KØBENHAVNS UNIVERSITET:
BACHELORSTUDIET I FYSIK

BACHELOR THESIS

Optimizing Reconstruction and Error Estimation of IceCube Events Using Graph Neural Networks

Authors:

Jonas Vinther	KU- ID: dlk339	GitHub
Johann Bock Severin	KU- ID: msk377	GitHub
Jakob Hallundbæk Schauser	KU- ID: pwn274	GitHub
Christian Kragh Jespersen	KU- ID: htd809	GitHub

Advisor:

Troels Christian Petersen	Email: petersen@nbi.ku.dk

This thesis consist of **35** pages of main text and **21** pages of appendices.

The thesis was handed in the 16th of June, 2021.

Contents

Abstract

The IceCube Neutrino Observatory is an experiment located at The South Pole, which aims to detect neutrinos and other particles in the ice sheet. This thesis presents a graph learning approach to low-energy muon/neutrino reconstruction and particle classification. The algorithms developed by the group attempt to improve upon the currently used reconstruction algorithm, Retro. By employing the versatile framework of graph neural networks (GNNs), we improve reconstruction speed by a factor of $O(10^5)$, directional accuracy by a factor of 15% to 20% for different performance metrics, and energy accuracy by a factor of 13%, while simultaneously improving uncertainty estimation. If the models are gathered in an ensemble, the overall performance was improved. This is especially significant at lower energies, but valid throughout the $1 - 10^4$ GeV energy range. When applying these models to the task of classifying whether a muon has stopped in the ice, we achieve an AUC of 0.93. When classifying particle types, the model cleanly classifies muons, but struggles with distinguishing between neutrino types.

1 Introduction

Of all the particles in the standard model, neutrinos are arguably the most puzzling. They have mysteriously small masses, exist in a superposition between different flavour-states and notoriously unwilling to interact with normal matter.

The IceCube Neutrino Observatory is currently the most promising experiment in increasing our understanding of neutrinos. This project focuses on a Machine Learning approach to reconstruction of low-energy particles in IceCube.

Although our main interest neutrinos are very sparse in IceCube, making up only one per million seen particle, the rest being almost exclusively atmospheric muons. This makes muons good candidates for reconstruction algorithm calibration.

This thesis begins by outlining the relevant physics before sketching out how the IceCube Observatory works. The theory behind graphs and Graph Neural Networks (GNNs) are then introduced before explaining our implementation in relation to the IceCube data. An overview of the four designed models are given along with the motivation behind their design. The collective reconstructive power of the models and their ensemble is evaluated and discussed for different types of regression and classification for muons and neutrinos. The thesis concludes with the findings of the project, several learning points, and suggestions for further work.

In this project the work is divided such that each author presents a firm understanding of the underlying theory, and each contributes with a model architecture, overall resulting in an equal contribution by every group member.

2 Particle Physics: Theory of Muons and Neutrinos

Understanding the building blocks of our universe is one of the oldest, most fundamental questions in physics. In our time, the field of high-energy physics is responsible for cataloging these building blocks, now known as particles. While everything around us is created from these fundamental particles, observing them has proven surprisingly tricky.

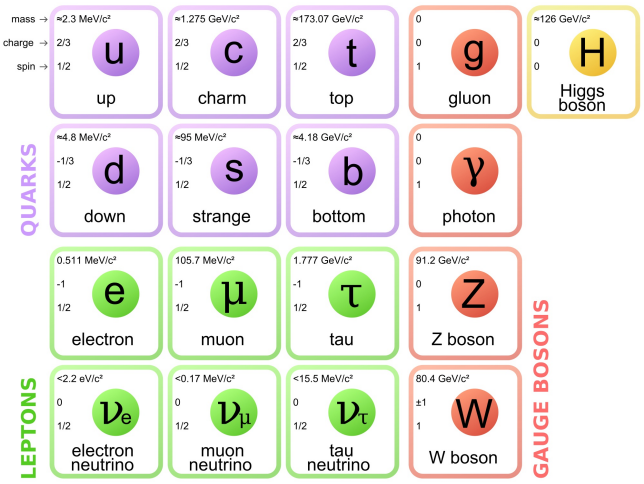

Figure 1: The Standard Model of particle physics (Image source: Public Domain)

This section will provide a general overview of the fundamental particles focusing on the parts most relevant for the project.

2.1 The Standard Model

Currently, the best model for describing the interplay between particles and forces is the standard model shown in Fig. 1.

In the standard model, particles are split into bosons, the carriers of the known forces, and fermions, the building blocks of matter. Because of the inherent spin of the two, they follow the Bose-Einstein and Fermi-Dirac statistics respectively.

Three of the four fundamental forces are mediated by a corresponding boson. The probability of any given particle interaction occurring is proportional to the square of a boson-specific and energy dependent coupling constant. The strong force is mediated by the gluon with the highest coupling constant of approximately 1.[1] The weak force is mediated by the $W^{\pm}$ for charged interaction and Z-bosons for neutral ones - both with a coupling of about 10^{-6}. The electromagnetic force is mediated by the photon and has a coupling constant equal to the fine-structure constant $\approx 1/137$. Gravity has no apparent force-carrier in our current system, but has a connection to the Higgs boson. The coupling constant of gravity is somewhere around 10^{-37} compared to the strong force, rendering it negligible when considering particle interactions.

The fermions, which make up most of the stan-

[1] making strong interactions the most likely

dard model, have even more variety. Fermions are further classified as either quarks or leptons. Due to their interactions with the strong force, a phenomenon known as colour-confinement means quarks are always observed in groups of either two (mesons) or three (baryons). In this project the main focus is on the leptons, which are not affected by the strong force. Therefore, a large experiment like IceCube is required to see a significant amount of reactions.

Furthermore, the leptons are divided into columns by their ***flavour***: *electron*, *muon* or *tau*. These flavours consist of a charged particle and its corresponding lighter, neutral neutrino. Neutrinos are commonly denoted by ν_α, with α indicating the neutrino type [1].

2.2 Neutrinos and Oscillations

Neutrinos are uncharged leptons, interacting purely gravitationally or via the weak force - the first of which is notoriously weak, the latter very short ranged. They exist in overabundance but are so weakly interacting that they are almost undetectable.

In the 1960s, Ray Davis started a radiochemical experiment in a mine in South Dakota to measure the flux of solar neutrinos. After multiple decades of observation, he found that the solar electron neutrino flux was about a factor of three lower than expected. The deficit was a mystery for many years, but was ultimately found to be caused by flavour transitions which led to the 2015 Nobel prize in Physics [2].

