SOUTH DAKOTA MINES

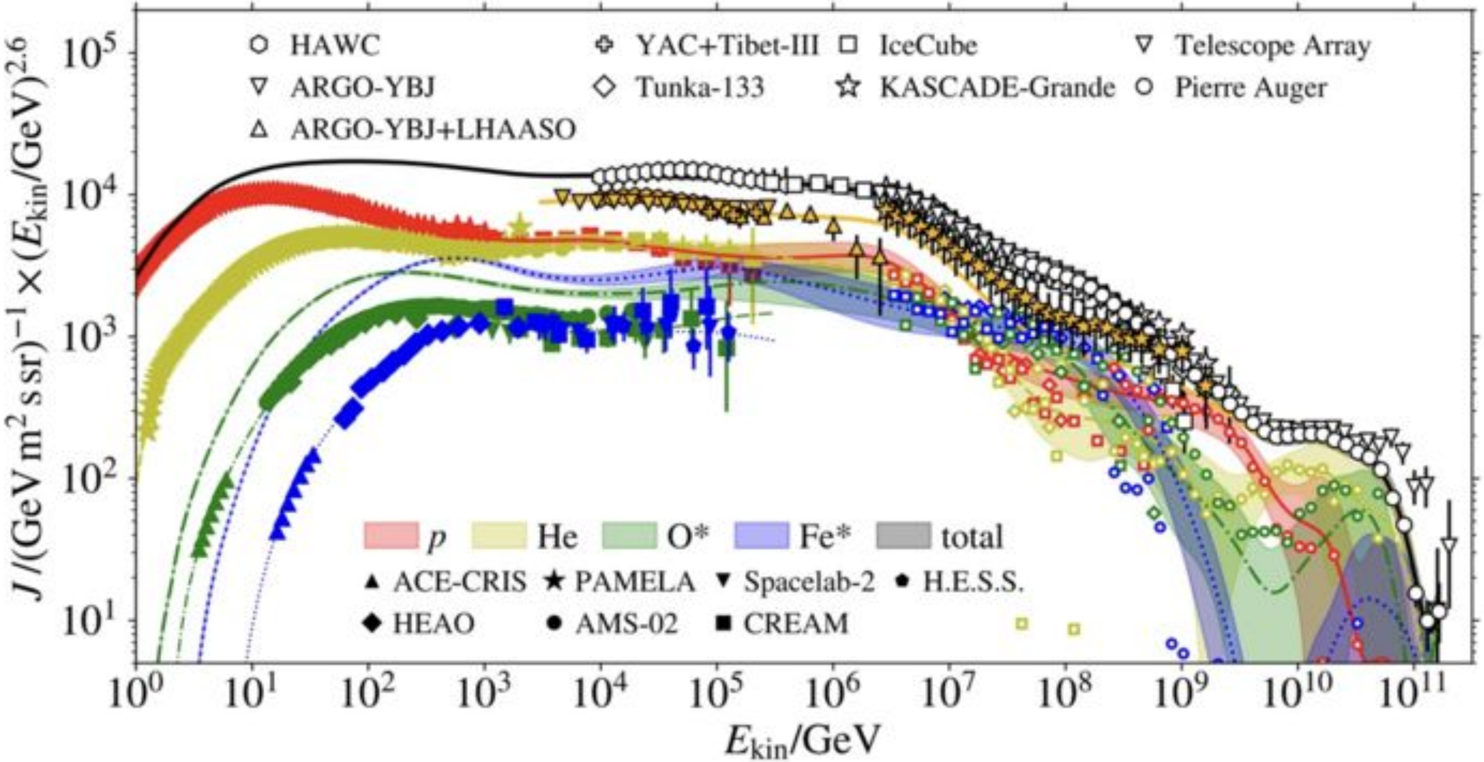
A graph neural network reconstruction for the IceAct telescopes -

