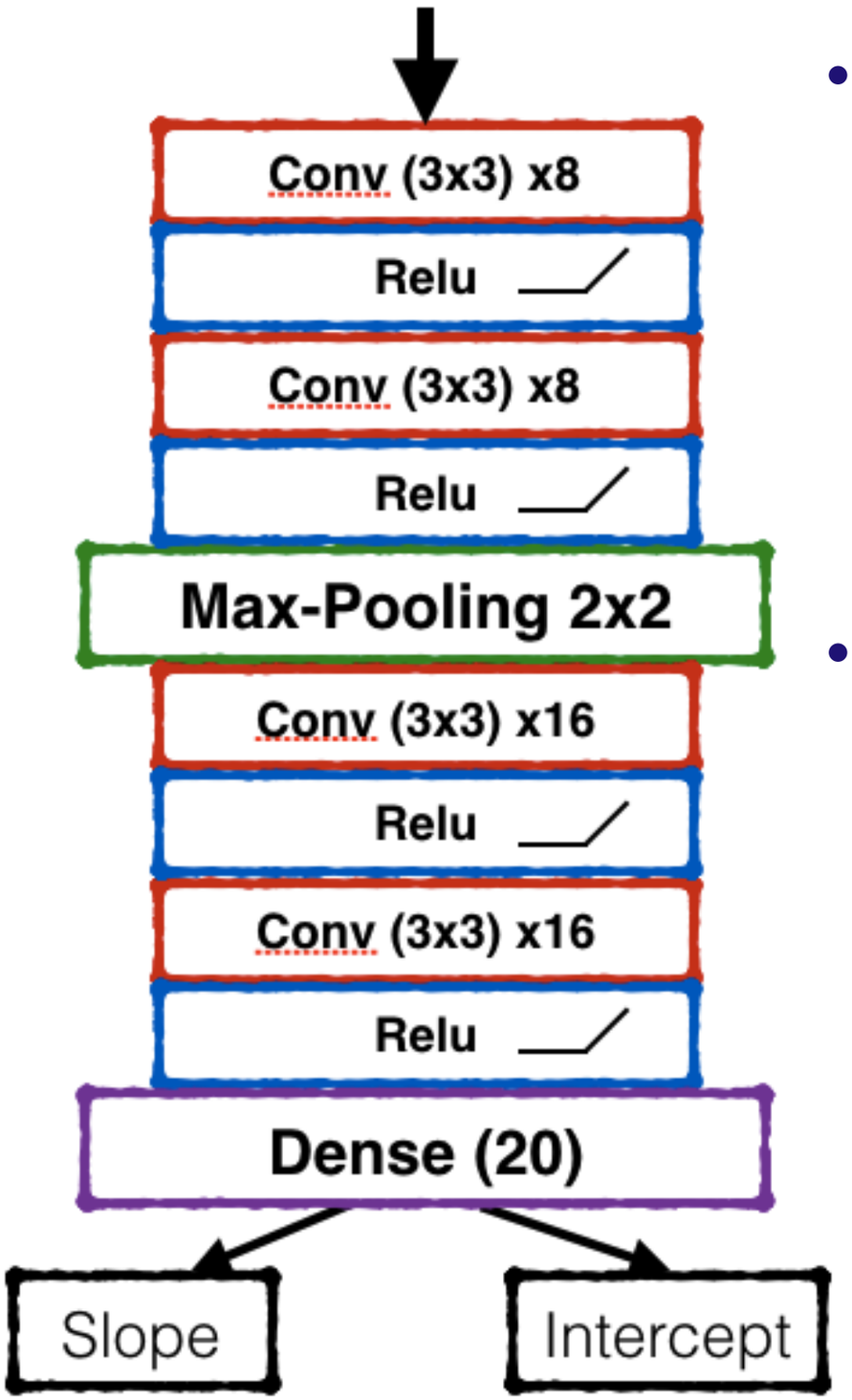
The end.

Thank you!

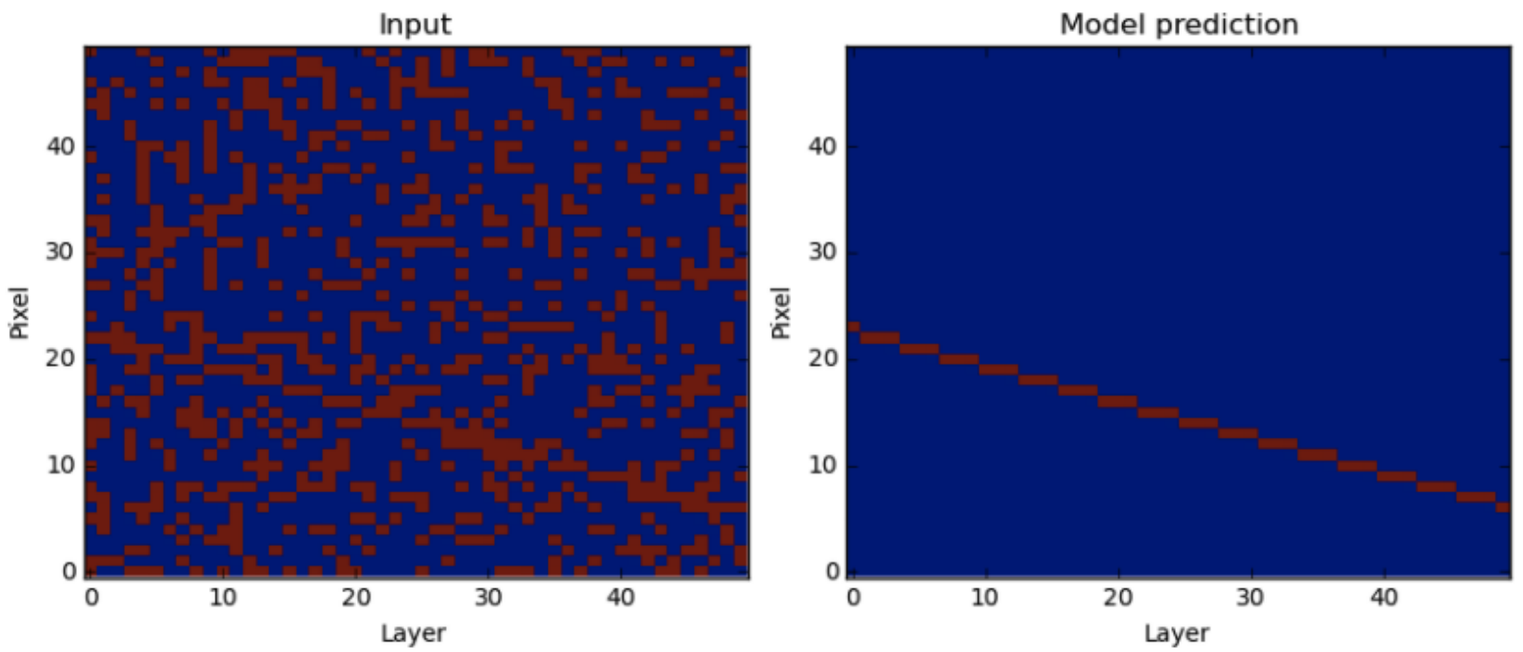
Other Particle Tracking applications from HEP.TrkX

[Work of Dustin Anderson]
CTD 2017

Track parameter estimation



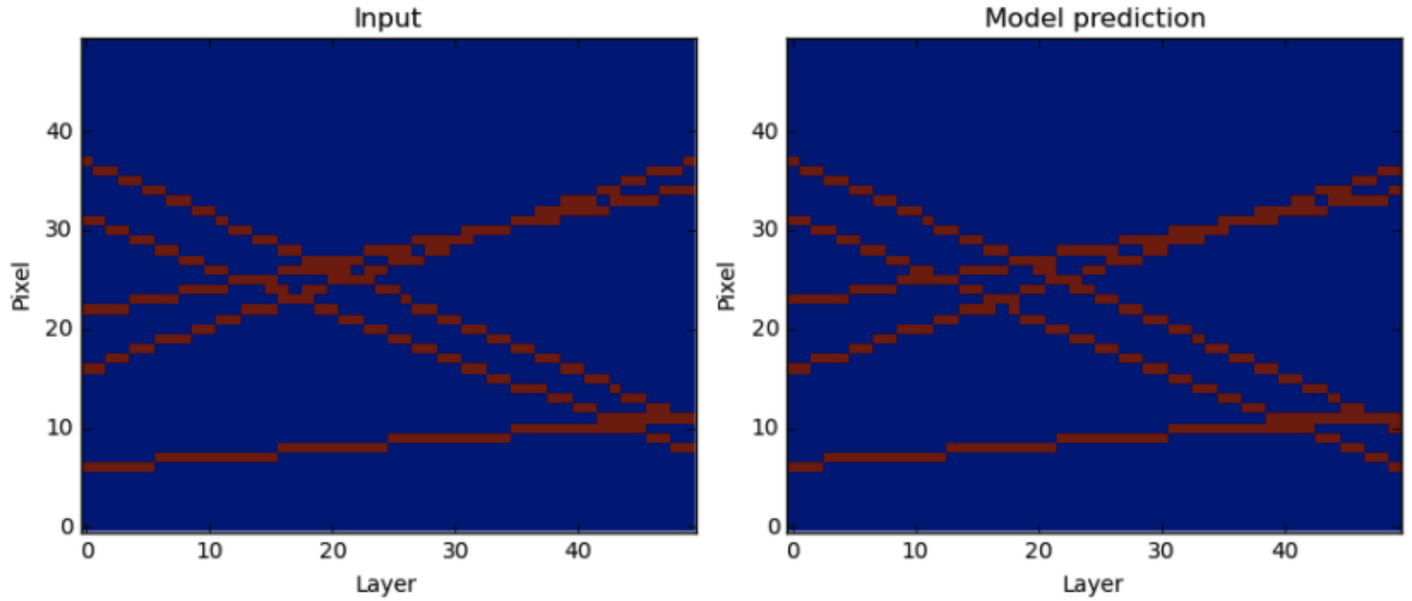
- Use a basic CNN with downsampling and regression head to estimate a track's parameters
  - could be an auxiliary target to guide training, or potentially useful as the final output of tracking!
- Identifying straight line params in heavy noise:

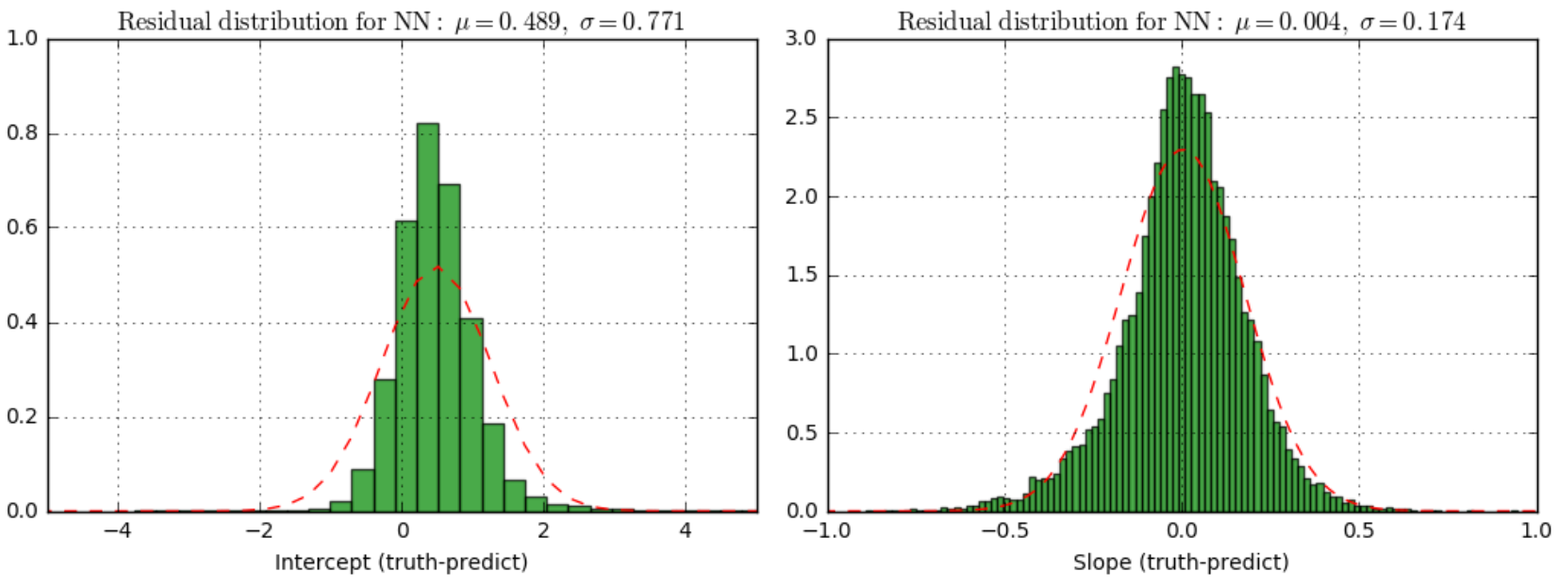


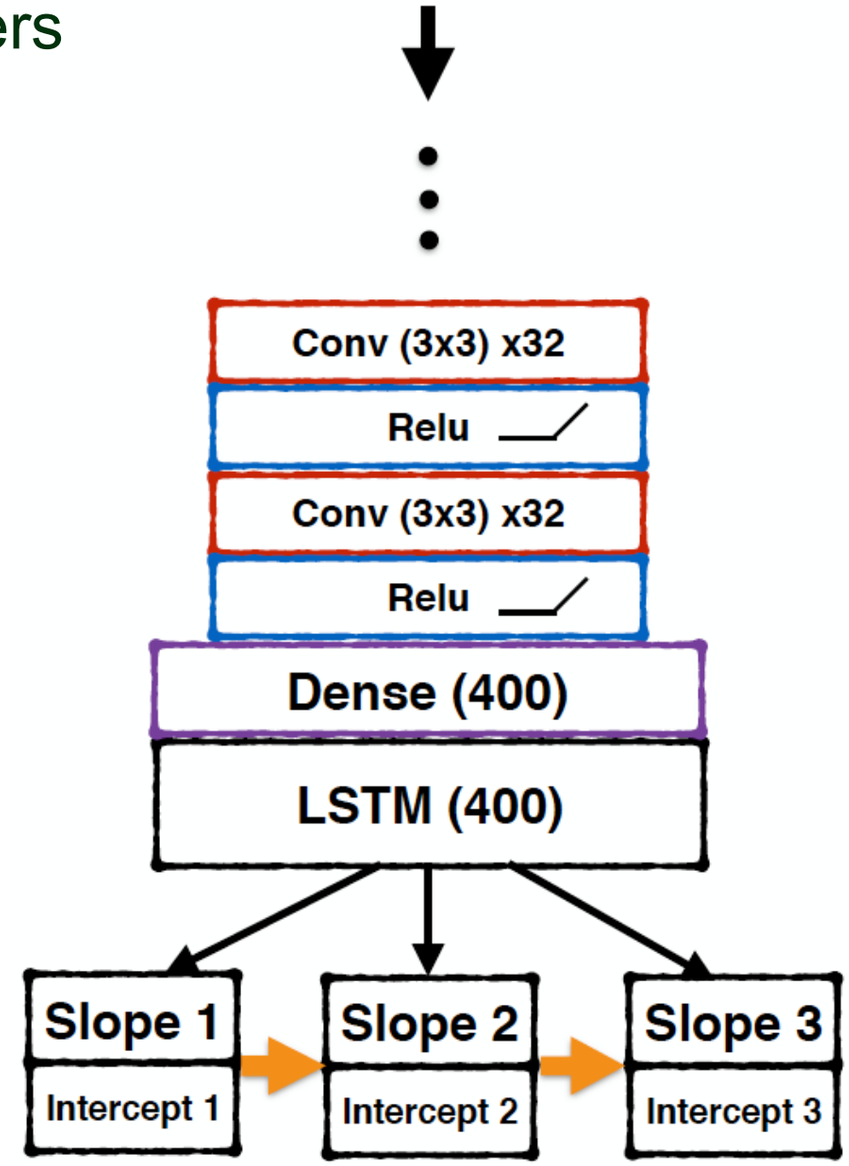
[Work of Dustin Anderson]
CTD 2017

Extending to variable number of tracks

- Attach an LSTM to a CNN to emit parameters for a variable number of tracks!
  - The LSTM generates the sequence of parameters
  - Requires an ordering the model can learn
  - Should provide some kind of stopping criteria



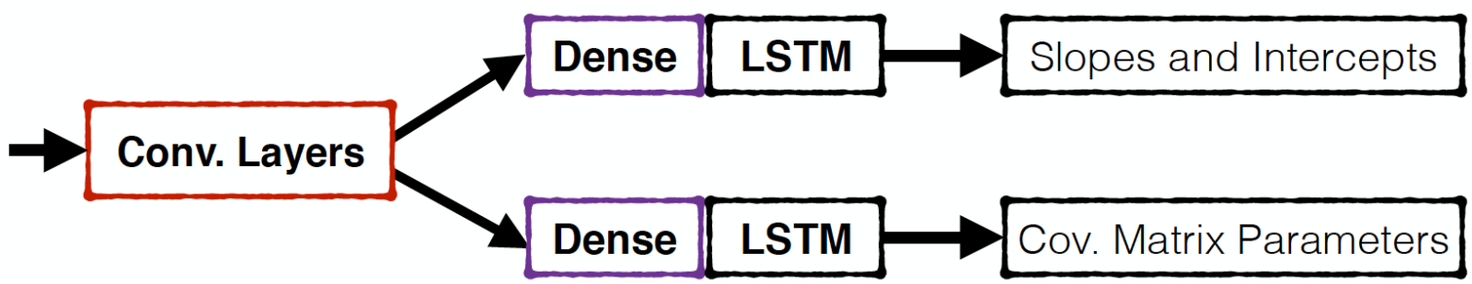




[Work of Dustin Anderson]
CTD 2017

Estimating uncertainties on parameters

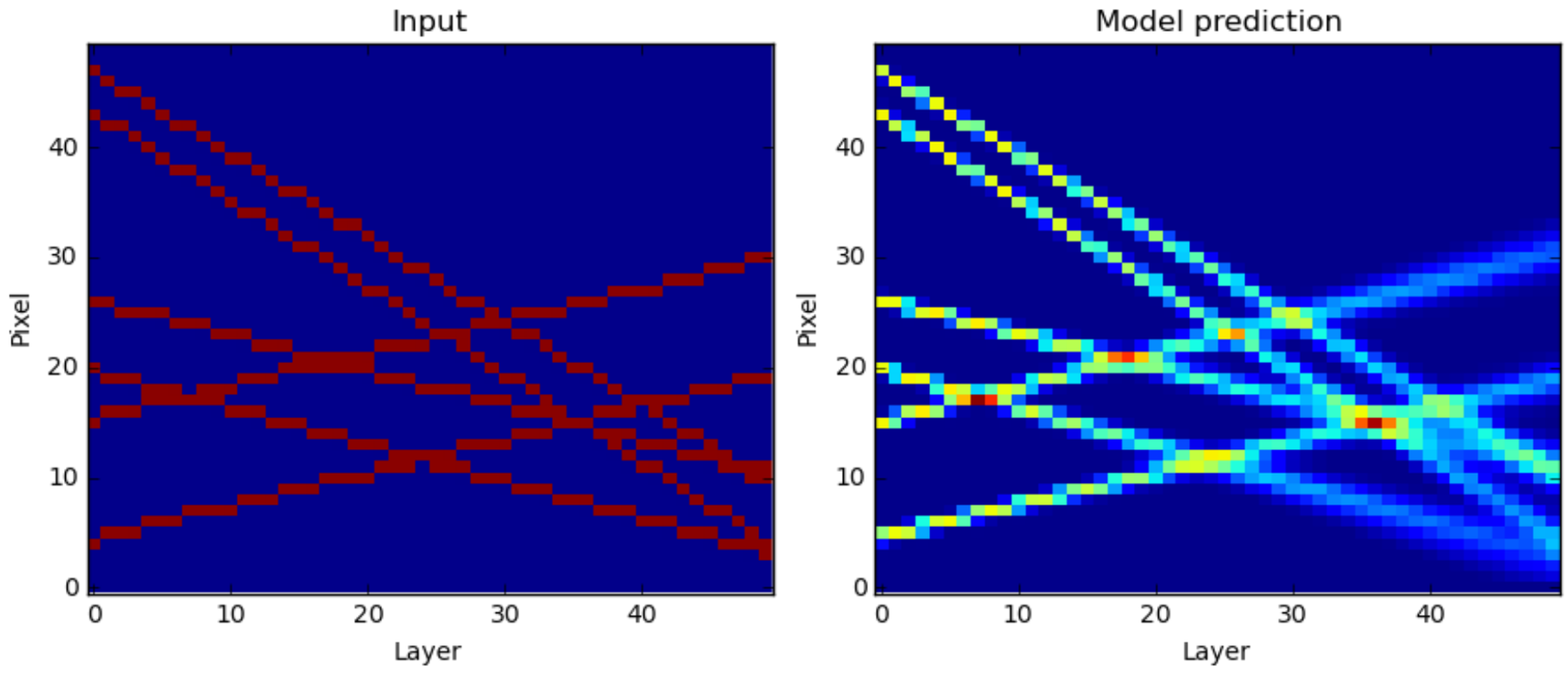
- Train the model to also estimate the uncertainties by adding additional targets:



- Train using a log gaussian likelihood loss:

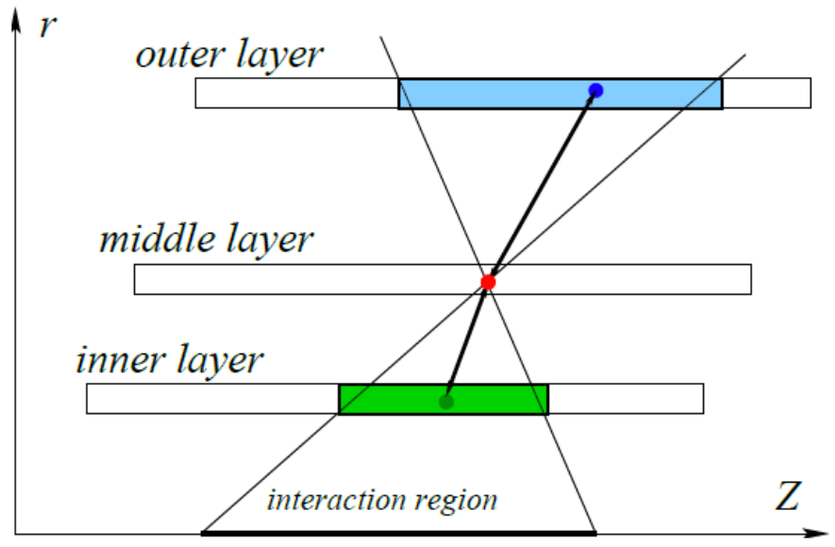
$$L(\boldsymbol{x}, \boldsymbol{y}) = \log|\boldsymbol{\Sigma}| + (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}))^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}))$$

