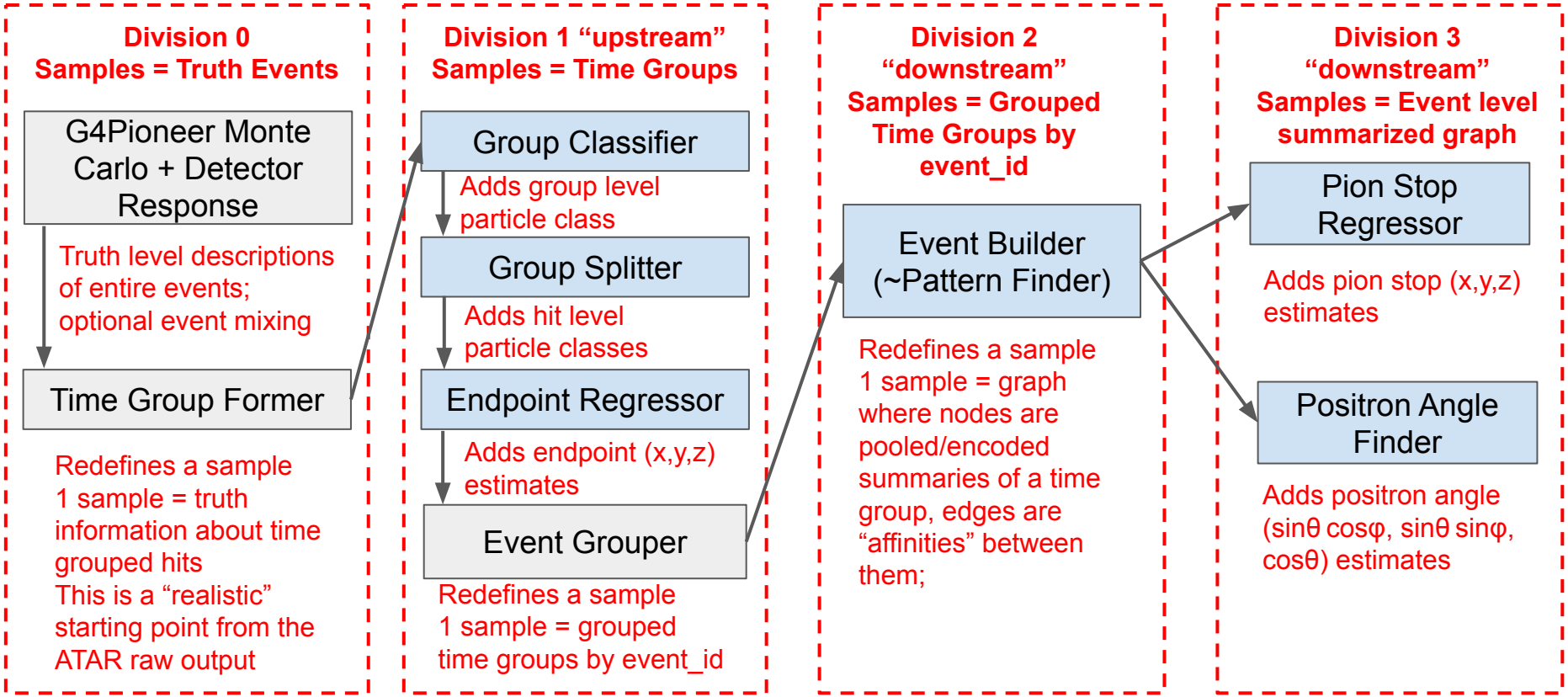
ATAR ML Based PIONEER Reconstruction as of 01/07/2026