Neutrinos of the three leptonic flavors have slightly different masses. As a consequence, the mass states propagate through space at marginally different rates. Since each particle exists in a quantum mechanical superposition between the mass states, a mixing between flavour states ensues for the individual travelling particle. Due to the connection between mass and flavour and the periodic nature of the probability change on a macroscopic length scale, this amounts to an oscillation between different flavour states. The mixing between the eigenstates is parametrized by the Pontecorvo–Maki –Nakagawa–Sakata (PMNS) matrix:

$$\begin{bmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{bmatrix} = \begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix}$$

In this representation, the flavour eigenstates are in a superposition of the mass eigenstates (and vice versa) and can therefore be written as a linear combination using the unitary mixing matrices:

$$|\nu_\alpha\rangle = \sum_{i=1,2,3}^{3} U_{\alpha i} |\nu_i\rangle \Leftrightarrow |\nu_i\rangle = \sum_{\alpha=e,\mu,\tau}^{3} U^*_{i\alpha} |\nu_\alpha\rangle \quad (1)$$

Here U is a matrix element from the unitary PMNS-matrix with i and α the different mass and flavour states respectively [3].

Due to the Dirac equation, the mass states in vacuum exist as plane-wave solutions of the form:

$$|\nu_i(t, \boldsymbol{r})\rangle = e^{-i(E_i t - \boldsymbol{p}_i \boldsymbol{r})} |\nu_i(0, \boldsymbol{0})\rangle$$

For ultra relativistic neutrinos the phase can be approximated by:[2]

$$E_i t - \boldsymbol{p}_i \boldsymbol{r} \simeq m_i^2 L/(2E_i) \qquad \text{(natural units)}$$

Where L is the traversed distance. Taking the second part of Eq. 1 and the plane-wave solution to define the probability of transition:

$$P(\nu_\alpha \to \nu_\beta) = \left|\langle v_\beta \mid v_\alpha(L)\rangle\right|^2 = \left|\sum_i U_{\alpha i} U^*_{\beta i} e^{-i\frac{m_i^2 L}{2E_\alpha}}\right|^2$$
$$\propto \sum_{i>j} A_{ij} sin^2\left(\frac{\Delta m_{ij}^2 L}{4E_\alpha}\right) + B_{ij} sin\left(\frac{\Delta m_{ij}^2 L}{2E_\alpha}\right)$$

Where A and B consist of the matrix elements. This ends up showing that, given a difference in mass, the complex phase introduces a length and energy-dependent transition-probability [4].

Matter introduces an effect in the wave function, thus creating a space-dependency of the flavour transitions. This gives neutrino detectors an ability to observe collections of mass, an example of which is the Earth's core, clearly visible in Fig. 2.

2.3 Muons

Every day, 275 million cosmic rays are detected by IceCube. Only about one in a million of these is an atmospheric neutrino. For this reason, the more readily interacting muons are of great interest. One of the advantages of muons are that they have a sufficiently short penetration depth in the

[2] This holds for all observed neutrinos since their masses are at least a factor 10^6 less than their energy

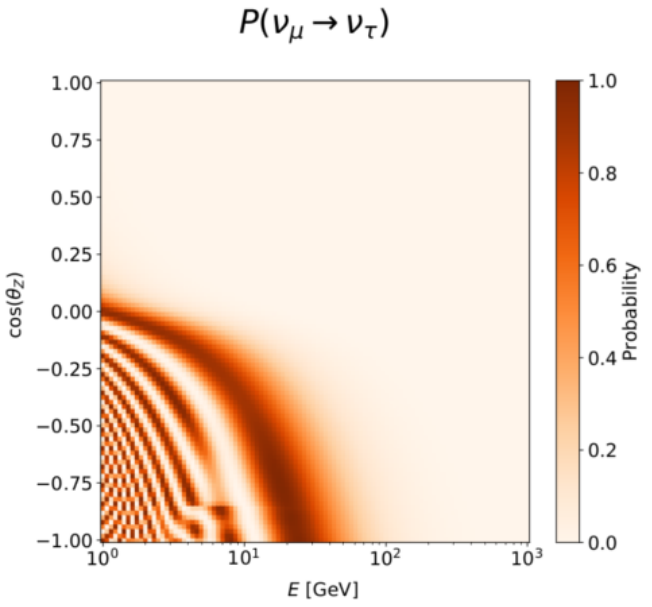

Figure 2: Neutrino oscillation plot. The Earth's core is visible as a perturbation at about 8 GeV for $\cos(\theta_z) \approx -0.8$ (Image source [5])

ice to be completely stopped within the experiment. This gives us a good calibration tool for regressing energy, by acquiring a sample of stopped muons where we know the kinetic energy displaced in the detector [6].

The rate at which a muon deposits energy in the ice is subject to many processes. A simplified overview is presented in Fig. 3. Since the differential cross-section is momentum-dependent, muons with energy > 10 TeV deposit energy at a high rate and are very rare. Muons with low energies < 50 GeV are quickly stopped by the Coulomb interaction in the ice. Thus most muons in IceCube are seen in the approximate interval $50\ \text{GeV} < E_\mu < 10\ \text{TeV}$.

2.4 Cosmic Rays

The Earth is constantly bombarded by a number of particles from the universe. As these particles interact with the atmosphere, they decay into other particles. Most interestingly for us, these collisions can result in both muons and neutrinos being created. The most common example is the pion to muon to electron decay which creates neutrinos in flavour-proportions of $(\nu_e : \nu_\mu : \nu_\tau) = (1 : 2 : 0)$. (Illustrated in the Feynman diagram in Fig. 5.)

Since almost all particles observed in IceCube come from the atmosphere (see Fig. 4), the muons come primarily from above the horizon of the experi-

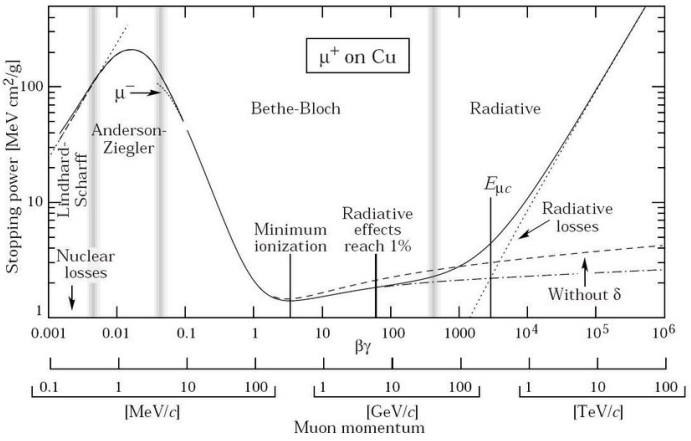

Figure 3: The rate at which muons radiate energy as a function of muon energy. While the figure is based on stopping power in Copper, the relevant effects and momentum-dependency are shown. (Image source: [7])