Workshop on Machine Learning for Analysis of High-Energy Cosmic Particles

Larissa Paul

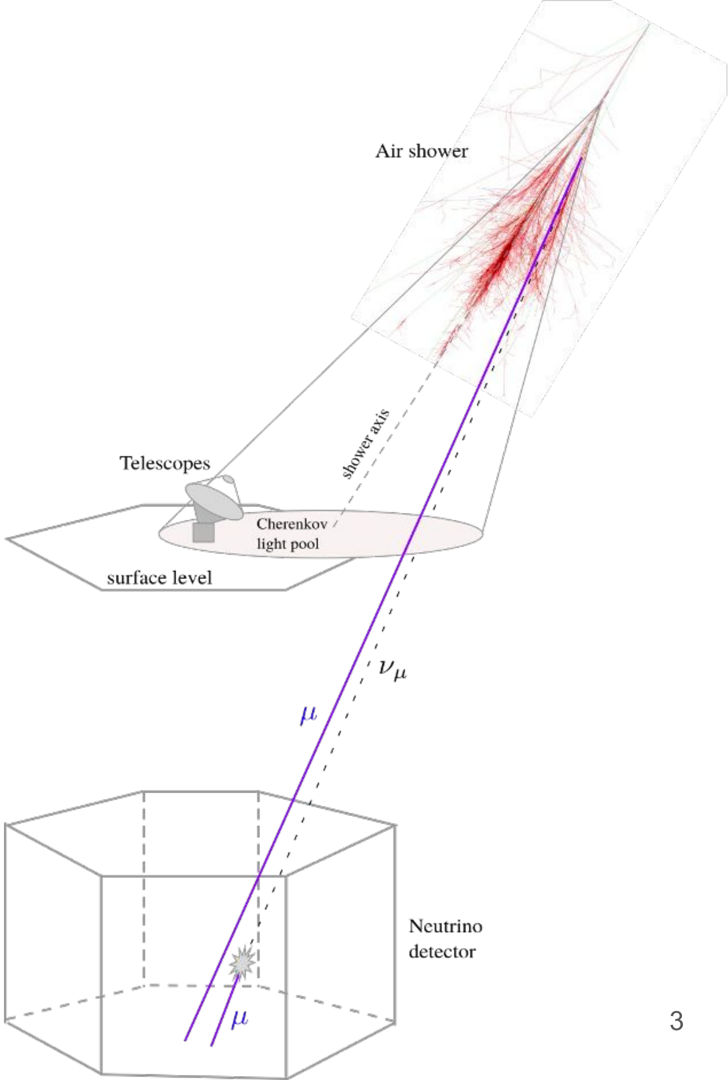
January 29th 2025

The Cosmic Ray Spectrum



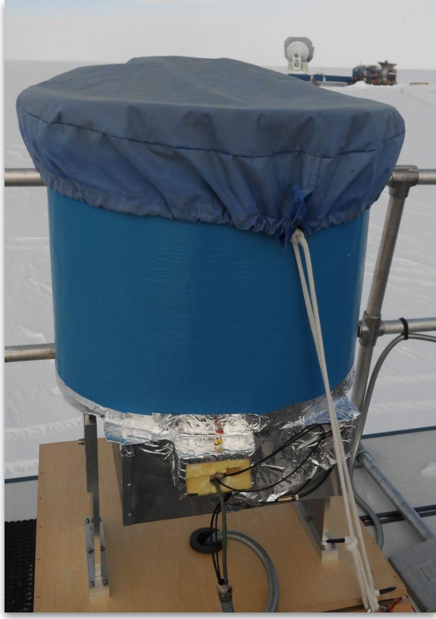
The IceAct telescopes:

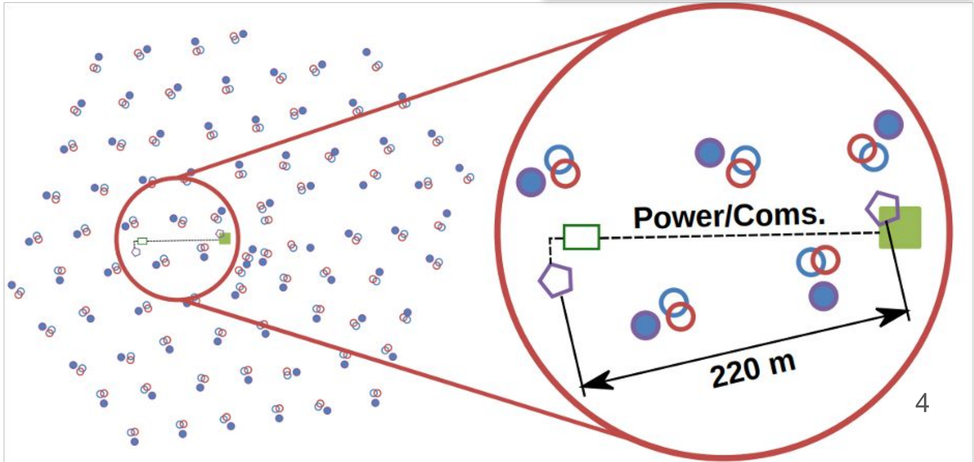
- Imaging air Cherenkov telescope with 61 pixel SiPM camera, small and robust
- 1 pixel ~ 1.5° => 1 camera ~ 12°
- Calorimetric measurement of the electromagnetic part of the air shower
- combine with particle footprint on ground level and in-ice muon reconstruction
- cross-checks of geometry and energy reconstruction for the different detector components
- hybrid cosmic ray measurement studies