- and *voila!*

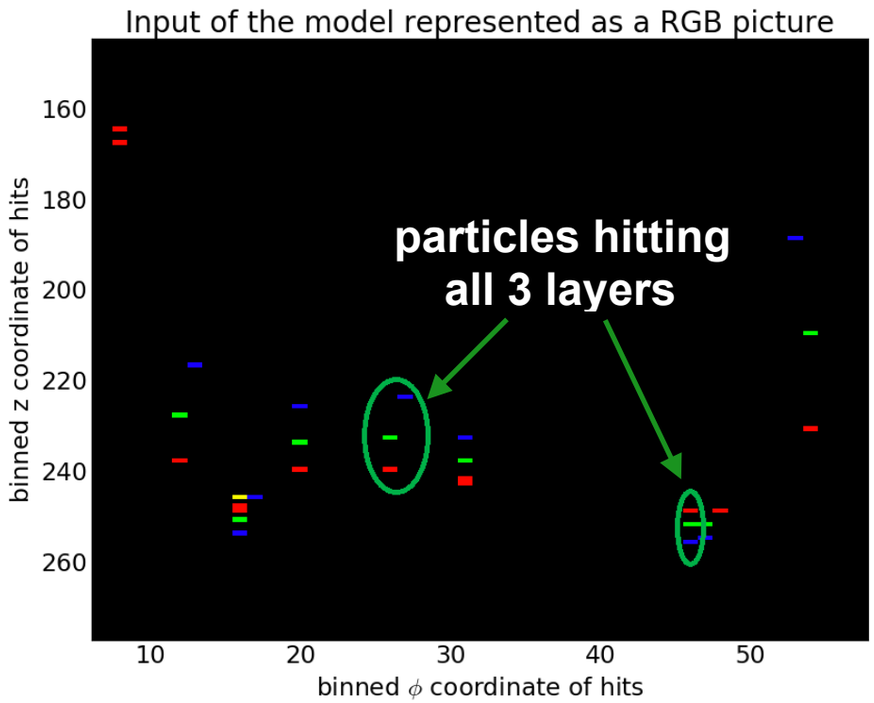


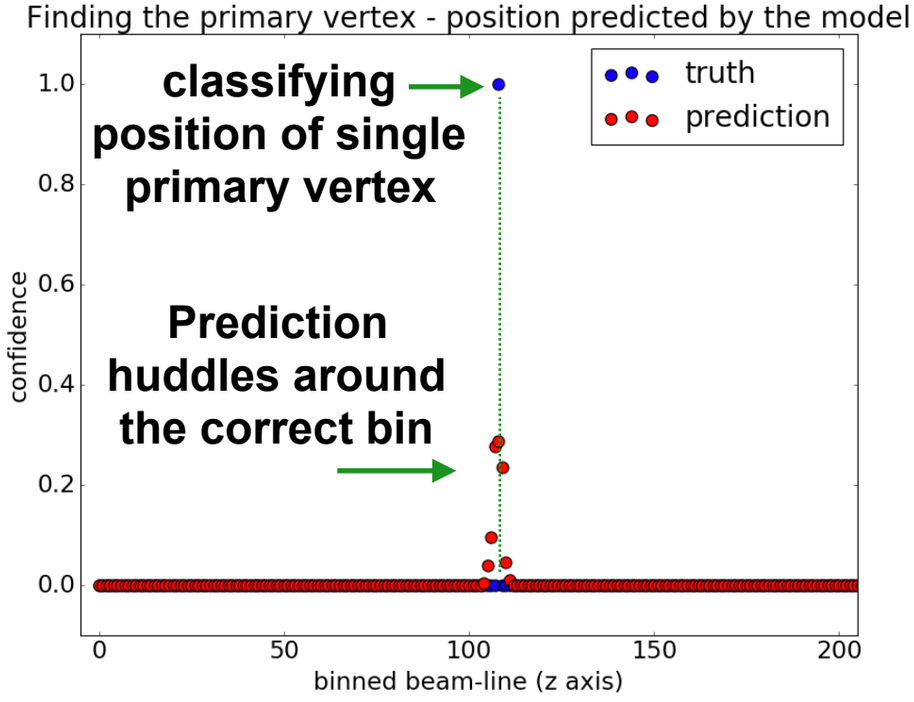
Vertex finding with CNNs

- Finding track seeds is slow because of the triplet combinatorics
- Try to estimate position(s) of the production vertex(es) to constrain the search
  - Input: 3 detector layers as 3-channel image
  - Output: binned z-position along interaction region