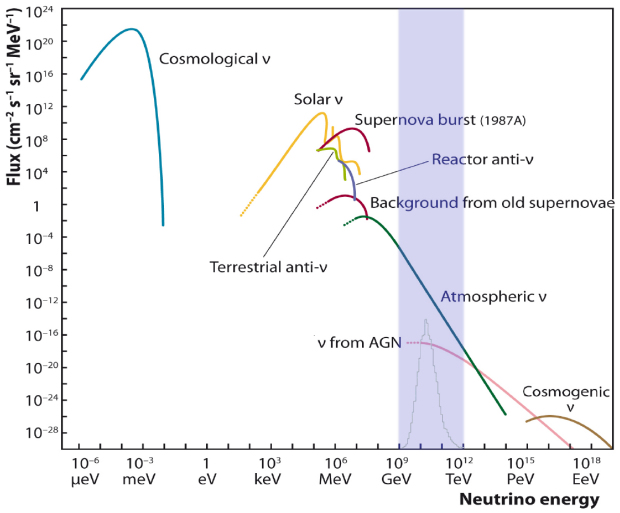

Figure 4: Theoretical flux as a function of energy with our approximate energies of interest marked in blue. (Image source: [8])

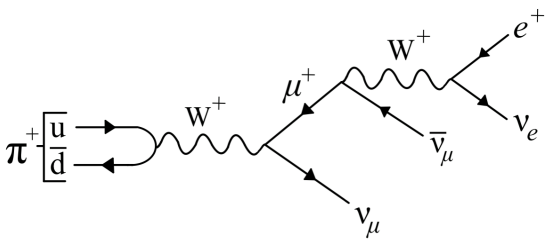

Figure 5: A Feynman diagram of the charged pion decay. The initial pion could also be negatively charged, changing the sign of all charges for the subsequent particles.

ment, partly due to their short lifetime and penetration depth, whereas neutrinos due to their faint interaction are seen from every angle. As neutrino oscillation is dependent on the length and time of travel, we see different neutrino flavours at different zenith angles and energies, as shown in Fig. 2. Keep in mind, the distribution of neutrinos as a function of zenith is reminiscent of $sin(\theta_z)$ since the area of the latitudinal bands goes as $dA = 2\pi r^2 sin(\theta)d\theta$. Consequently, IceCube sees less neutrinos through the center of the Earth compared to the horizontal.

2.5 Charged/Neutral Current Decays

Neutrinos have two distinct weak interaction types, differentiated by the mediating boson type: charged current (via the W-bosons) and neutral current (via the Z-boson). The interactions are named due to the W-bosons allowing interaction with an electric charge, while the Z only affects the spin and momentum of the interacting particles.

As the neutral current always decays into a Z and a neutrino (with the energy split about evenly between them), only half of the incoming energy is visible to the detector due to our current inability to consistently observe the neutrinos [9]. Neutral current corresponds to about 10-15% of the collected data. It is common to separate the two types of events, training only on the easier charged current events, however this separation was not made in this thesis. The Feynman diagrams for neutral and charged current decays can be seen in Fig. 6.

2.6 Cherenkov Radiation

In the IceCube experiment the particles are not detected directly, instead the particles are instead found by looking for emitted Cherenkov light in the ice. When a charged particle moves through a medium, it emits radiation which excites the molecules around it. Upon returning to their ground state, the molecules re-emit photons. For superluminal particles with $v > c_{medium}$ the particle exceeds the speed of the photon waves (which propagate with c_{medium}). [3] Similar to the sonic boom when a jet travels faster than the speed of sound, the light from the excited molecules will create a delayed wave trailing

[3] This does not contradict special relativity since $v < c_{vacuum}$

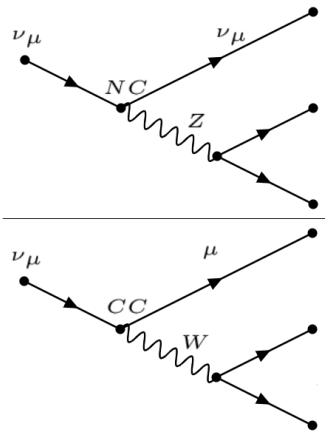

Figure 6: A schematic of the difference between a neutral and charged current decay. In the charged case, the primary lepton is visible in the detector, while for the neutral case, the primary neutrino is generally not visible. This amounts to the visible energy in the NC case being lower by about a factor 2.

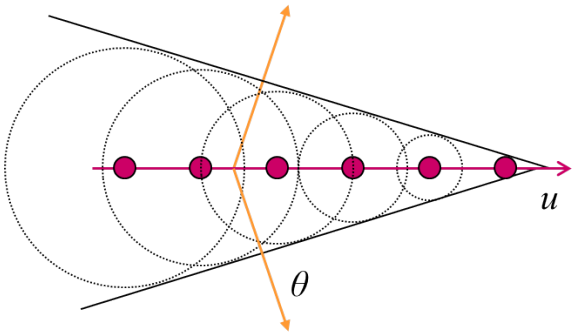

Figure 7: An illustration of the wave front created by a charged, superluminal particle. (Image source: [10])

the charged particle. This phenomena is known as Cherenkov radiation and as the emitted light usually lies around the ultraviolet band, it is seen as a bright blue light [10]. An illustration is shown in Fig. 7.

3 The IceCube Experiment

This section provides an overview of the IceCube detector, how measurements are made, how simulations are carried out, and what considerations can be made regarding the forthcoming observations.

3.1 The Detector

While most detectors are built from the ground up, filled with material for particles to interact with, IceCube uses multiple gigatons of ice in the pre-

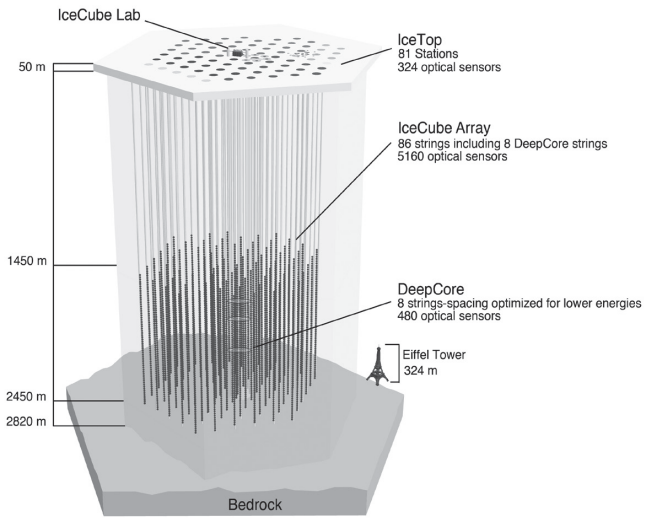

Figure 8: A schematic of the layout of the IceCube Neutrino Observatory (Image source: [15])

existing glacier on The South Pole. As the name suggests, IceCube is shaped as a hexagonal cylinder. The neutrino observatory, which is the largest in the world, took about a decade to build, finishing in 2010. The main detector array consists of 5160 sensors split between 86 strings. The strings are about 2.5 km long, spaced about 125 meters apart with sensors every 7 meters [11].