The IceAct telescopes: Current installation

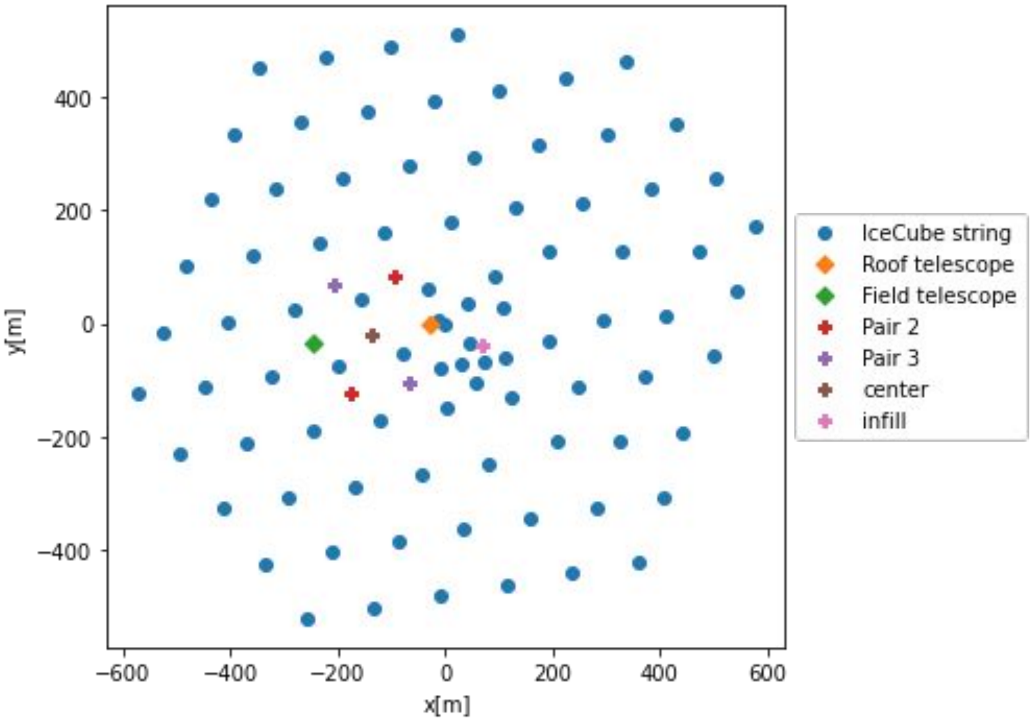
- 2 IceAct telescopes taking data since 2019
- 1 additional telescope installed in 2024
- 2 more telescopes installed January 2025





CORSIKA simulation data:

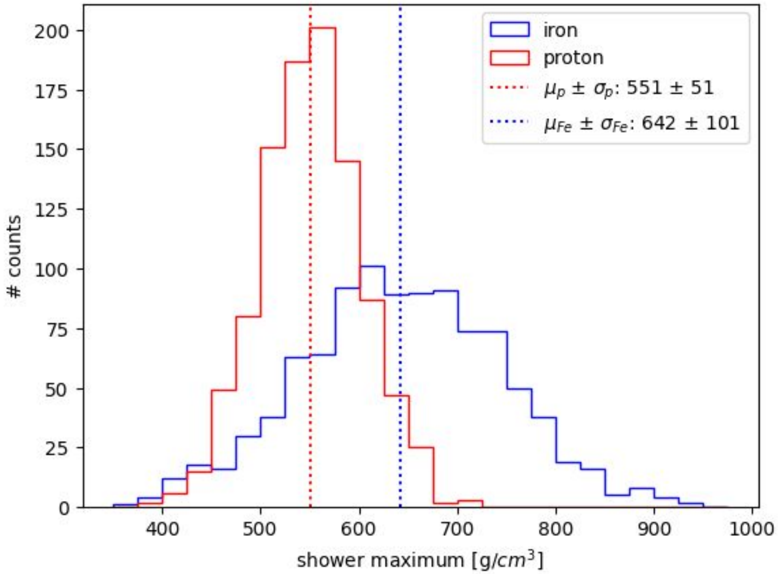
- E**-1, 3TeV-1PeV(/6PeV)
- 0-20 zenith
- Interaction model: Sybill2.3c
- round array with increasing radius:
  - 3.5 ≤ log10(E) ≤ 4
    => r = 250m;
  - log10(E) > 4
    => dlog10(E)=0.25
    => dr = 50m



Shower maximum as measure for the composition measurement:

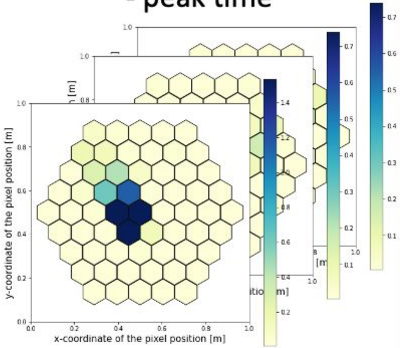
- The telescopes measure the el. mag. part of the air shower
  => therefore they are able to measure the height of the shower maximum
- For this talk:
  Figure of merit is used as goodness of reconstruction of the maximum

$$FOM = \frac{|\mu_{Fe} - \mu_p|}{\sqrt{\sigma_p^2 + \sigma_{Fe}^2}},$$



Standard graph neural network reconstruction

Graphs features:
- 61 nodes = 61 pixel
- each node has 4 layers:
  - pixel x position
  - pixel y position
  - peak height
  - peak time



Graph properties:
- Number of nodes can differ between events
- No fixed structure needed

(stereo reconstruction: 122 nodes)