- Using semi-realistic tracking data from ACTS

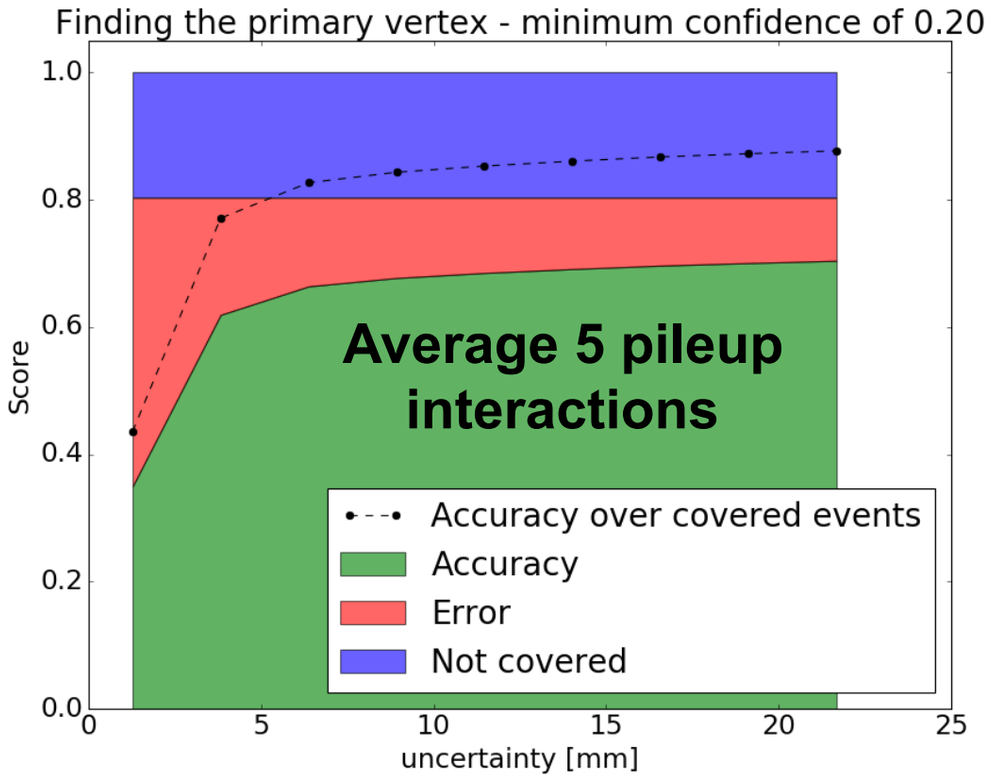


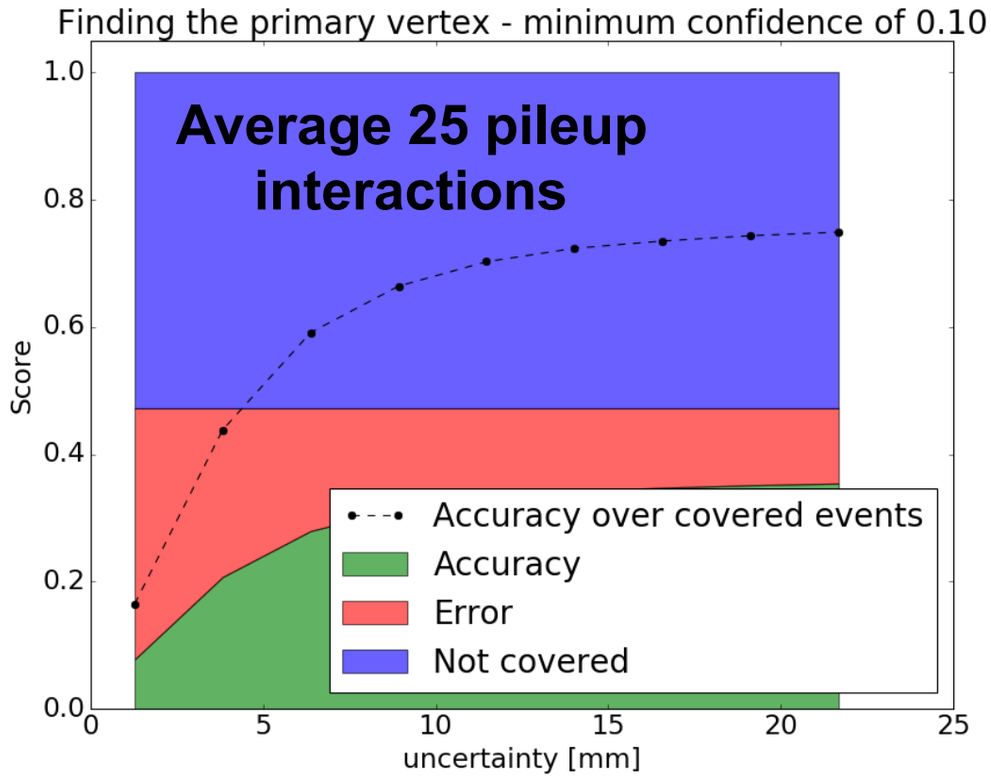


Julien Esseiva's bachelor thesis
Shown at ACAT 2017

https://gitlab.cern.ch/acts

Vertex finding with CNNs

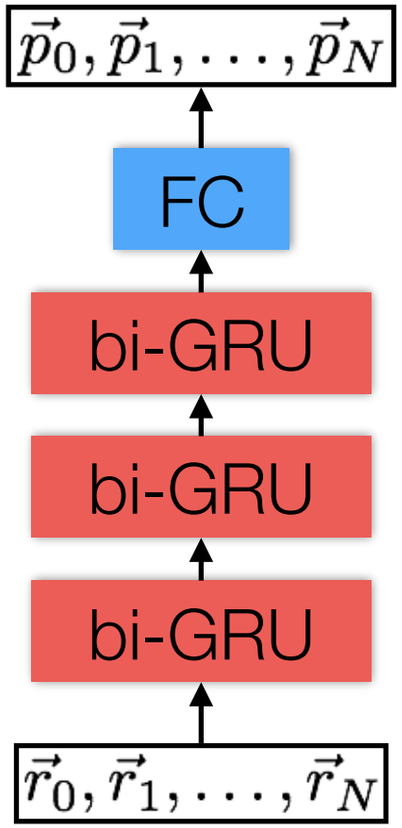


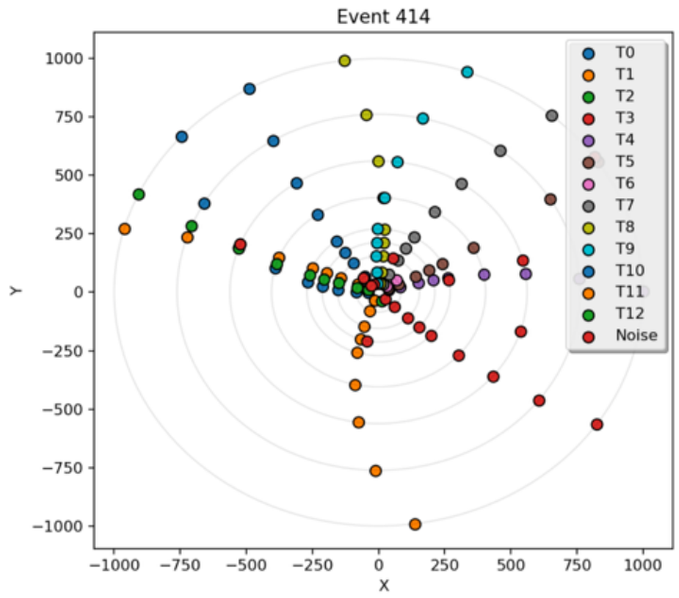


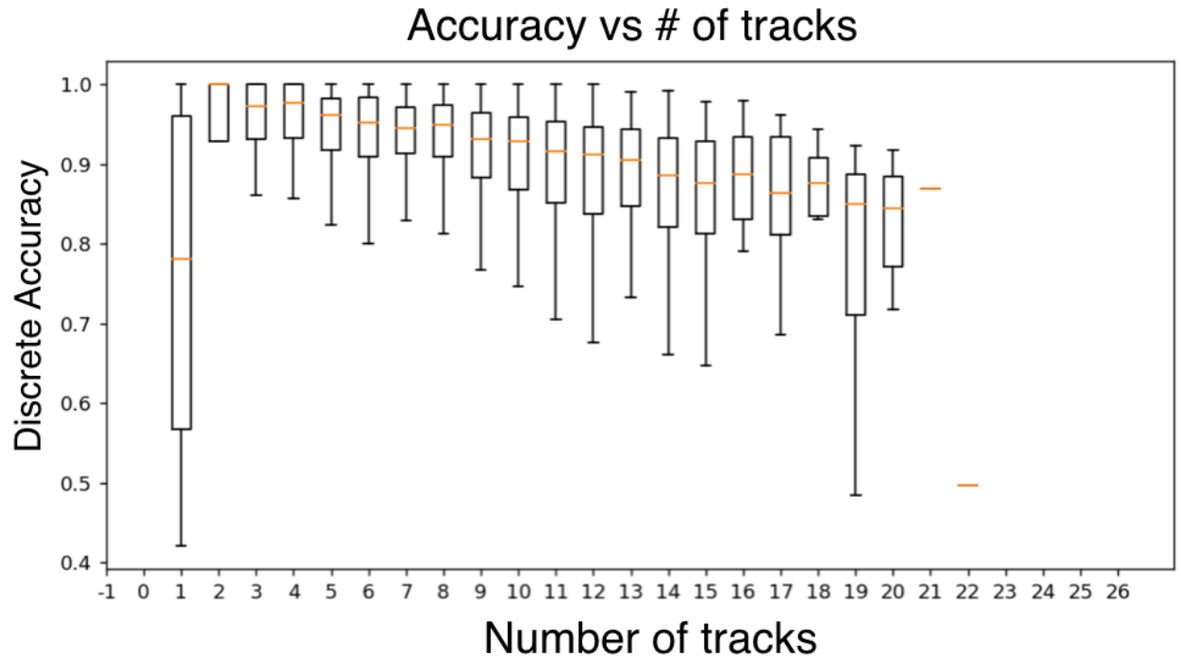
- Performs ok when finding primary vertex with only handful of pileup vertices
- Mediocre performance at μ=25
- Performs poorly at finding multiple vertices (not shown)
- Probably not good enough yet to be useful
  - Room for improvement

Julien Esseiva's bachelor thesis
Shown at ACAT 2017

Hit sequence to track assignment







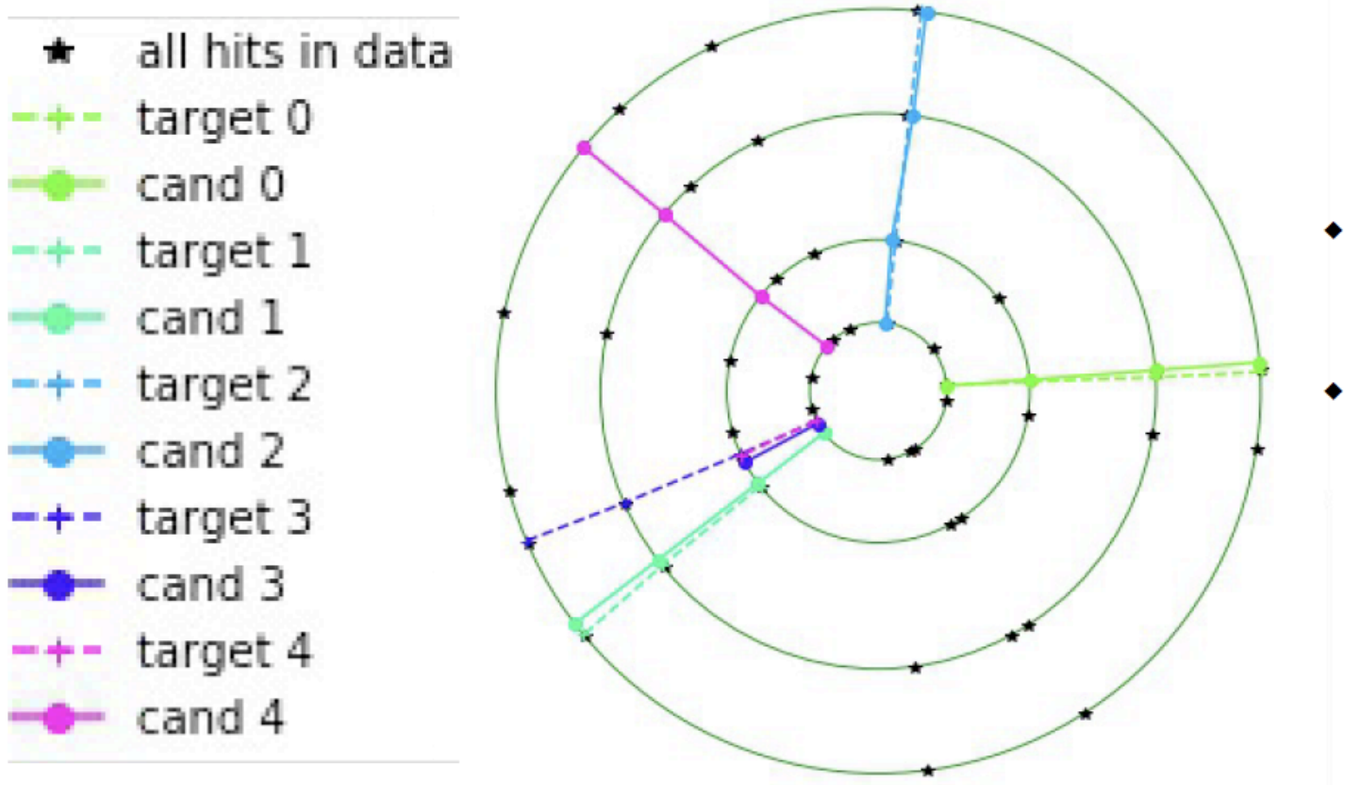
$$\vec{r} = (r, \phi, z)$$
$$\vec{p} = (p_{\mathrm{trk1}}, p_{\mathrm{trk2}}, \ldots, p_{\mathrm{trkMax}})$$

Work of Daniel R. Zurawski, Keshav Kapoor, interns at FNAL. Shown at ACAT 2017

- Sort *all* hits in an event according to position
- Feed hits into a few layers of bi-directional recurrent net (GRU)
- Output is a set of assignment probabilities to track groups
  - Ordering of output track categories is similarly sorted as hits
  - Requires assumed maximum number of tracks

seq-2-seq tracking

- Input sequence of hits per layers (one sequence per layer)
  - One LSTM cell per layer
- Output sequence of hits per candidates
  - Final LSTM runs for as many candidates the model can predict



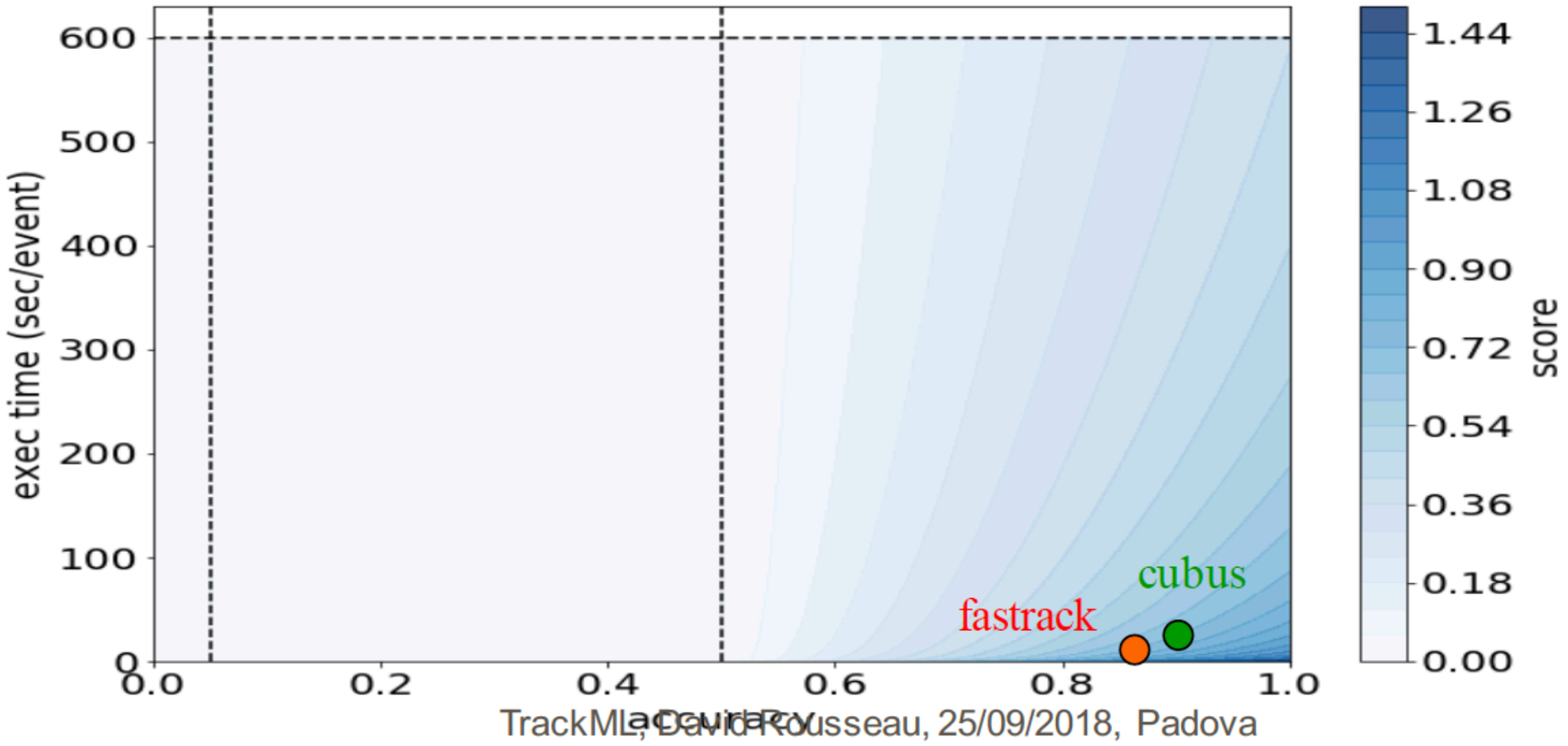
- Restricted to 4 layers (with seeding in mind)
- Full performance evaluation still to be done