On top of the ice, there are 162 tanks, each containing two sensors. This surface-based sensor-array is called *IceTop* and is used to filter atmospheric particles from astrophysical events. This method is named *Veto* [12].

The physical expanse of IceCube was chosen for the purpose of detecting neutrinos and stopped muons. This means it should be large enough for some muons to expend all their energy, while also not so large as to have too many local events [13].

As IceCube was being built, multiple irregularities showed themselves. First, a dust layer lies at a depth of about 2000 meters, as seen in Fig. 10. Secondly, to lower the DOM-filled strings into the ice, cylindrical shafts were dug. When the now sensor-populated holes were refilled with water, bubbles were created. As the water froze, the bubbles were suspended in an unpredictable way creating non-trivial scatterings for light along the shafts [14].

3.2 DOMs

The main sensors in IceCube are the DOMs (Digital Optical Modules). These spherical, glass-covered

Figure 9: A DOM. (Image source: [17])

detectors take advantage of the light from Cherenkov radiation to detect superluminal particles (Fig. 9). Once triggered, a DOM will record the hit for about $6.4\mu s$ or $0.43\mu s$ depending on the internal digitizer used, capturing the waveform with an onboard photomultiplier tube [16]. While the current generation of DOMs are built with sensors aimed downwards to mostly catch neutrinos, the generality of the design has allowed IceCube to also be used for detection of other particles (muon-detections being the most prevalent).

3.3 Upgrade and DeepCore

DOMs have been in IceCube since it was first opened for operation, but the observatory has seen major updates to its structure. First, the hexagonal grid of sensors in IceCube proved to have a major design flaw. Some particles can be angled in a way that they travel down the rows of DOMs while barely registering on any (see purple arrow in Fig. 10). Thus the first upgrade, *DeepCore*, was conceived [18]. DeepCore expanded on the central part of the experiment with 480 DOMs, raising the density of sensors and breaking the discrete symmetry. As seen in Fig. 10, the DeepCore DOMs are mainly placed below the dust layer with a small portion of sensors placed above serving as a veto module to filter out atmospheric particles. The second upgrade, simply called *Upgrade*, is still in progress. Upgrade will add 750 new sensors, *D-Eggs* and *MDOMs*, with more advanced designs and optical detectors in multiple directions [19].

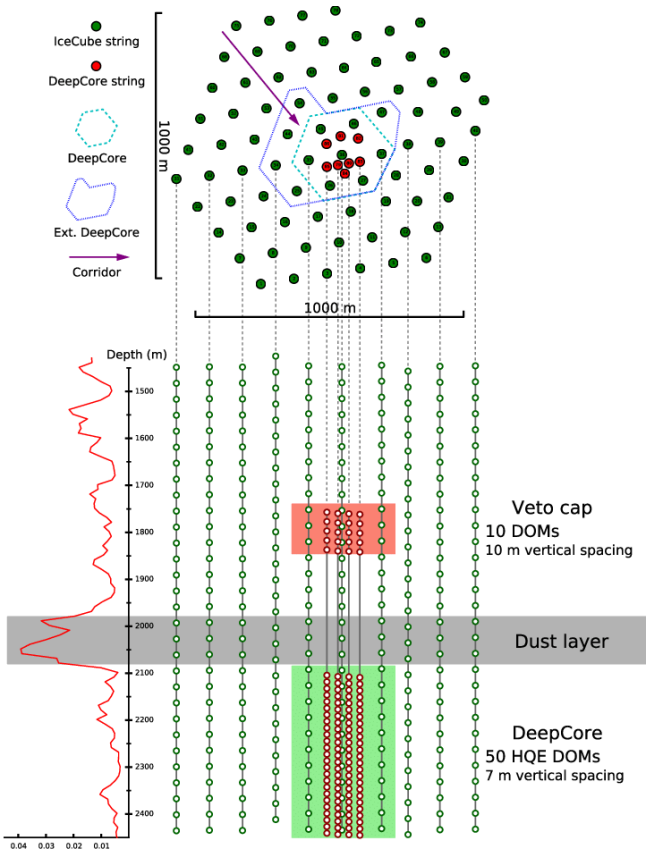

Figure 10: DeepCore. (Image source: [20])

3.4 Retro Reconstruction

The main regression algorithm employed at IceCube is called *Retro Reconstruction* and is based on a Maximum Likelihood Estimator [21]. To speed up calculations, the algorithm uses pre-generated lookup-tables of light propagation throughout the detector. Retro is reliable and precise but is comparatively slow, on average taking 5-40 seconds per reconstruction, but upwards of 100 seconds for certain events. In this thesis it will be used as a baseline for comparing results. Since the average daily flux of triggered events in IceCube is on the order of 10^8, faster reconstruction is paramount.

3.5 MCMC Simulations

One of the largest bottlenecks in the Markov Chain Monte Carlo (MCMC) simulation of muons is the propagation through the atmosphere and ice, long before a particle reaches the detector. *MuonGun* circumvents this problem by utilising data from previous (more advanced) simulations and simply tries to match the parameter-distributions. As the particles are 'created' at the very edge of the detector, *MuonGun* has the ability to quickly create large training samples of atmospheric muons at a fraction of the required computing power [22].

Moving up in complexity, *Corsika* is a relatively old algorithm, using MC methods to simulate air showers of many different types. Through eight major iterations, it has stayed a staple in the atmospheric HEP community since its creation in 1998. As it looks at many of the likely methods of decay, it is accurate for most detector arrays. *Corsika* works especially well at higher energies [23].

Genie is the final source of MCMC data with direct relevance to our project. Genie is a state of the art neutrino simulation collaboration, spanning all facets of the problem [24]. From the theoretical perspective to analysing newly collected data, the Genie-group works to create one of the most physically accurate data generators. Genie provides the main data and truth values for the neutrino work of the OscNext group.

3.6 Cleaning

The DOMs of IceCube are highly sensitive and subject to frequent noise of the order ≈ 1 kHz, but as they operate on nanosecond timescales, they are still quite reliable. Nonetheless, due to the noise, strict requirements have to be met before a DOM is triggered. This is determined by the hard local coincidence (HLC) condition, where a minimum of DOMs have to trigger within a short time span ($\pm$1000 ns). IceCube uses the "SMT8" trigger, which is met when 8 DOMs satisfy the HLC condition in a 5 µs window, and the pulses from DOMs will be logged from -10 to 10 µs relative to the trigger [18].