Hidden layers:
Matrix convolution with the graph features

Adjacency matrix :
each pixels knows itself and its neighbors

Edges between nodes:
same connection between all pixels

Model:
2 GatedGraphConv. Layer
1 GlobalSumPool Layer
3 Dense Layer

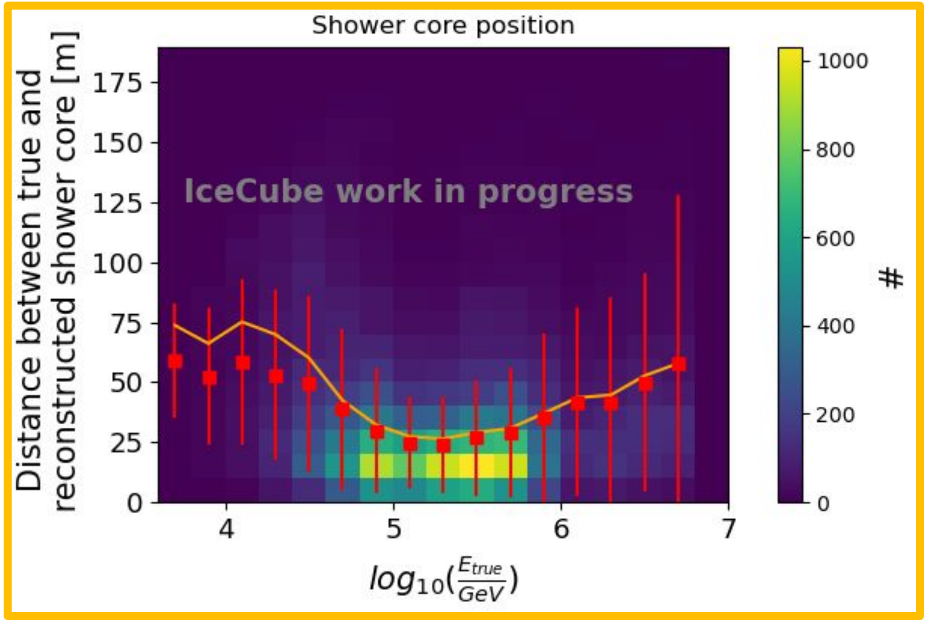
Output variables:
8 parameters
- E = energy primary
- D = distance between shower core and the telescope
- Sine and cosine angle on the x-y plane between x-axis and vector to shower core
- Zenith
- Sine and cosine of the azimuth angle
- Height of the shower maximum