Jack Carlton
University of Kentucky

= Not an ML stage

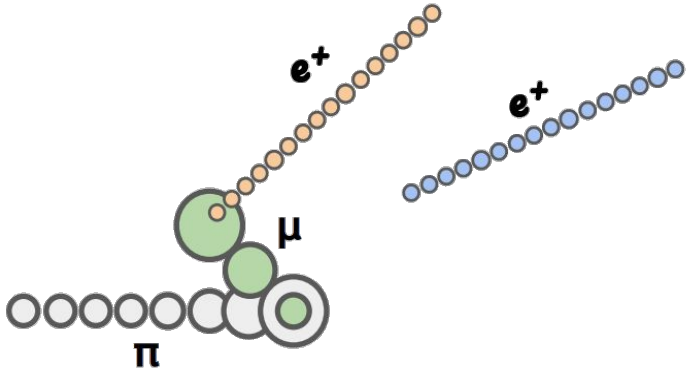
= An ML (trained) stage

Monte Carlo → ATAR ML Reconstruction



G4Pioneer Monte Carlo + Detector Response

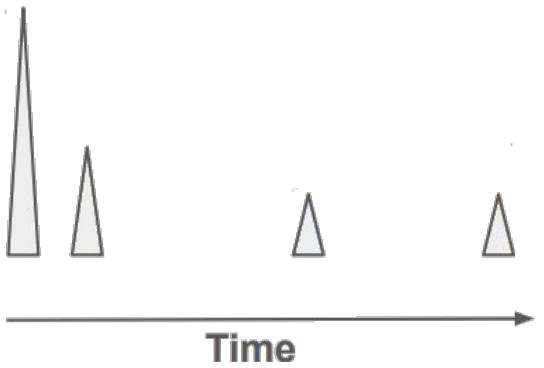
- **G4Pioneer Monte Carlo** simulates the underlying physics
  - Geant4 Fork
- **Detector Response** converts truth into detector-level signals
  - Depends on detector geometry
  - Handles event mixing
- ML training requires both detector-level outputs and truth-level information
  - Current Detector Response does not expose all required quantities cleanly
  - TODO: Add a Detector Response mode that writes all required detector + truth data to an RNTuple, forming a stable input for ML dataset construction



Cartoon example of detector response with truth information added to hits

Time Group Former

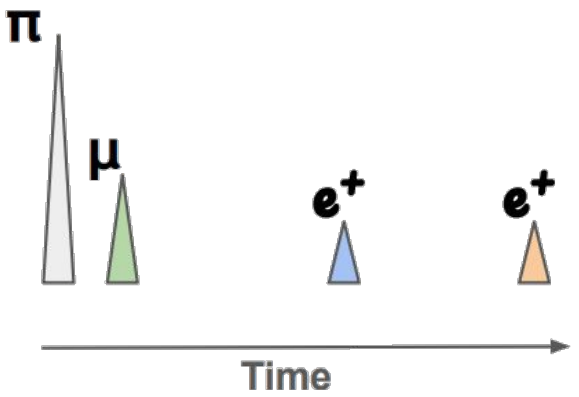
- Detector response events are split into time-based clusters
  - This stage determines the method we use to cluster the hits
- Truth-level information (e.g. particle labels) is retained for ML targets
- Each time group is one ML sample
- A time group contains:
  - Variable-length hit-level features
  - Fixed group-level summary features



Cartoon example of time group spikes in energy deposition

Group Classifier

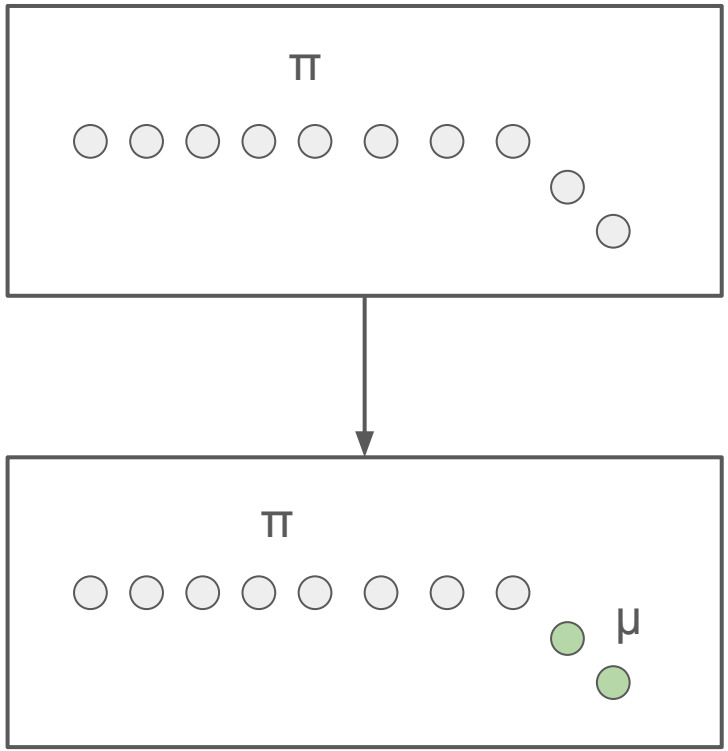
- **Multi-label GNN classifier**
- Inputs:
  - Detector response information represented as a graph
- Targets:
  - Truth level representation of what particles are in the group
  - Particularly: `[pionInGroup, muonInGroup, MIPinGroup]`
- Outputs:
  - Predicted probability that the time group belongs to each class `[prob_pionInGroup, prob_muonInGroup, prob_MIPinGroup]`