Since a lot of noise is generated from the DOMs themselves, not all pulses within the logged time are associated with the actual event, nor are all logged events actual events. To clean the pulses in a single event, IceCube uses the Seeded Radius-Time (SRT) cleaning. Here pulses are stored if they satisfy that another pulse is found within $R = 150$ m and $|\delta t| = 1000$ ns (equal to a velocity of $0.5c$). This is applied iteratively, starting from the pulses triggering the "SMT8" condition, and adding all the events that satisfy the RT criterion [25]. As each event consist of multiple triggered DOMs, many of which are due to noise, 7 levels of post-cleaning exist, where a higher cleaning level corresponds to

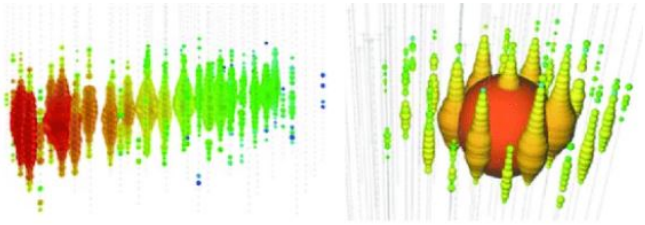

Figure 11: The two most common event types for neutrinos. A tracklike event is shown on the left, and a cascade event on the right. (Figure source: [27])

a less noisy sample.

3.7 Track-like and Cascade Events

When regressing the trajectory of a neutrino, it should be noted that the different neutrino types react quite differently with the ice. For tau-neutrinos, ν_τ, the decay will result in a tauon that quickly decays. The electron from ν_e also quickly interacts with the ice. Thus the two types of decay are very localized in the detector. These are called cascade events and are generally hard to regress, since the shower does not leave a clear directional asymmetry. However, the muons originating from muon neutrino decays travel significantly further, due to their higher mass, and leave a track of light in the detector. These events are called track-like events and are easier to regress [26].

A representation of the difference can be seen in Fig. 11.

4 Graph Neural Networks

This section contains the theory behind using neural networks in prediction tasks, an overview of the general learning procedure, and specifically, learning on mathematical graphs.

4.1 Why Neural Networks?

Before we dive into understanding neural networks (NN), one might ask: Why bother with these strange black box methods at all? Our choice to study event reconstruction through machine learning on graphs was influenced by both the strengths and weaknesses of NNs in contrast to algorithms - the primary strength being the *universality* of NNs. Two theorems, one regarding network width, the other network depth, taken together as the *Universal Approximation Theorem* (UAT), provide the basis for this. Together these state that any function can be approximated to an arbitrary precision by an NN if allowed to extend to arbitrary size in either depth or width. Thus, any continuous function can **in theory** be approximated to arbitrary precision, but the theorem does not state how to find said NN.

However, the true strength of NNs is shown when approximating a function that does not yet exist, which can lead to new discoveries, breakthroughs in performance, or a pseudo-proof that current algorithms are working at the information limit. While this seems wonderful, one must realize that this is by no means guaranteed to be achieved by any specific network.

The NNs also have a few drawbacks compared to classical statistical models. First of all, large datasets are required in order to achieve convergence. Secondly, NNs are trained stochastically and two networks are therefore not guaranteed to be the same even given the same initial starting point, data and parameters for training. Furthermore, the complex nature of NNs makes them very hard to interpret. Each of these problems constitute an area of research in themselves.

4.2 Introduction to Neural Networks

Artificial NNs are made to simulate the brain's neural structure, with each neuron sending signals along to the next depending on its inputs and inherent hidden states. In a simple NN, we consider perceptrons as seen in Fig. 12. The network is then built from consecutive stacking of several connected *hidden layers* each being a vector of perceptrons $\boldsymbol{h}$. Only allowing connections in one direction renders a feed-forward network. A single hidden state in $\boldsymbol{h}$ is indexed as h_i. h_i is calculated summing each value from the previous layer separately scaled by connection weights w_{ji}, and adding a bias b_i:

$$h_i = \sum_j w_{ji} x_j + b_i \tag{2}$$

Now going from one layer to the next will be equal to the matrix multiplication with the weight matrix $\boldsymbol{W}$:

$$\boldsymbol{h^{t+1}} = \boldsymbol{W^t h^t} + \boldsymbol{b^t} \tag{3}$$

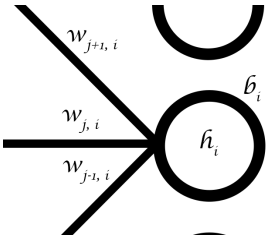

Figure 12: Illustration of a single perceptron.

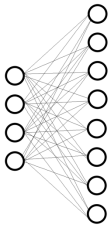

Figure 13: 4 perceptrons being connected to 8 in a feed-forward NN

where the superscript defines the layer number. The ensemble of perceptrons in a layer with their respective weights are illustrated in Fig. 13.

4.2.1 Activation Layers

While a collection of the basic linear layers described this far is quite versatile, further flexibility and non-linear behaviour should be added by the use of activation layers. An activation layer is made up of a single differentiable function used on a hidden state. The activation functions with relevance to this project are shown in Fig. 14 and given as:

- **RELU**: Casts negative values to zero:

$$\mathrm{RELU}(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{4}$$

- **Leaky RELU**: Has a small gradient ($\alpha > 0$) for $x \leq 0$:

$$\mathrm{lRELU}(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases} \tag{5}$$

- **GELU**: allows nodes to relay slightly negative numbers, while mimicking RELU.[4]

$$GELU(x) = x \cdot \frac{1}{2}\left[1 + erf\left(x/\sqrt{2}\right)\right]$$

- **Hyperbolic tangent**: Used to smoothly limit values $(-1, 1)$.

- **Sigmoid**: used to smoothly limit values in $(0, 1)$, which is especially useful for classification tasks.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

[4] As the Gaussian error function *erf* is computationally expensive most implementations use an approximated version [28].

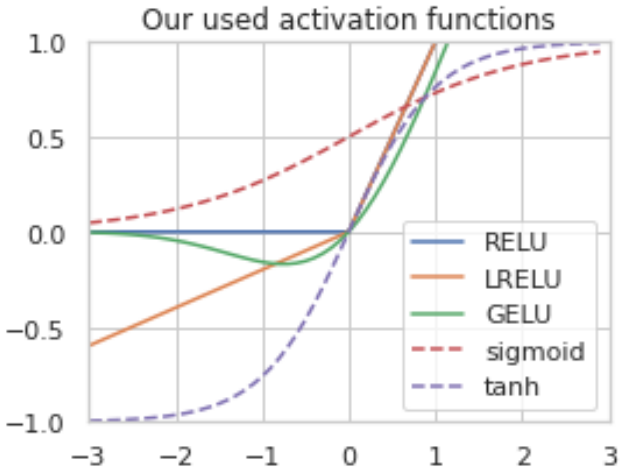

Figure 14: Overview over activation functions.