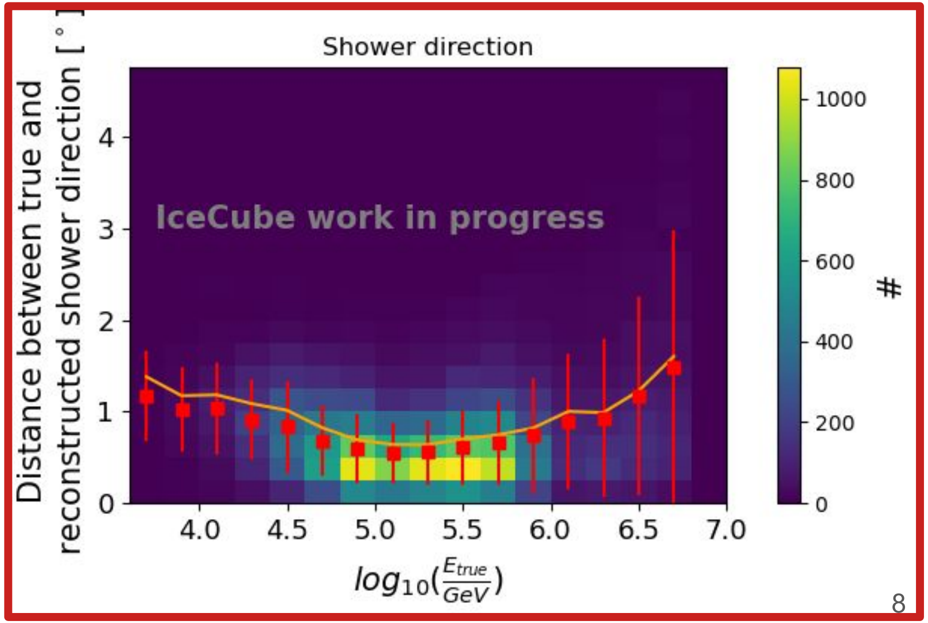
Shower core position

Shower direction

GNN results: shower core position and direction

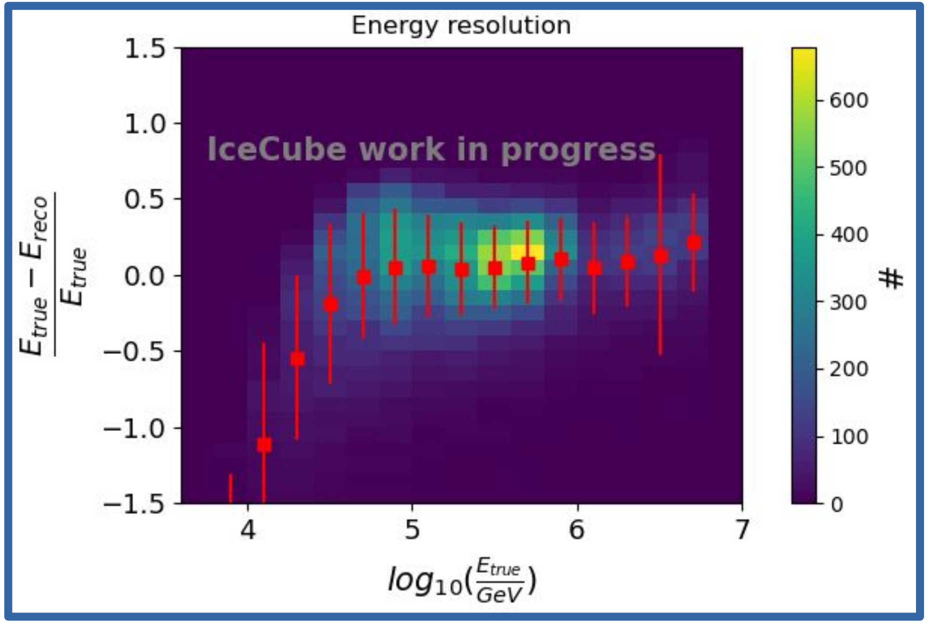
- There are clear deficits in the low and high end of the simulation
- The low energy end is determined by the trigger threshold
- In the middle energy region is working good
- The average distance between the telescope and the shower core increases with energy

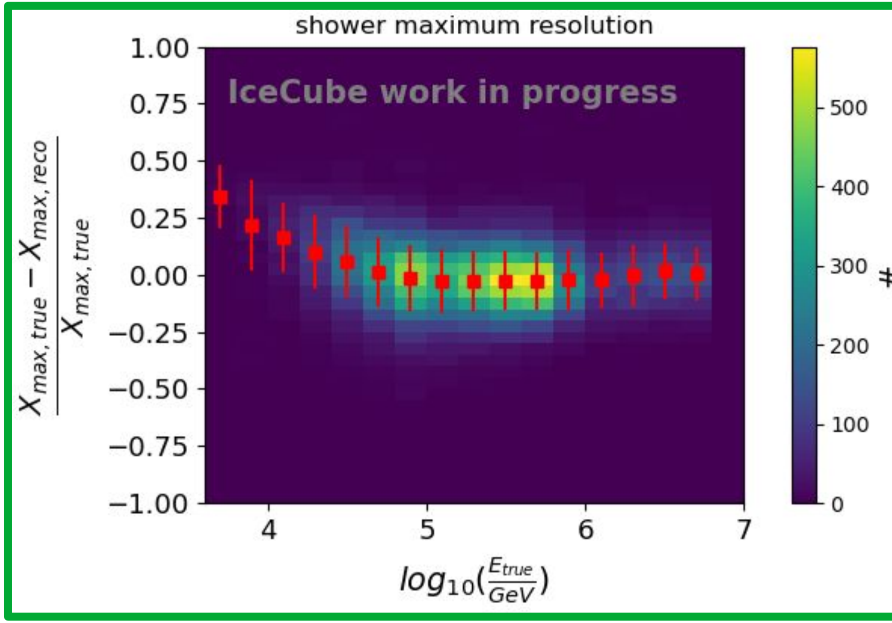




GNN results: energy and shower maximum

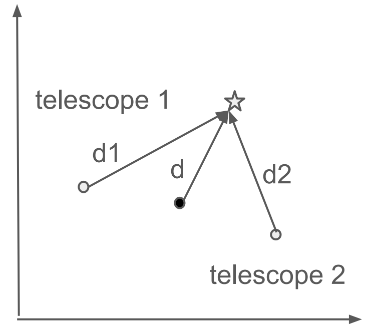
- Even though there are deficits in the low and high end of the simulation energy and shower maximum is working pretty good





Variations to the graph neural network reconstruction

- **1 Telescopes:** each camera picture is treated as a single event
  ~ 16k events
- **2 Telescopes:** pairs of telescopes are formed (roof - field), only if both see an event data is analysed
  ~ 2k events
- **1 or 2 Telescopes:** both types of events are taken
  ~ 15k events



Input parameter: Add telescope information to graphs

- Unique Id for each pair [[-1,1],[-0.66,0.66],[-0.33,0.33]] => **id**
- ignore that there are different telescopes => **none**

Output parameter:

Variation in the attempt to reconstruct the shower core positions

- Reconstruct distance between the center of the telescopes and shower core and angle between x axis and vector to the shower core => **dɑ**
- Reconstruct distances and angles for both telescopes => **d12ɑ** (4 values)

Additional parameter:

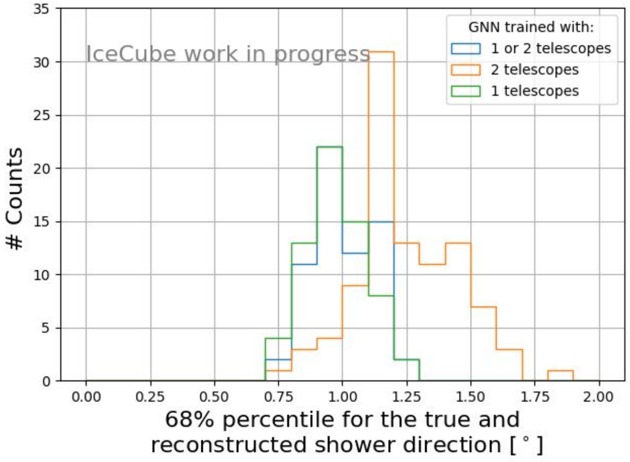
E = energy, **θφ**= zenith and azimuth, $\mathbf{X_{max}}$ = Shower Maximum

Bold abbreviations will be used for labeling in plots. Only letters which are present were used during training.