Cartoon example of time group spikes in energy deposition, now labeled by group classifier

Group Splitter

- **Multi-label GNN classifier**
- Inputs:
  - Detector response information represented as a graph
  - Predicted time group level particle class
- Targets:
  - Truth level representation of the particle type for each hit
  - Particularly, for each hit in the graph: `[[is_pion, is_muon, is_mip], [is_pion, is_muon, is_mip], …]`
- Outputs:
  - Predicted probability that each hit belongs to each class `[[prob_pion, prob_muon, prob_mip], [prob_pion, prob_muon, prob_mip], …]`



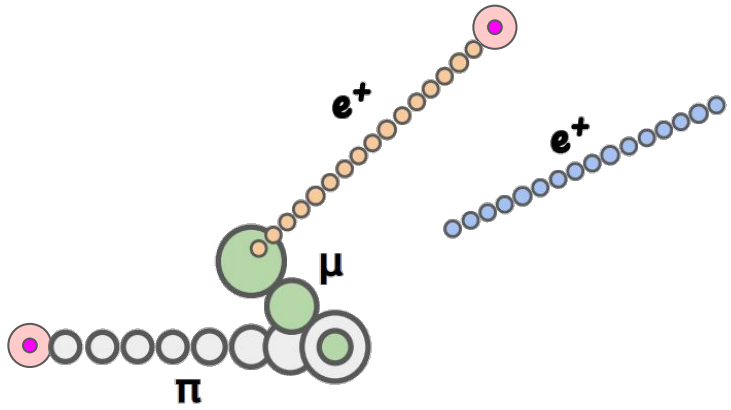
Cartoon example of how a time group may be split into multiple particle classifications

Endpoint Regressor

- **Quantile GNN Regressor**
- Inputs:
  - Detector response information represented as a graph
  - Predicted time group level particle class
  - Predicted hit level particle classes
- Targets:
  - Truth level endpoints
  - Particularly: `[[x_start, y_start, z_start], [x_end, y_end, z_end]]`
- Outputs:
  - Predicted endpoints with quantiles `[[[x_start_q1, y_start_q1, z_start_q1], [x_end_q1, y_end_q1, z_end_q1]], ...]`

= Median endpoint estimate

= Error range determined from quantiles

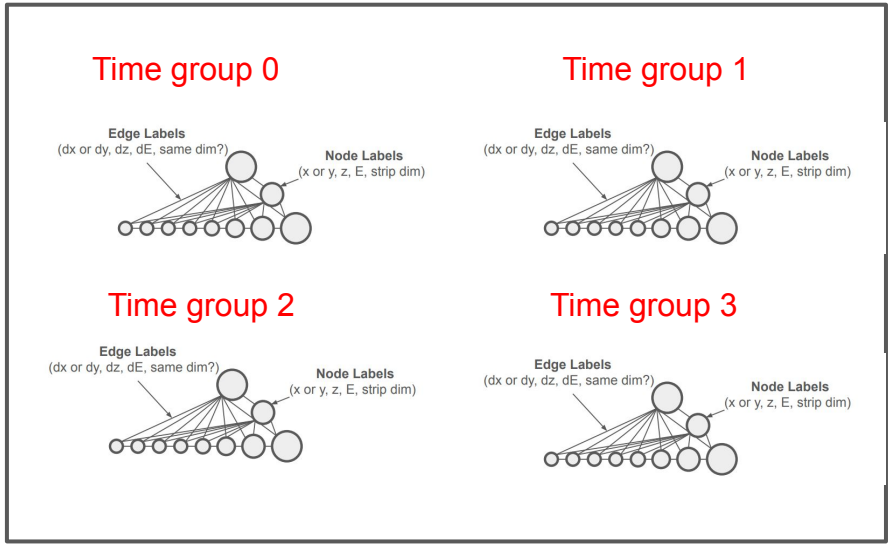


Cartoon example of how we estimate endpoints with uncertainties given by the quantiles