- Lastly, **SoftMax** was also used, which is different from the rest in that it does not take a single input but a collection of inputs, and normalizes them according to the Boltzmann Distribution of negative input:

$$SoftMax(x_i, \{x_n\}) = \frac{e^{x_i}}{\sum_n^N e^{x_n}} \tag{7}$$

4.3 Learning

The goal of training an NN is to fit values for the weight matrix $\boldsymbol{W}$ and the biases $\boldsymbol{b}$ throughout the network so the output $\boldsymbol{y_{reco}}$ best approximates some target value $\boldsymbol{y_{true}}$. For this reason $\boldsymbol{W}$ and $\boldsymbol{b}$ are referred to as the learnable parameters of the network, $\boldsymbol{\theta}$.

Describing the NN as a function, we represent it in the following way:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{y_{reco}} \tag{8}$$

For the training we now define a loss function, $\mathcal{L}(\boldsymbol{y_{true}}, \boldsymbol{y_{reco}})$, which is defined such that a lower value of $\mathcal{L}$ is obtained if the prediction of y_{reco} is closer to y_{truth}. One such loss function can be a sum of squared errors, which would be calculated as: $\mathcal{L} = (\boldsymbol{y_{true}} - \boldsymbol{y_{reco}})^T(\boldsymbol{y_{true}} - \boldsymbol{y_{reco}})$ and would be 0 for a perfect prediction.

The goal is now to fit $\boldsymbol{\theta}$ such that $\mathcal{L}$ is minimized over all data points in the training sample:

$$\tilde{\boldsymbol{\theta}} = \mathrm{argmin}_{\boldsymbol{\theta}} \sum_{data} \mathcal{L}(f(\boldsymbol{x}, \boldsymbol{\theta}), \boldsymbol{y}) \tag{9}$$

4.3.1 Backpropagation

As all of our implemented algorithms are of the feed-forward kind, they learn by utilizing backpropagation. This term encapsulates computing the gradients of the weights with respect to a specific loss function $\mathcal{L}(y_{true}, y_{reco})$ such that they can be used for gradient descent.

Considering the gradients only, the path through a model can be seen as simply consisting of repeated and alternating matrix multiplication and non-linear activation functions. For the weight matrix of layer i W^i and activation function ϕ a prediction $y_{reco} = f(x)$ is calculated as:

$$f(x) = \phi^N \left(W^N \phi^{N-1} \left(W^{N-1} \cdots \phi^1 \left(W^1 x \right) \cdots \right) \right)$$

The loss is then calculated as $\mathcal{L}(y, f(x))$. As all layers are nested, the derivative of the individual layer weights W^ns can be calculated by the chain rule.

This formulation grants us the ability to recursively find the component of the gradient. Thus, given an input/output pair, starting from the final layer, backpropagation is used to calculate the gradient terms for every layer. The weight of each layer is then updated by the *optimizer* using the gradients and a factor called the learning rate, which will be further explained in the next section.

4.3.2 Optimizing

As previously mentioned, minimizing the loss function is key. But since the loss landscape is highly non-convex, finding an optimal spot is non-trivial as saddle points and local extrema can be confused for global minima. The work is done via stochastic gradient descent and the algorithm used is called an *optimizer*.

Our choice of optimizer is called Adam. Adam has an adaptive step size and is therefore (along with RMSprop and AdaGrad) one of the most versatile algorithms at finding minima [29]. Adam has a number of hyper-parameters that can change how the gradient descent plays out, *learning rate* being the most important. Learning rate is tunable for most optimizers and corresponds to the 'step size' of the algorithm for each application of the gradients. Typical values lie on the order of 0.001; higher learning rates can result in quicker descent while lower values can help find minima that might otherwise be skipped. As this parameter is vital to the speed and success of the optimization, *learning rate-finders* are used. These work by running a number of epochs at different learning rates, keeping track of the change in loss, and ultimately finding the sweet spot between diverging and stagnating.

Even though Adam has an adaptive step size a learning rate-finder can help find an initial value, some literature suggests that changing the learning rate during training can improve the convergence of the optimizer [30]. This is called *scheduling* and is most commonly an exponential decay or a linear rise followed by a descent called one-cycle.

As Adam nears a minimum and internally lowers its step size, the change in loss might come from the model learning the individual quirks of the specific dataset instead of the general features of the data. This problem is called *overfitting* and is very common. Here *early-stopping* can be applied, somewhat negating the problem by cross referencing the loss with a calculated loss for a set of "validation data" the model has not been trained on. If the validation loss has not seen an improvement for a set number of epochs, the training is stopped.

4.3.3 Regularization and Dropout

To combat overfitting of a model one could also add regularization to a model. During regularization, a term dependent on the size of the weights is added to the loss function. In the simplest cases, this is done by the L_1 penalty: $\lambda \sum_i |w_i|$, where λ is a hyper-parameter. If the weights have no effect on the primary loss function they will have an optimal value of 0, and we can thus limit the practical number of weights used throughout the network. Another commonly used regularizing term is the L_2-penalty: $\sum_i w_i^2$. One could interpret this penalty term as a prior of a Gaussian weight distribution in a Bayesian interpretation [31].

Furthermore, one could introduce randomness into the model by using dropout layers. Dropout is inspired from bagging, where multiple smaller models are trained on subsets of the data, avoiding overfitting on the entire dataset. Dropout approximates this process by generating a random binary mask

for each layer, only allowing some hidden states to pass to the next layer. Effectively, this creates a subset of the model and performs a back-propagation which will train only this specific subset. In the next iteration the dropout layers will be randomly generated again and another subset will be improved. Thus, when looping over the data a different subset of the model will see the same data each time, limiting the ability for the model to overfit to the training data [31].

4.3.4 Batch Normalization

In deep NNs, multiple layers are computed sequentially. The values of the hidden states can sometimes take on values in vastly different domains, which, especially in later layers, can increase the training time needed to reach a loss minimum. To reduce this effect, *Batch Normalization* is introduced as an attempt to limit the way that earlier layers impact the general order of magnitude of the weights and biases for later layers. The idea of batch normalization is to normalize the input between each layer so that the input to the next layer will have mean, $\mu = 0$ and variance, $\sigma^2 = 1$ [32].