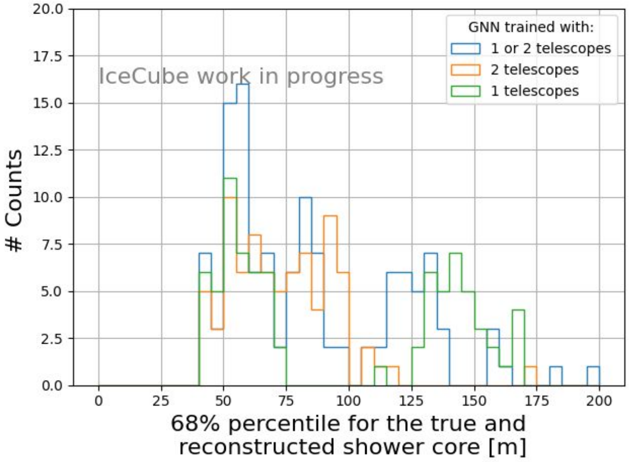
Comparing different types of training events:

- Each entry is a GNN with different input or output parameters
- 2 telescopes improve the average reconstruction of the shower core position and the reconstruction of the shower maximum
- For 2 telescopes the angular reconstruction gets worse, but it might be the statistics
- For shower direction and shower position the best GNNs are GNNs specialized on reconstructing the direction/position of the shower core, but …