Event Grouper

- Re-groups time groups into events by event_id
- Forms a list of graphs
- Side effect:
  - The ML reconstruction is sensitive to how we define an ATAR event
  - The traditional reconstruction has this same problem (?, need to check if the traditional reco uses event_id at all)

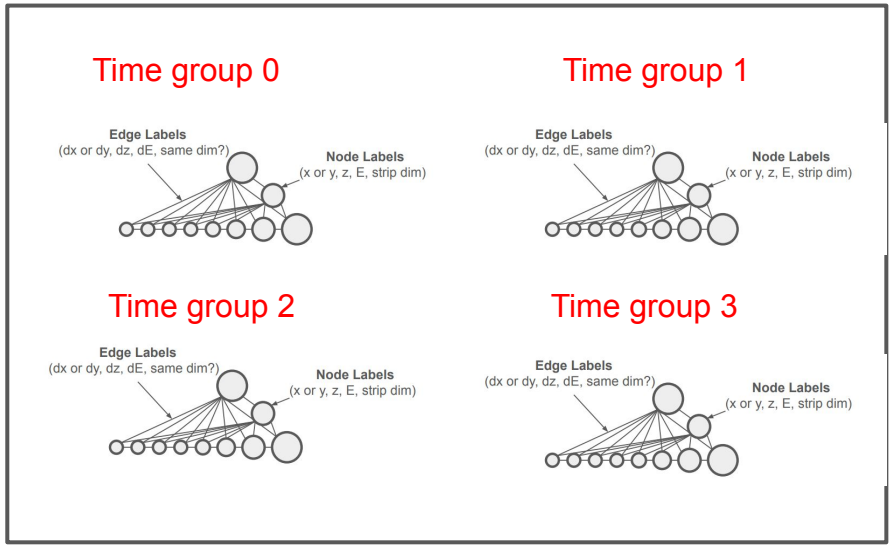
All time groups with a given event_id



Cartoon example of an event, or group of time groups

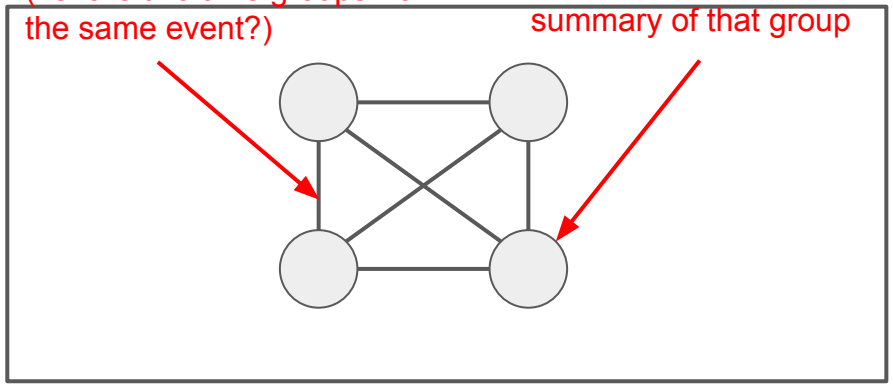
Event Builder

- **Affinity GNN classifier**
- Inputs:
  - Detector response information represented as a graph
  - Predicted time group level particle class
  - Predicted hit level particle classes
  - Predicted endpoint quantiles
- Targets:
  - Agreement between truth level event origins between nodes; i.e. “are these two nodes from the same event”. Explicitly:
    `target_affinity[i][j] = 1 if origin_id[i] == origin_id[j]`
- Outputs:
  - Predicted “affinities” (0,1)
  - We construct a graph using this information, letting Nodes = pooled/encoded group summary
- Note:
  - This module will NOT work if there are no mixed events in the data set, it effectively trains a null-op



Edges:
Learned affinity between groups (i.e. are two time groups from the same event?)

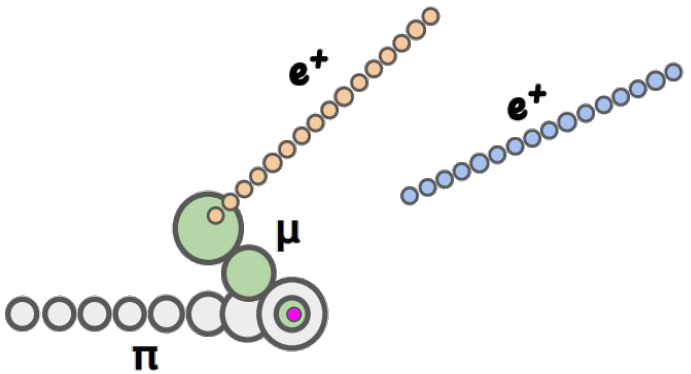
Node labels:
On per time group graph, pooled/encoded summary of that group