From Jean-Roch Vlimant (CHEP 2018)

Backup

Throughput phase

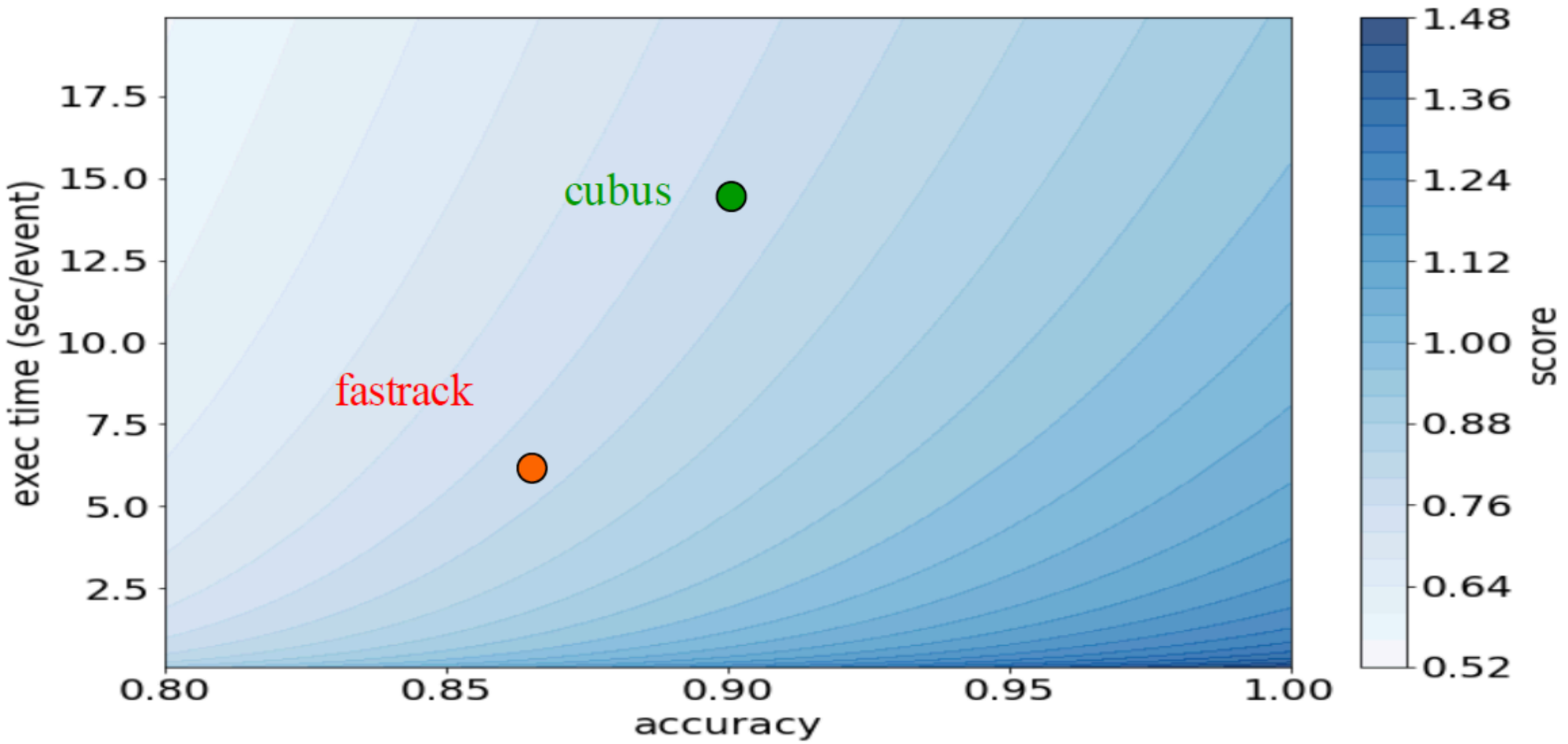
- Launched last week until 6th November on Codalab
- Participants submit their code, not their solution
- Docker with 4GB memory and 2 processors
- Codalab runs the code, evaluate accuracy (as before) and time
- *Ranking score* = $\sqrt{}$*log*(1 + *600*/*time*) ∗ (*accuracy* − *0.5*)^2
- https://competitions.codalab.org/competitions/20112



From David Rousseau

Throughput phase LB

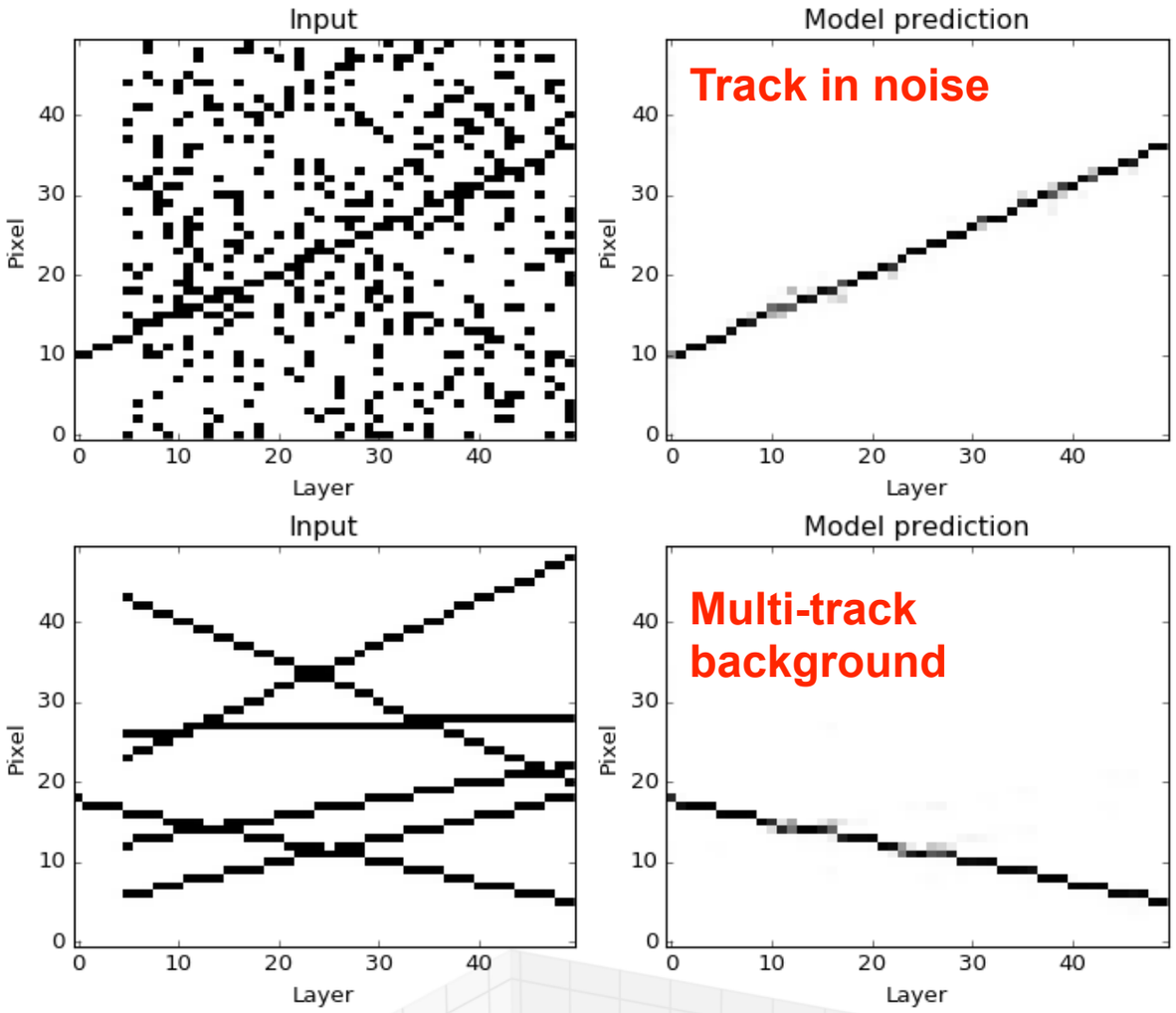
- Current ATLAS code takes 250 s for ~95% efficiency but
  - Different geometry
  - Complete track parameters (Kalman filter)
- ➔still this is exciting!