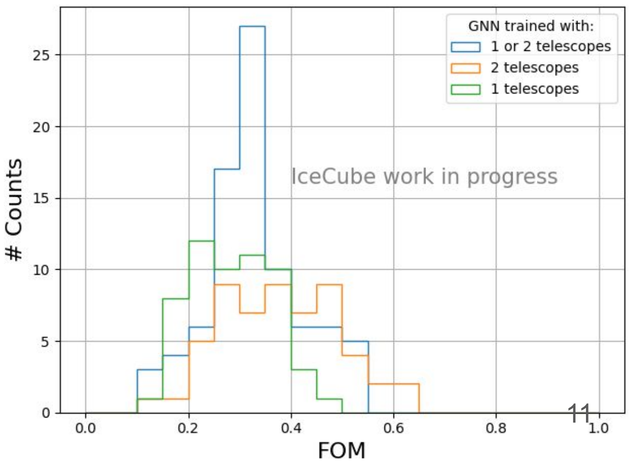
Shower direction



Shower position



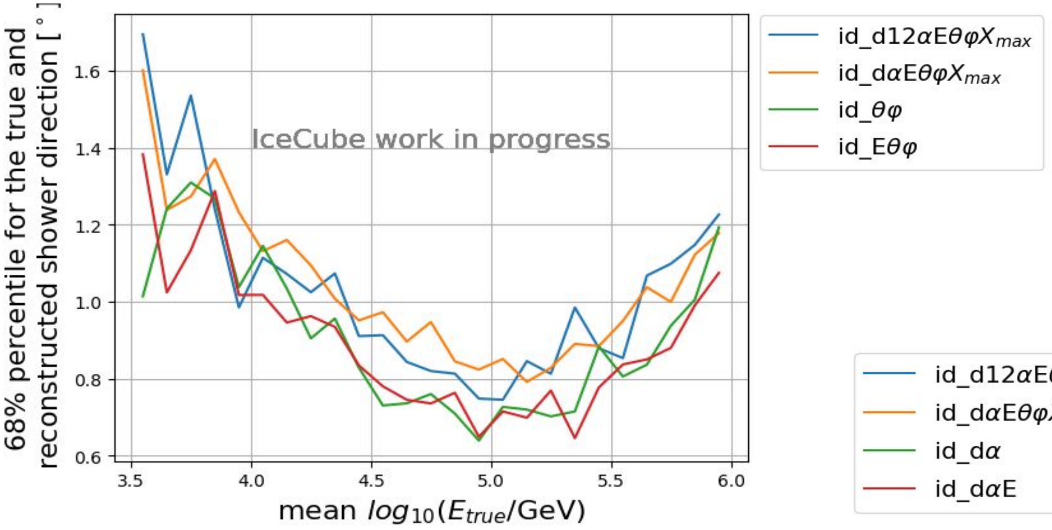
FOM



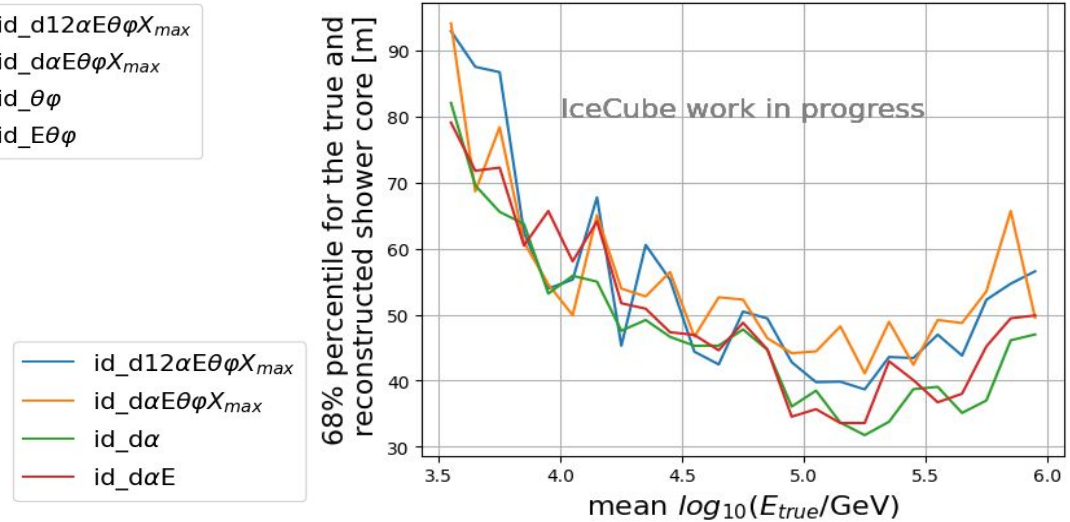
Energy dependent comparison of the GNNs:

- “Generalized” GNNs means GNN trained on all parameters possible, meaning the also give you energy and shower maximum
- Differences between specialized and more generalized GNNs are small over the whole energy range

Shower direction



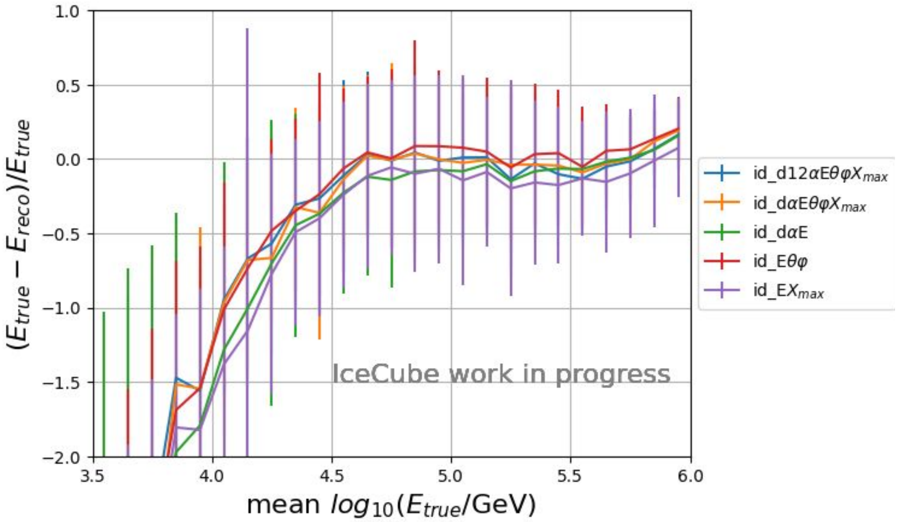
Shower position



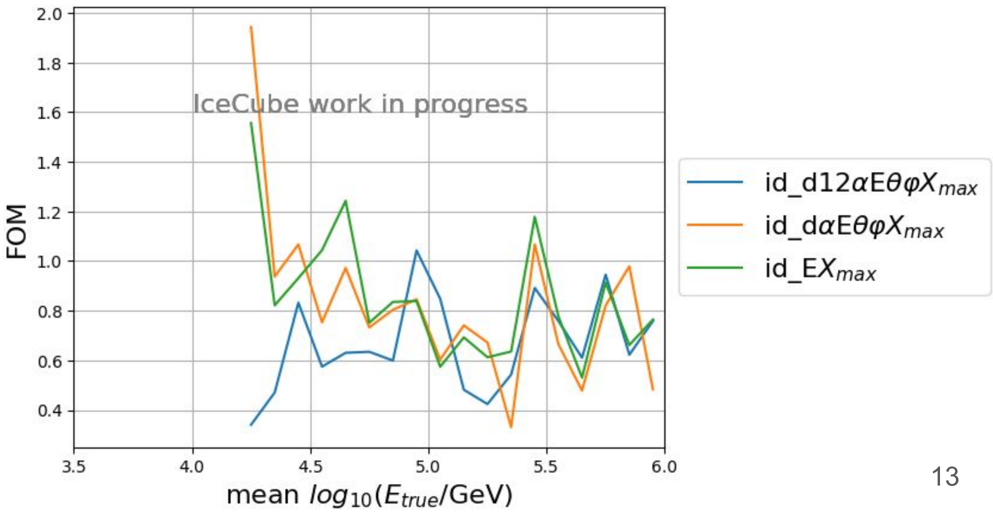
Energy dependent comparison of the GNNs:

- Energy reconstruction and reconstruction does not get better with specialization

Energie reconstruction



Reconstruction of the shower maximum



Conclusion:

- Different dataset, input and output parameters for the GNN were tested
- The best sets of output parameters dependent a lot on the goal of the analysis
- Generally trying to reconstruct all shower parameters at the same time is not always the best solution, but gives you one solution for all parameters with one training

QUESTIONS?