Pion Stop Regressor

- **GNN Regressor**
- Inputs:
  - Graph produced by event builder
- Targets:
  - Truth level pion stop
  - Particularly: `[x_stop,y_stop,z_stop]`
- Outputs:
  - Predicted pion stop `[x_stop,y_stop,z_stop]`

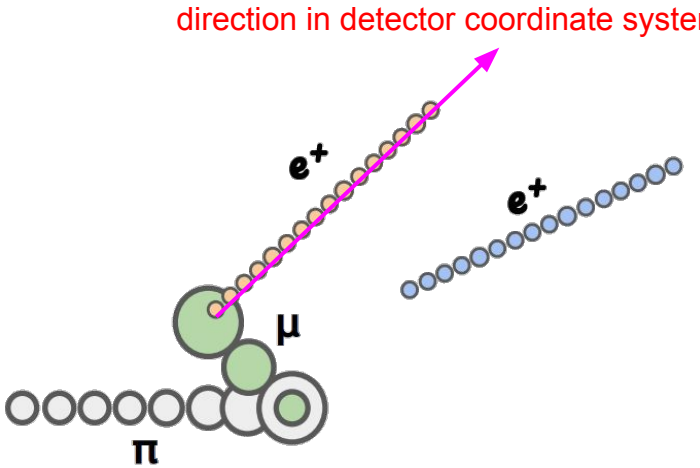
● = Pion stop estimate



Cartoon example of pion stop estimate

Positron Angle Regressor

- **GNN Regressor**
- Inputs:
  - Graph produced by event builder
- Targets:
  - Truth level direction vector for positron angle
  - Particularly:
    $[\sin\theta\cos\varphi,\ \sin\theta\sin\varphi,\ \cos\theta]$
- Outputs:
  - Predicted direction vector for positron angle
    $[\sin\theta\cos\varphi,\ \sin\theta\sin\varphi,\ \cos\theta]$

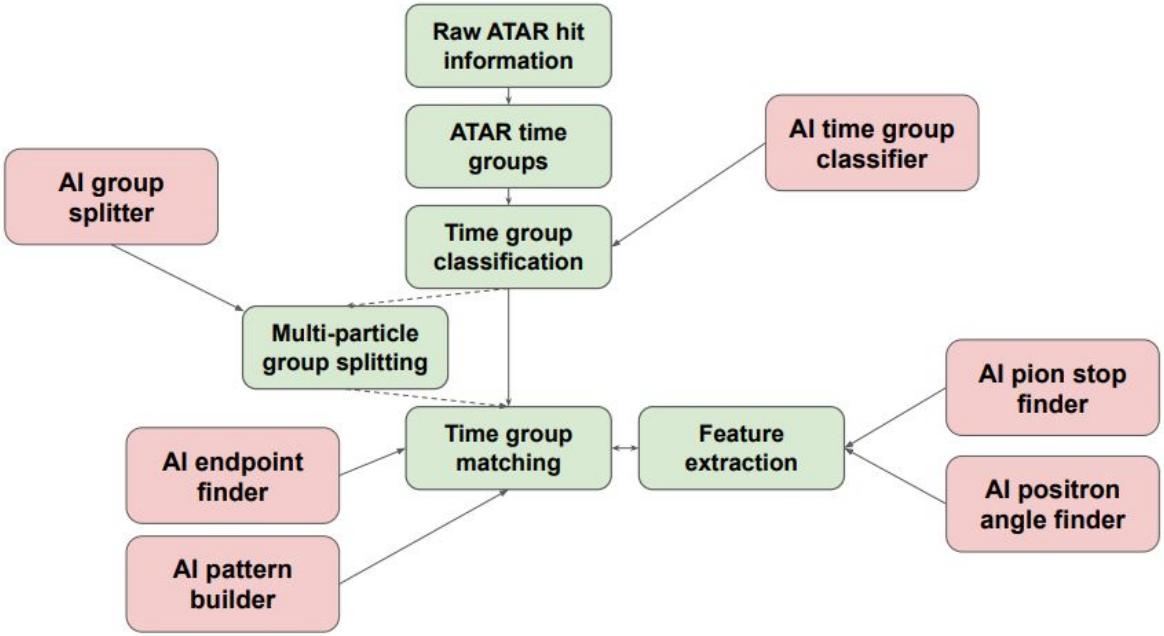


Cartoon example of positron angle estimate

Auxiliary Slides

Omar’s Diagram

- Conceptually, all the same stages
- I’ve just added “boundaries” to my diagram; between each phase the definition of the dataset we train our models on changes
- Much better explained in [Omar’s presentation]