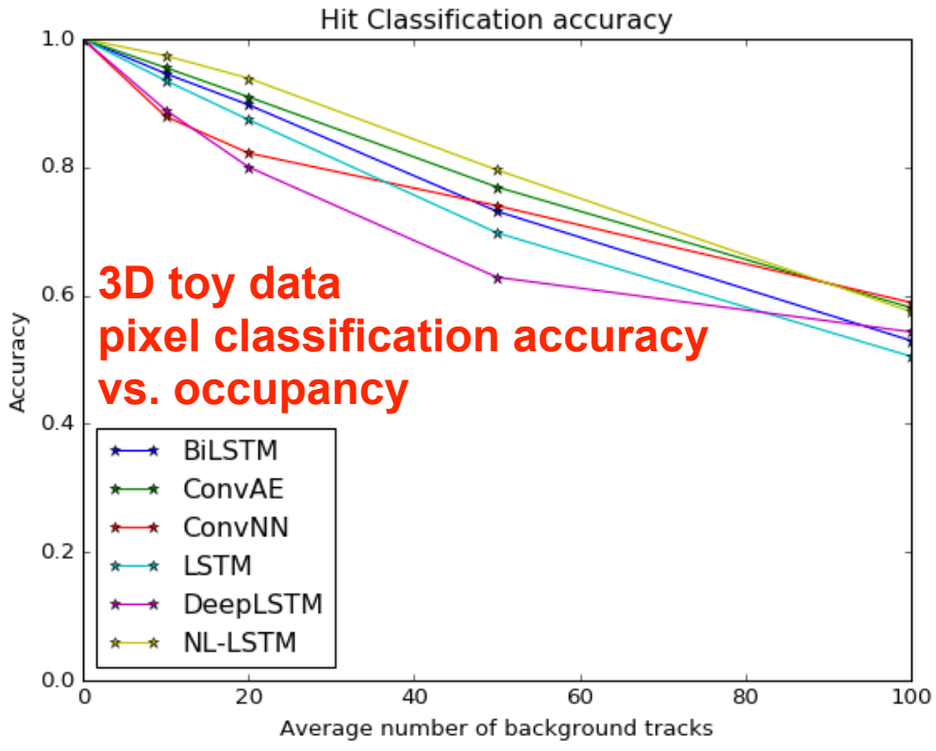


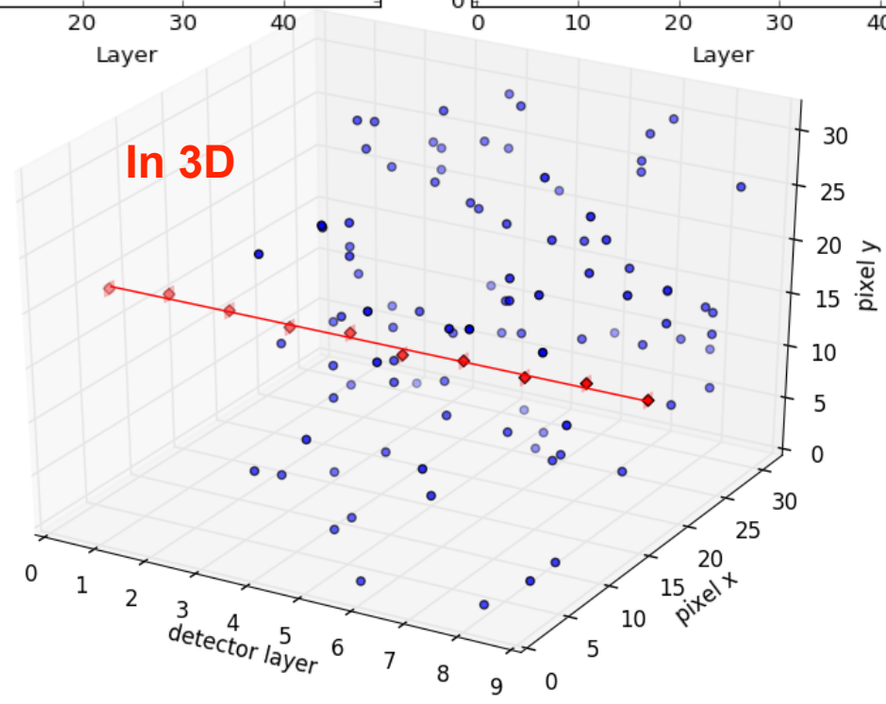
From David Rousseau

Image based tracking

- Image segmentation task on very simple 2D and 3D toy data
- RNNs and CNNs can find the pixels belonging to a desired track
- Performance degrades with increased occupancy

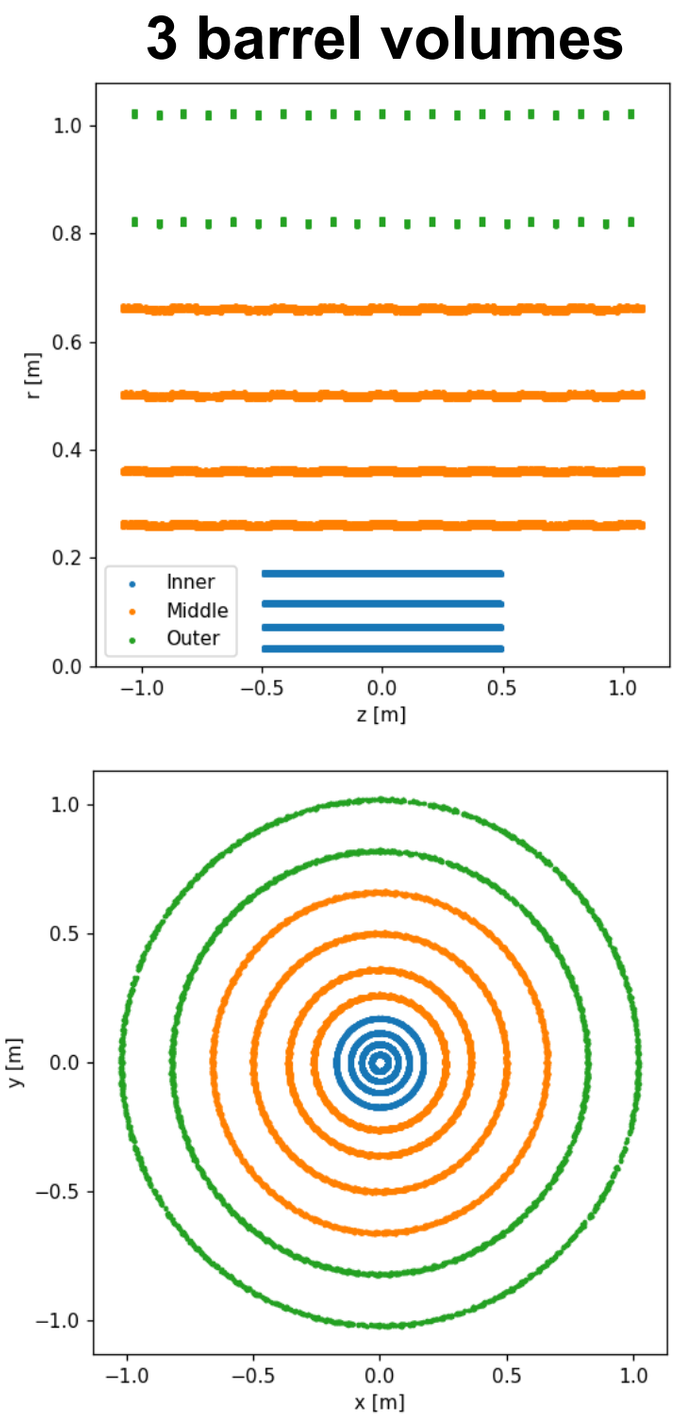


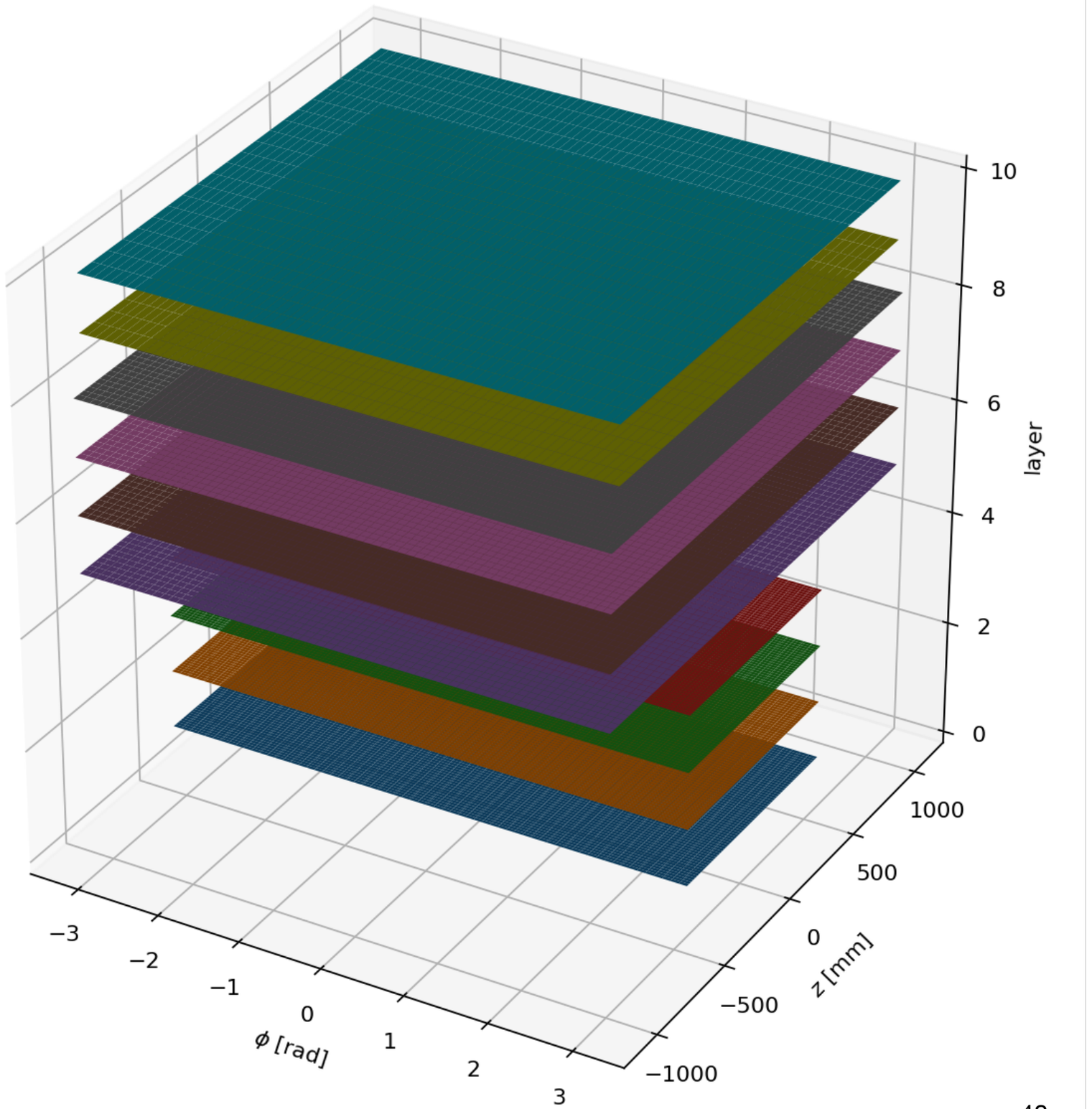




CTD-WIT 2017: https://doi.org/10.1051/epjconf/201715000003
Code: https://github.com/HEPTrkX/heptrkx-ctd