Batch normalization works differently when training than when predicting. During training, every batch of inputs is normalized according to the mean and variance by:

$$\hat{x} = \frac{x - \text{mean}(x)}{\sqrt{\text{Var}(x) + \epsilon}} \tag{10}$$

where ϵ is a small numerical fudge factor ensuring stability. Furthermore, the mean and variance used for predicting is learned by having a moving mean and variance estimate during training. Which after each training step updates according to:

$$\mu_{t+1} = m\mu_t + (1-m) \cdot \text{mean}(x) \tag{11}$$
$$\sigma^2_{t+1} = m\sigma^2_t + (1-m) \cdot \text{Var}(x) \tag{12}$$

where m is a momentum term between 0 and 1.[5] The predictions are then made with:

$$\hat{x} = \frac{x - \mu_{final}}{\sqrt{\sigma^2_{final} + \epsilon}} \tag{13}$$

[5] m is typically around 0.99 [33]

4.3.5 Basic Loss Functions

Since minimization of the loss function is the ultimate goal of training, the choice of loss is very decisive of the way the network will work. There are many types of loss functions, some more useful than others, but essentially any differentiable function can be used. In this section, some of the most common loss functions will be outlined.

When performing regression, the end goal is to minimize the difference between reconstructed variables and the truth. This motivates the use of simple differences as loss function and the *Mean Absolute Error* defined as:

$$\text{MAE}(y_{true}, y_{reco}) = \frac{1}{N}\sum_i^N |y_{i,true} - y_{i,reco}|$$

where the difference between reconstruction and truth is punished linearly. Different priorities can lead to other forms of this loss function. *Mean Squared Error* is also very common. This is defined by:

$$\text{MSE}(y_{true}, y_{reco}) = \frac{1}{N}\sum_i^N (y_{i,true} - y_{i,reco})^2 \tag{14}$$

Using this error, outliers contribute significantly to the loss.

In classification, another kind of loss should be used. The *Cross Entropy* (CE) is a measure of distance between two probability distributions in information bits. For multiple classes, the binary cross-entropy $H(p_c, q_c) = p_c \log(q_c) + (1-p_c)\log(1-q_c)$ can be used, where $p_c \in 0, 1$ is the truth variable for class c and q_c is the predicted probability. The multiclass Cross Entropy is then:

$$\text{CE}(y_{truth}, y_{reco}) = -\frac{1}{N}\sum_i^N \sum_c^C p_c \log(q_c)$$

Since $p_c = 0$ or $p_c = 1$ only one term will be nonzero for each class, and both terms will go to 0 only if the prediction is perfect, while going to ∞ for a completely wrong classification [34].

4.3.6 Probabilistic Loss

Since the information available in each event is limited, predictions will generally not be exact.From the first measurements of data to the final reconstruction, small errors and approximations are accumulated. This leads to the actual uncertainties

on a prediction being unknown, which motivates the use of a probabilistic loss function. In addition to making a prediction of the target, the NN also predicts its own uncertainty.

The probabilistic loss is heavily inspired by the maximum likelihood fit as described in reference [34]. Considering an event of target variables, y_{true}, there is some probability of observing the data X, given by $p(X|y_{true})$[6]. But since the goal is to determine $p(y_{true}|X)$ we use Bayes' formula to write:

$$p(y_{true}|X) = \frac{p(y_{true})}{p(X)} p(X|y_{true}) \tag{15}$$

where the probability of seeing the event $p(y_{true})$ and for seeing the data $p(X)$ effectively serve as a normalization constant. The goal is now given X to produce a proper distribution for the target variable y. This is done by using the UAT and trying to approximate the right hand side of Eq. 15 as an NN model, giving us:

$$p(y_{true}|X) \approx f(y_{true}, X, \theta) \tag{16}$$

Where θ represents the trainable parameters of the NN. When it is time to tune the network it is done by fitting a maximum likelihood. Here it is assumed that the different events are independent, such that the probability of getting a set of y values will be:

$$L = \prod_i p(y_i|X_i) \approx \prod_i f(y_i, X_i, \theta) \tag{17}$$

With a set of $\{(X_i, y_i)\}$ it is now the goal to choose θ such that L is maximized. Since a product of many probabilities easily vanish when any one of them nears 0, the logarithm is taken:

$$\log(L) = \sum_i \log(f(y_i, X_i, \theta)) \tag{18}$$

And with the goal of finding $\text{argmax}_\theta \log(L)$, the similarity with a loss function motivates the choice to define the probabilistic loss as:

$$\mathcal{L} = -\sum_i \log(f(y_i, X_i, \theta)) \tag{19}$$

where the sign is introduced, such that minimizing the loss maximizes the likelihood.

[6]The notation $p(A|B)$ reads as the probability of A given B

Normal Distributed Regression Assuming that the probability density function of a truth variable given the data is normally distributed:

$$p(y_i|\mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(y_i - \mu_i)^2}{2\sigma_i^2}\right) \tag{20}$$

It is necessary to predict the two variables, μ and σ given some data X. This is done by creating an NN that has two outputs: (μ_i, σ_i) and then minimizing the loss given by:

$$\mathcal{L} = \frac{(y_i - \mu_i)^2}{2\sigma_i^2} + \log\sigma_i + \text{const} \tag{21}$$

Introducing $\kappa = 1/\sigma^2$, the more numerically stable version is:

$$\mathcal{L} = \frac{\kappa_i}{2}(y_i - \mu_i)^2 - \frac{1}{2}\log\kappa_i + \text{const} \tag{22}$$

This notion can be extended to multivariate distributions where the prediction of covariances become possibly. If instead the covariance matrix is assumed to be diagonal, this simplifies to taking the product of individual Gaussians.

Spherical Likelihood Regression However, directional statistics do not naturally take place in Cartesian space but on the surfaces of N-dimensional unit hyperspheres. Here, instead of minimizing based on direct differences between prediction and truth, one minimizes based on the subtended angle Ω between two points on the hypersphere.

The 3-dimensional von Mises-Fisher distribution is a close approximation to the normal distribution on the 3-dimensional sphere:

$$vMF(\boldsymbol{x}|\boldsymbol{\mu}, \kappa) = \frac{\kappa}{4\pi sinh(\kappa)} e^{\kappa \boldsymbol{\mu}^T \boldsymbol{x}} \tag{23}$$

$\boldsymbol{\mu}$ and $\boldsymbol{x}$ are unit vectors and κ is the concentration parameter.[7]

In spherical coordinates the cosine of the angle between the unit vectors is:

$$\begin{aligned}\boldsymbol{\mu}^T\boldsymbol{x} = cos(\Omega) =& sin(\theta_\mu)cos(\phi_\mu)sin(\theta_x)cos(\phi_x) \\ &+ sin(\theta_\mu)sin(\phi_\mu)sin(\theta_x)sin(\phi_x) \\ &+ cos(\theta_\mu)cos(\theta_x)\end{aligned}$$

[7]For small κ, vMF is approximately uniform while for large $\kappa \approx \frac{1}{2\sigma^2}$

In general, for directional probability distributions, the angular standard deviation is given by [35]:

$$\sigma = \sqrt{-2ln\left(E[cos(\Omega)]\right)} \tag{24}$$

For the 3-dimensional vMF :

$$\begin{aligned} E_{vMF}[cos(\Omega)] &= 2\pi \int_0^\pi cos(\Omega) \frac{\kappa}{4\pi sinh(\kappa)} e^{\kappa cos(\Omega)} d\Omega \\ &= coth(\kappa) - \frac{1}{\kappa} \end{aligned}$$

Notes on numerical implementation can be found in 12.1.1.