Current Discrepancies

- We have the following discrepancies from what I described in the current framework, but these will be changed:
  1. Event mixing is done right before event building, so the “phase 1” stages don’t see mixed event right now
  2. There’s no “infrastructure” code that takes us from monte carlo to workable ML training data currently, Omar has his own “special” data conversion we’ve been using

Why Have Divisions?

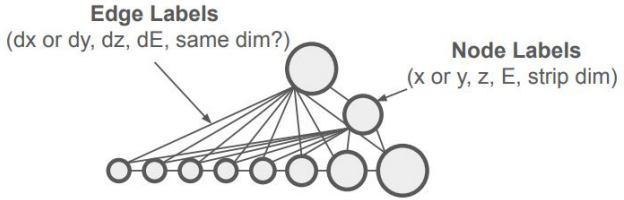
- Each pipeline stage operates on a well-defined “sample”
  - Each division defines a **dataset boundary**
    - A dataset boundary is the point at which the meaning of a row is fixed and written to storage, so downstream stages can rely on it without knowing how it was produced
    - It’s okay to create appending files or masks,
    - If you must mutate data (i.e. change your definition of a “sample” then you create another dataset boundary), then you should create a new dataset boundary or “division”
- If we don’t create clear divisions
  - Hidden coupling between stages can occur
    - I.e. you change one stage, an unrelated stage no longer works
  - Batching becomes unclear
    - Your definition of a batch changes between data boundaries
    - Memory and performance become less predictable
- In short, each division should scale separately

GraphRecord Former

- We don’t store data as their graphs, we store them as the minimal information needed to build the graph

- As a result we have a “data loading stage” before running any model where we construct a graph from the data

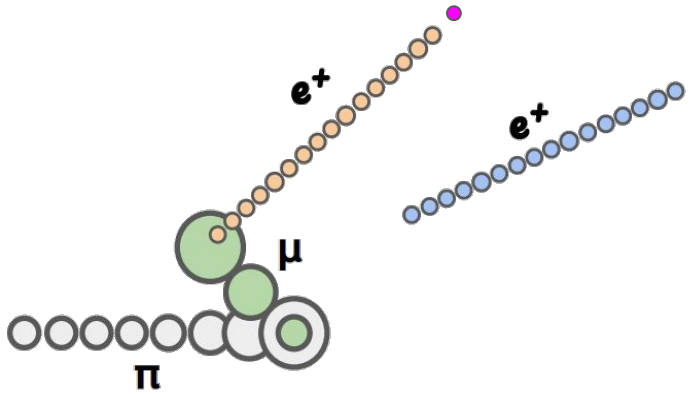
```
Columns:
_____________
event_id
theta
phi
pion_stop_x
pion_stop_y
pion_stop_z
total_pion_energy
...
hit_coord→ list < float >
hit_z→ list < float >
hit_energy→ list < float >
hit_pdg_mask→ list < int >
```



Pion Stop Prediction

- One additional processing step before the event builder runs is the pion stop prediction is computed
- This is a minor step used to help downstream models
- Pion stop predicted as just the median of the endpoint predicted by the endpoint regressor model