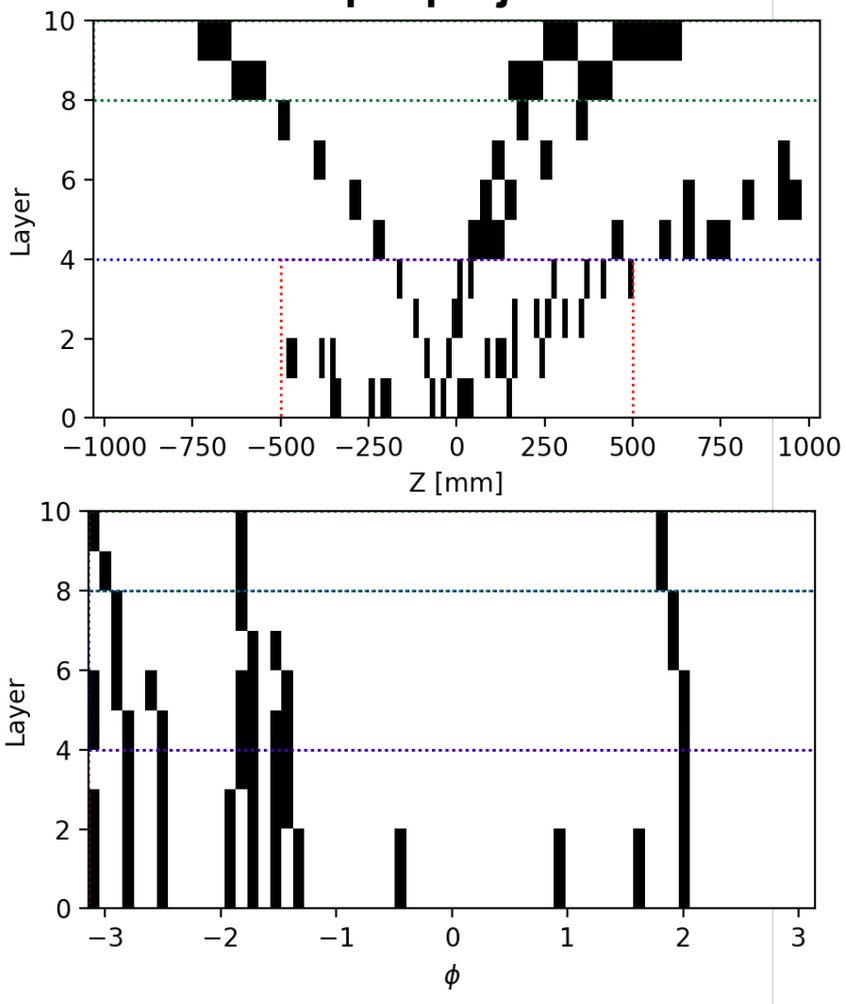
ACTS detector images

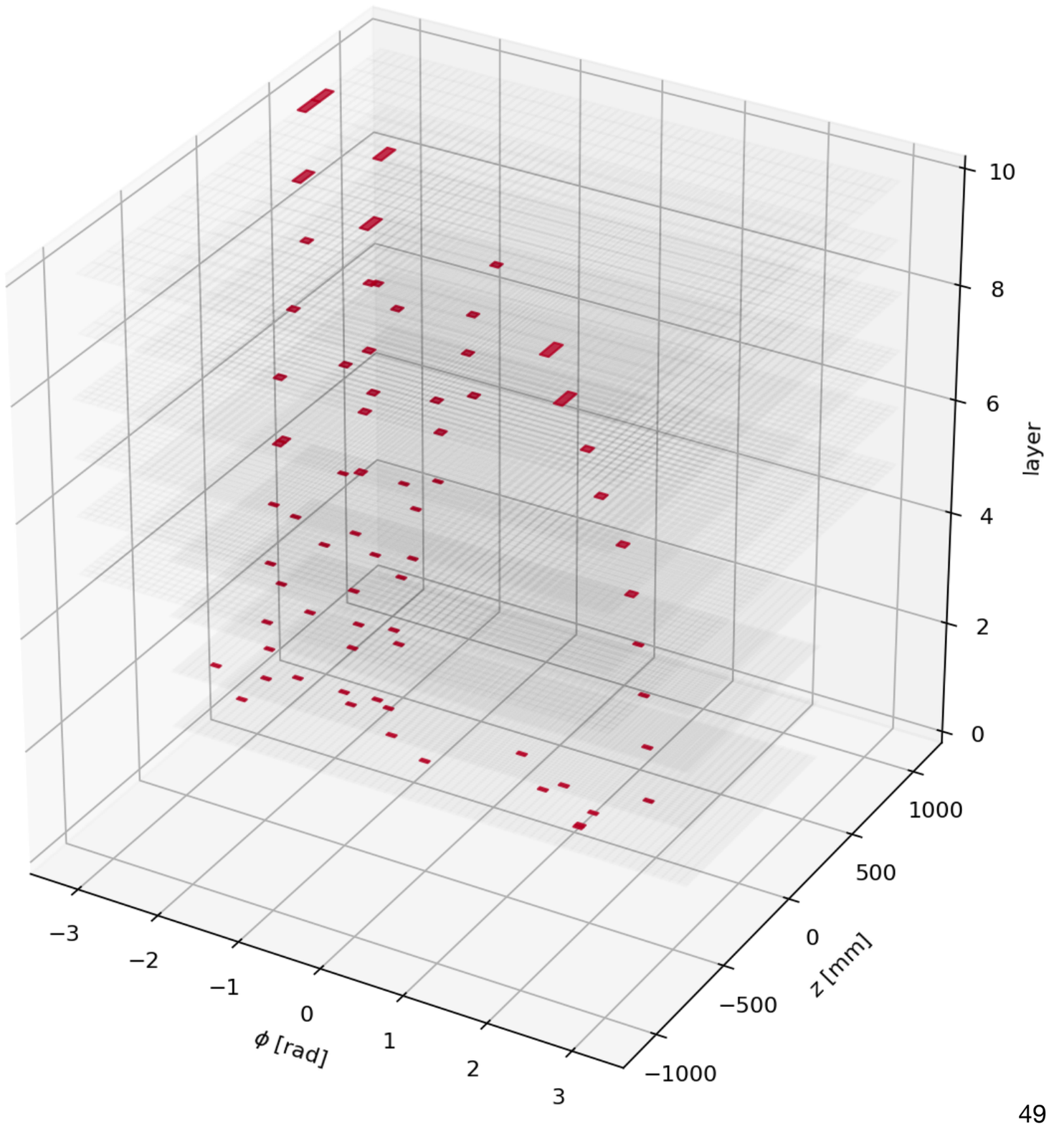




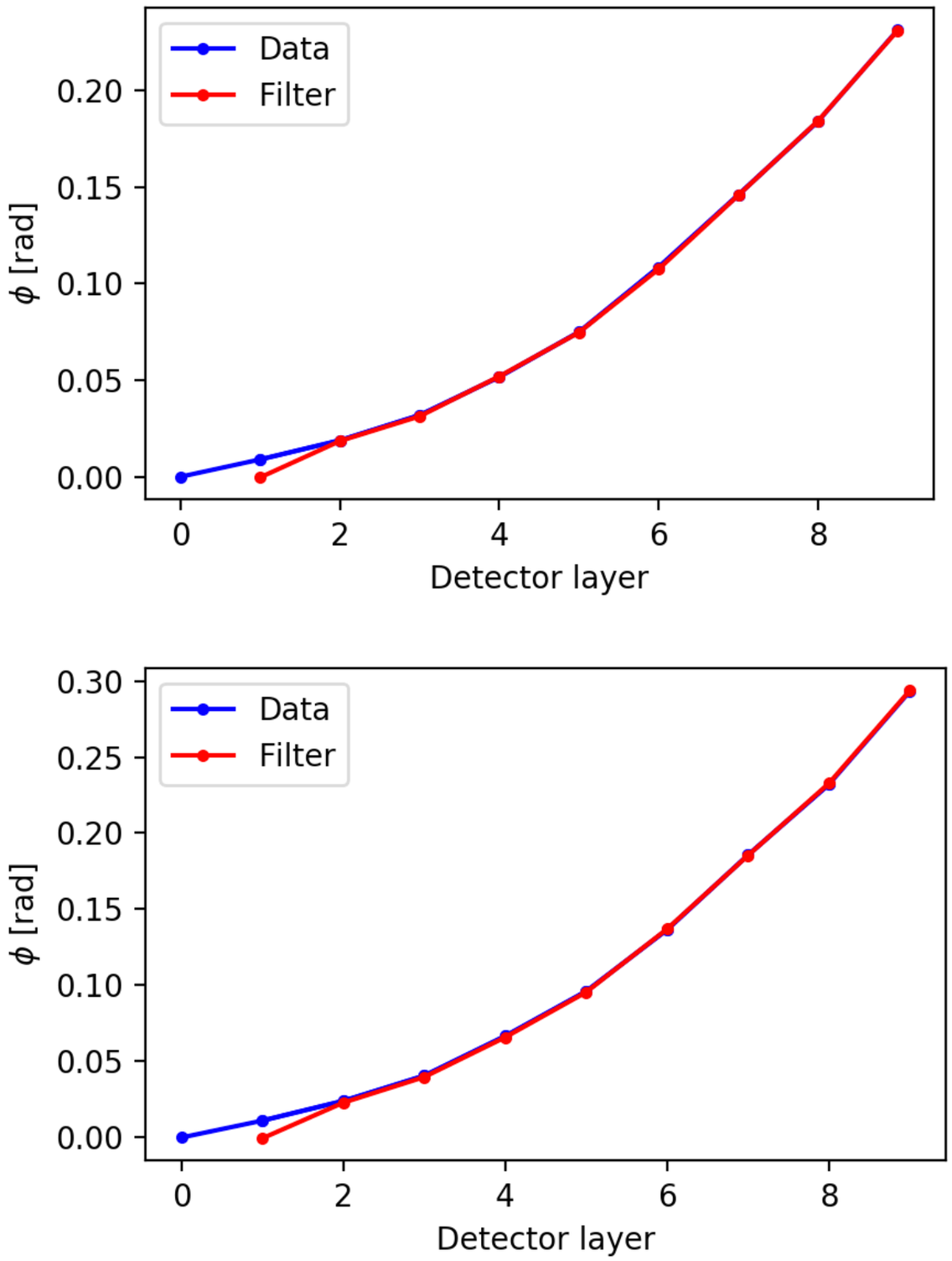
ACTS detector images

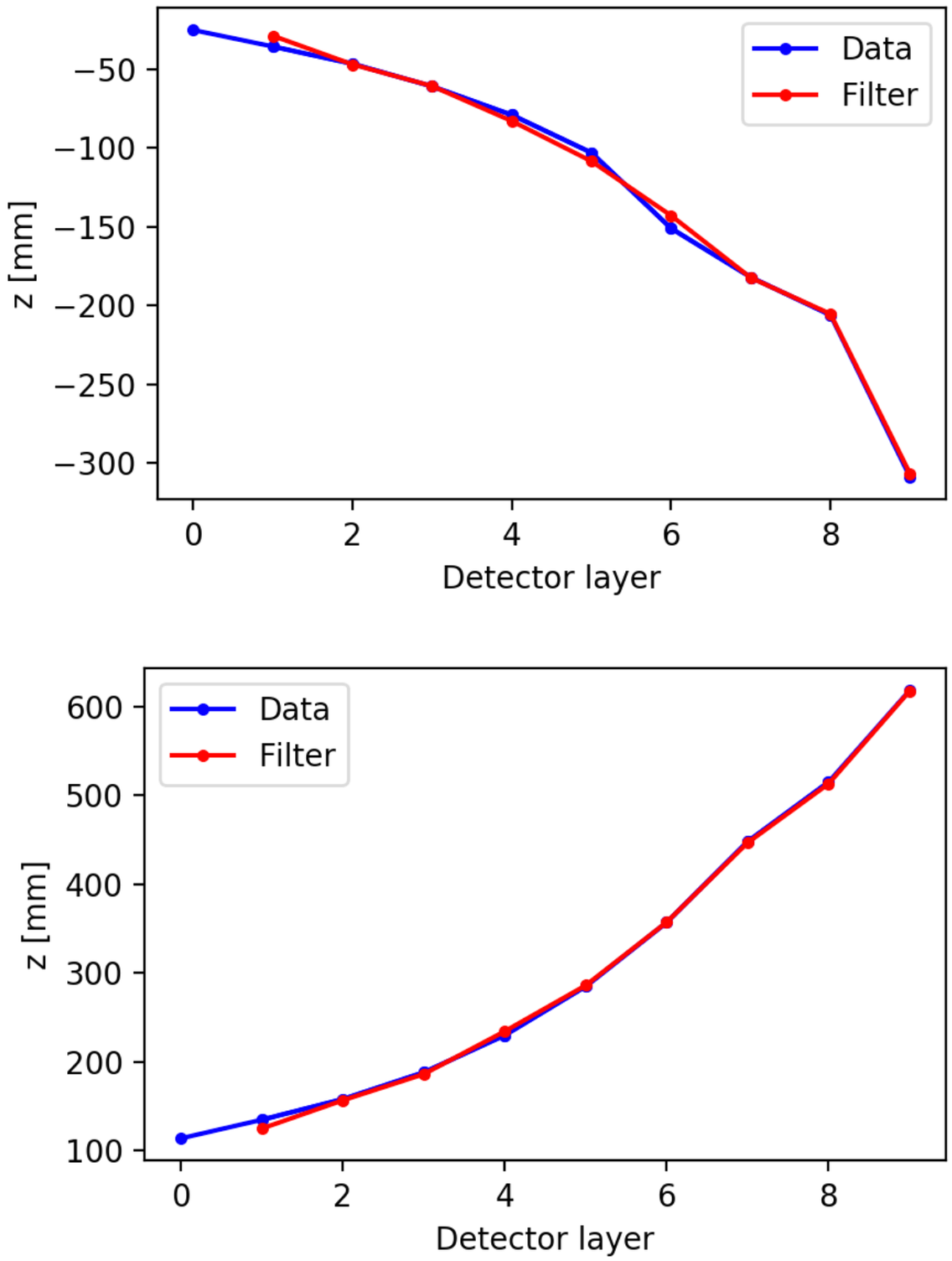
Binned into 3 detector images



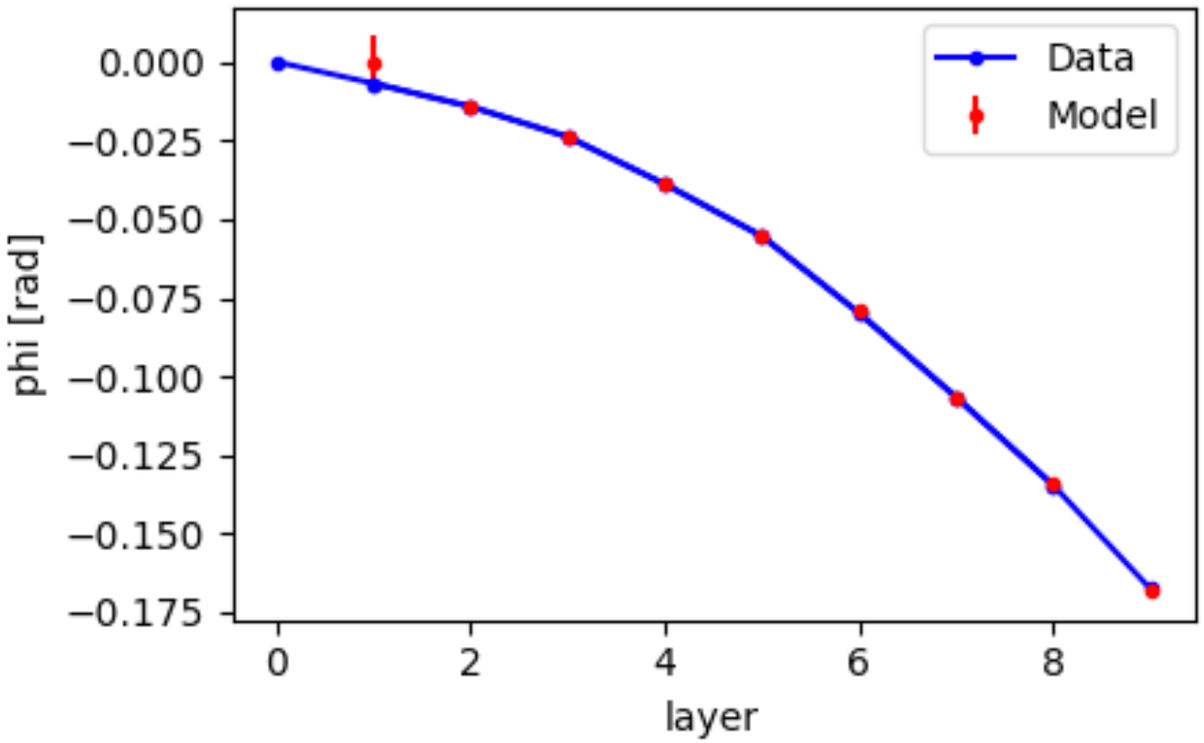


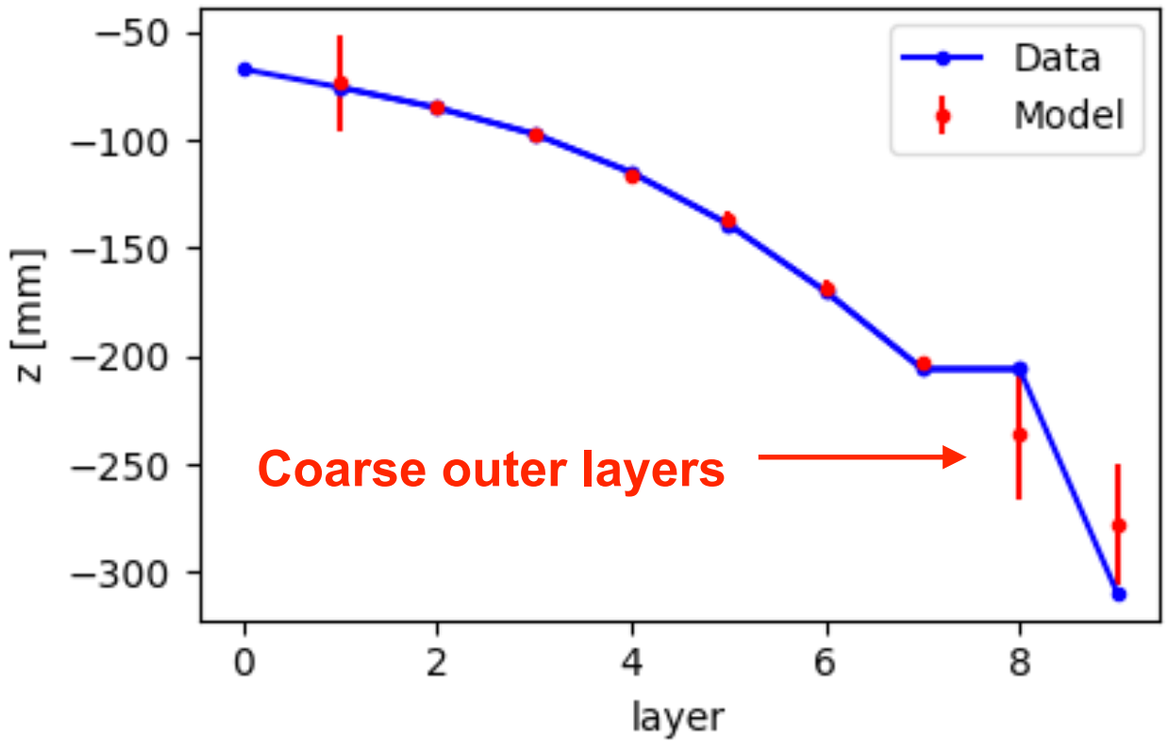
RNN hit predictor examples





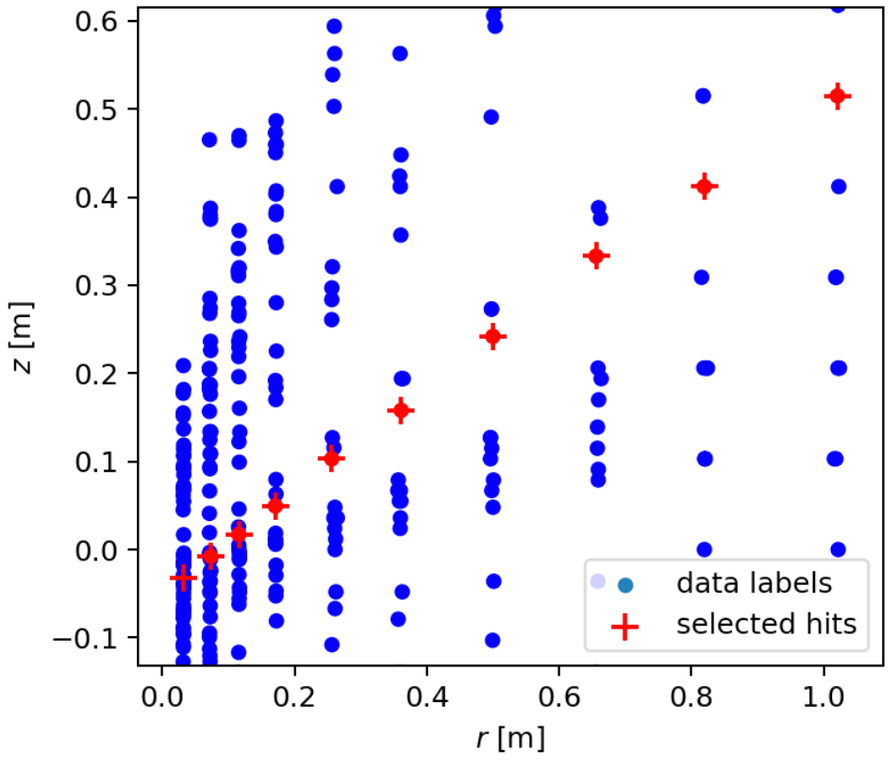
RNN Gaussian hit predictor examples





Can these be used to build tracks?

- We tested on simple low-occupancy data with naive approach
  - no combinatorial branching
  - But promising results
- Full algorithm in development



RNN hit predictor models

```
HitPredictor(
  (lstm): LSTM(3, 32, batch_first=True)
  (fc): Linear(in_features=32, out_features=2)
)
Parameters: 4802
```

```
HitGausPredictor(
  (lstm): LSTM(3, 32, batch_first=True)
  (fc): Linear(in_features=32, out_features=5)
)
Parameters: 4901
```

Geometric deep learning

http://geometricdeeplearning.com/

- **Models that learn on graphs and manifolds**
  - Message-passing architectures allow to build localized, hierarchical features
  - Can extend desirable CNN properties to graphs
- **Some relevant examples:**
  - Semi-Supervised Classification with Graph Convolutional Networks
  - Variational Graph Auto-Encoders
  - Interaction Networks for Learning about Objects, Relations and Physics
  - Neural Message Passing for Jet Physics

Architecture details

- Inputs:

$X \longrightarrow$ (N x D) node feature matrix

$R_i \longrightarrow$ (N x E) association matrix of nodes to input edges

$R_o \longrightarrow$ (N x E) association matrix of nodes to output edges