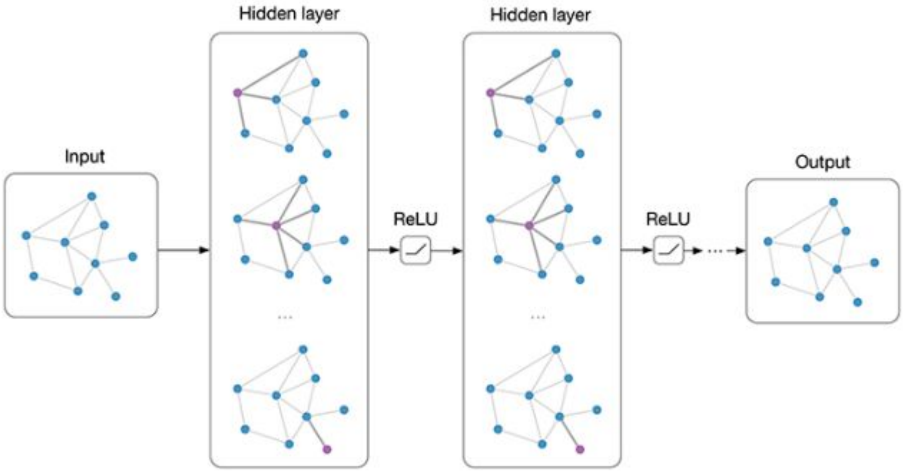
GNN model

```
##########################################################
# BUILD MODEL
##########################################################
X_in = Input(shape=(F,), name="X_in")
A_in = Input(shape=(None,), sparse=True, name="A_in")
E_in = Input(shape=(S,), name="E_in")
I_in = Input(shape=(), name="segment_ids_in", dtype=tf.int32)

X_1 = GatedGraphConv(32,4)([X_in, A_in, E_in])
X_2 = GatedGraphConv(32,4)([X_1, A_in, E_in])
##########################################################
# Pooling
##########################################################
X_3 = GlobalSumPool()([X_2, I_in])
##########################################################
# Dense
##########################################################
X_4 = Dense(512, activation="relu")(X_3)
X_5 = Dense(128, activation="relu")(X_4)
X_6 = Dense(32, activation="relu")(X_5)
output = Dense(n_out, activation="linear")(X_6)

model = Model(inputs=[X_in, A_in, E_in, I_in], outputs=output)

##########################################################
# Optimizer
##########################################################
optimizer = RMSprop()
##########################################################
# Compile
##########################################################
model.compile(optimizer=optimizer, loss="mse", metrics=['msle'])
model.summary()
```

```
##########################################################
# Parameters
##########################################################
learning_rate = 1e-2  # Learning rate
epochs = 200  # Number of training epochs
es_patience = 10  # Patience for early stopping
batch_size = 32  # Batch size

##########################################################
# Split data into batches
##########################################################
#xi = array of input parameters
#ai = adjacency matrix
#ei = edges between nodes
#yi = output variables
 dataset_tr = [Graph(x=xi, a=ai, e=ei, y=yi) for xi,ai,ei,yi in training_data]
dataset_va = [Graph(x=xi, a=ai, e=ei, y=yi) for xi,ai,ei,yi in validation_data]
dataset_te = [Graph(x=xi, a=ai, e=ei, y=yi) for xi,ai,ei,yi in test_data]

##########################################################
# Split data into batches
##########################################################
loader_tr  = DisjointLoader(dataset_tr, batch_size=batch_size, epochs=epochs)
loader_va = DisjointLoader(dataset_val, batch_size=batch_size)
loader_te = DisjointLoader(dataset_te, batch_size=batch_size)

##########################################################
# Fit Dataset
##########################################################
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', save_best_only=True)
history= model.fit(loader_tr.load(), steps_per_epoch=loader_tr.steps_per_epoch, \
          validation_data=loader_va.load() , validation_steps=loader_va.steps_per_epoch,\
           epochs=epochs, callbacks=[es, mc])
```

Introduction into graph neural network (gnn)

- Simple Graph Neural Network using Spektral package
- A Graph consists of nodes, each node has features and connections to other nodes
- For each event the number of nodes can be different
- The connection between nodes is defined in an adjacency matrix
- Hidden layers are matrix convolutions of the graphs and the adjacency matrix
- The normalization of the matrix differs depending on the chosen convolutional layer



https://tkipf.github.io/graph-convolutional-networks/