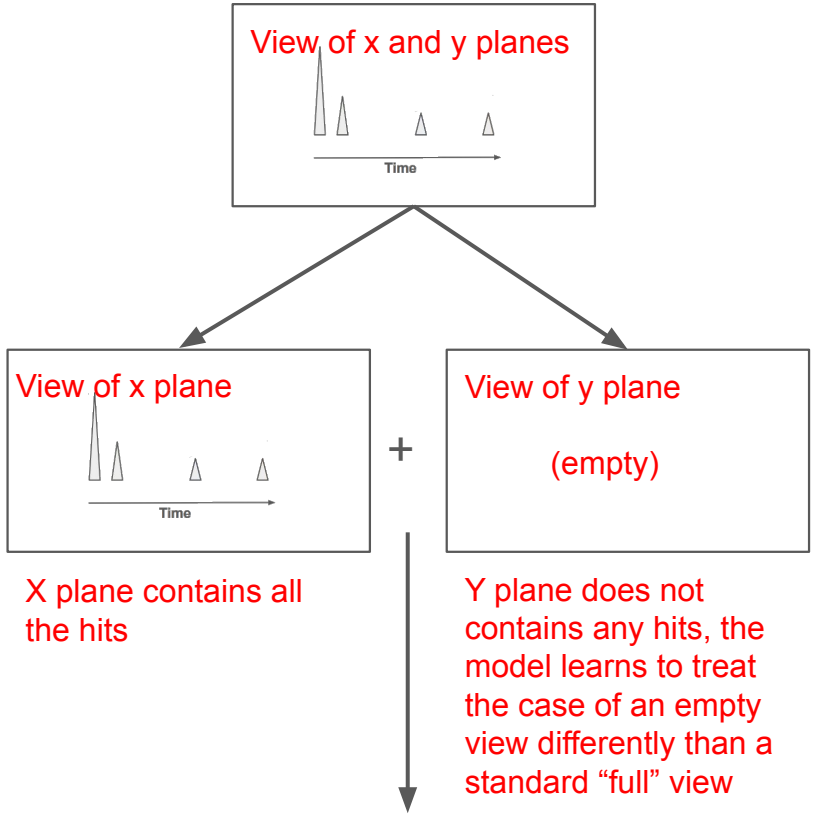
● = Median endpoint estimate



Example cartoon of the pion stop “initial guess” used to help division 2 models

Stereoscopic View

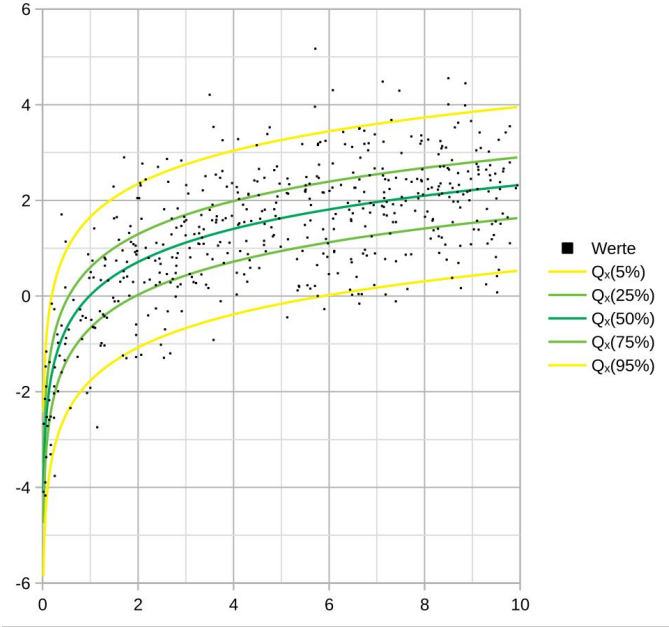
- Before each “Division 1” model “does it’s job”, it first construct a “stereoscopic” view of the data
  - Hits are split by view:
    - 0 = X plane, 1 = Y plane
  - Each view is embedded independently, then fused: [emb_view0, emb_view1, mask0, mask1]
    - mask_i = 1 if view i has hits, else 0
  - **This process is learned for each model**
- Why this helps:
  - Treats view ID as discrete routing
  - Explicitly encodes that intermediate views (e.g. “0.5”) do not exist
- Why we want this stage learnable:
  - weight each view
  - rescale per-view features
  - handle missing or weak views
- Why it’s model-specific
  - Different tasks rely on views differently
    - e.g. missing Y view matters more for endpoint finding than for particle classification



Exaggerated cartoon example of how a stereoscopic view can help in the case of missing views

Quantiles

- For regressions, quantiles are useful to extract uncertainties
- Uses a quantile regression loss function
- We can use quantile regression to have our models target a specific quantiles
  - Usually we select
    - Lower quantile = 16 ~ 1σ
    - Mid quantile = 50 ~ median
    - Upper quantile = 84 ~ +1σ



Wikipedia example of Quantile Regression. Notice how it forms confidence bands around the best fit (or median)

Event Mixing

- Event mixing is already handled in the pioneer simulation framework in the detector response
  - Effectively piles up simulated events into one event. The number of simulated events and probability of pileup are parameterized.

- Currently, the ML reconstruction has its own version of event mixing that runs before division 2; this is