- The edge network is a 2-layer MLP with tanh and sigmoid activations:

$$w = f_{\text{edge}}(R_i^T X, R_o^T X) \longrightarrow \text{(E) edge weight array}$$

- The node network is a 2-layer MLP with tanh activations:

$$X' = f_{\text{node}}\Big((R_i \odot w)R_o^T X, (R_o \odot w)R_i^T X, X\Big) \longrightarrow \text{(N x D) node features}$$

- Outputs
  - Node classifier: binary classifier layer gives score for each node
  - Segment classifier: final edge network application gives score for each edge

GNN hit classifier

```
NodeClassifier(
  (input_network): Sequential(
    (0): Linear(in_features=4, out_features=64)
    (1): Tanh()
  )
  (edge_network): EdgeNetwork(
    (network): Sequential(
      (0): Linear(in_features=136, out_features=64)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=1)
      (3): Sigmoid()
    )
  )
  (node_network): NodeNetwork(
    (network): Sequential(
      (0): Linear(in_features=204, out_features=64)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=64)
      (3): Tanh()
    )
  )
  (output_network): Sequential(
    (0): Linear(in_features=68, out_features=1)
    (1): Sigmoid()
  )
)
Parameters: 26502
```

GNN segment classifier

```
SegmentClassifier (
  (input_network): Sequential (
    (0): Linear (3 -> 32)
    (1): Tanh ()
  )
  (edge_network): EdgeNetwork (
    (network): Sequential (
      (0): Linear (70 -> 32)
      (1): Tanh ()
      (2): Linear (32 -> 1)
      (3): Sigmoid ()
    )
  )
  (node_network): NodeNetwork (
    (network): Sequential (
      (0): Linear (105 -> 32)
      (1): Tanh ()
      (2): Linear (32 -> 32)
      (3): Tanh ()
    )
  )
)
Parameters: 6881
```