Since the network cannot learn azimuth and zenith separately, when using $-ln(vMF(\boldsymbol{x}|\boldsymbol{\mu}, \kappa))$ as the loss function,[8] it can be hard to make specified progress on either the zenith or azimuthal angle. For neutrinos, the azimuthal angle is harder to regress, so if instead only a prediction for the zenith angle is wanted, the following becomes useful.

Polar Likelihood Regression The von Mises-Fisher distribution for the 2-dimensional case is a close approximation to the wrapped normal distribution, and is therefore also known as the circular normal distribution:

$$\mathcal{CN}(\boldsymbol{x}|\boldsymbol{\mu}, \kappa) = \frac{e^{\kappa \boldsymbol{\mu}^T \boldsymbol{x}}}{2\pi I_0(\kappa)}$$

where $\boldsymbol{\mu}^T \boldsymbol{x} = cos(\Omega)$ is just the cosine of the angle difference of the angle in question and $I_0(\kappa)$ is the modified Bessel function of order 0.

The standard deviation is then given by

$$\sigma = \sqrt{-2ln\left(\frac{I_1(\kappa)}{I_0(\kappa)}\right)}$$

.[9]

A few notes on its numerical implementation can be found in the appendix section 12.1.2

Spherical Likelihood as Two Polar Likelihoods If instead a simultaneous probabilistic prediction of both the azimuth and the zenith angle is wanted, a second option, as opposed to using the vMF, is to treat the two angles independently and multiply their individual probabilities, which is then the likelihood to be maximized. More concretely, this second option consists of multiplying the circular normal distribution for the azimuth angle with the circular normal distribution for the zenith angle.

$$\begin{aligned} \mathcal{P}(\phi_x, \theta_x | \phi_\mu, \theta_\mu, \kappa_\phi, \kappa_\theta) &= \mathcal{CN}(\phi_x | \phi_\mu, \kappa_\phi) \mathcal{CN}(\theta_x | \theta_\mu, \kappa_\theta) \\ &= \frac{e^{\kappa_\phi cos(\phi_x - \phi_\mu)}}{2\pi I_0(\kappa_\phi)} \frac{e^{\kappa_\theta cos(\theta_x - |\theta_\mu|)}}{2\pi I_0(\kappa_\theta)} \end{aligned}$$

This yields a separation of the angles, which allows the network to learn them separately. Furthermore, it also provides an uncertainty on the zenith and azimuth predictions themselves instead of the subtended angle. A comparison of the different loss functions for directional regression can be found in Sec. 7.4.

4.3.7 Adding Distribution Constraints

Since the above loss functions consider each event individually, they may optimize for individual events and not approach the general distribution of the target variable correctly. One can attempt to correct this in several ways, but not without some loss in precision. One way of doing this is to add a loss based on simple distribution statistics like the mean and standard deviation, and adding loss if the predicted distribution does not match these. If instead, one desires to correct the distribution more particularly, differentiability and thus backpropagation quickly becomes an issue, as e.g. the difference between two histograms would not be a differentiable metric. Another option is to correct the distribution using a kernel density estimate (KDE) of the true distribution, and taking the L1 norm between the true and predicted distributions as an added loss term. A KDE works by taking the sum of differentiable probability density functions (e.g. Gaussians), letting the mean of any PDF correspond to a single data point.[10] As one gets more data, this approaches the true distribution of the variable in question while maintaining differentiablility.

[8] Since they are combined into a single scalar

[9] Since $E_{\mathcal{CN}}[cos(\Omega)] = \frac{I_1(\kappa)}{I_0(\kappa)}$

[10] Choosing σ of the Gaussian is more subjective, but unimportant for large datasets.

4.3.8 Performance Metrics

When models are trained with different loss functions, the loss itself cannot immediately be used to determine which model is better. It is therefore necessary to define some performance metrics that indicate how well a model works, independently of the loss. Here we focus on metrics indicating median absolute deviation, bias, and spread.

For **energy predictions** ($\log_{10}(E)$) the main interest is the absolute residuals. The bias can be estimated by the mean of the residuals:

$$\frac{1}{N}\sum_{i}^{N}(y_{i,reco} - y_{i,true}) \tag{25}$$

or one could use the median to reduce the importance of outliers. To estimate the variance of the residuals, one could calculate the standard deviation of the residuals. However, as the distribution is not necessarily Gaussian, the inter quartile range (IQR) can be used as an estimate which is also more robust to outliers. The IQR is the difference between the 3rd and 1st quartile. Here we use

$$w = \frac{IQR(y_{reco} - y_{true})}{1.349} \tag{26}$$

where w would be equal to the standard deviation if the residual distribution were a Gaussian distribution.

For **direction predictions**, a few metrics are useful. Firstly, both the zenith and the azimuthal angle can be predicted and the residuals here can be treated like the energy. To account for the periodicy of the angles one could compute the residuals by $\sin^{-1}(\sin(A - B))$.

Furthermore, a useful metric is angular difference. With unit vectors, this is found by $\cos(\Omega) = \vec{a} \cdot \vec{b}$.[11]

In this project, for all angular metrics, we focus on the describing the magnitude of the residuals. To capture the distributions well, the 16th, 50th and 68th percentile of the absolute residuals are reported. The median (50th percentile) is chosen as a robust metric to indicate the general size of the residual, while the 16th is chosen as an estimate of "median - 1 σ".[12] The 68th percentile is chosen to compare with previous work [25].

[11] For spherical coordinates see Spherical Likelihood Regression in Sec. 4.3.6

[12] Since one σ should encapsulate 68% of a distribution, $\approx$ 34% lay below the median

For **classification** tasks, the accuracy can easily be calculated as the fraction of predictions where the prediction is equal to a given target label. However, this does not account for unbalanced populations where the most commonly occurring class could dominate the accuracy. Furthermore, the prediction is usually a probability between 0 and 1 instead of exactly 0 or 1, so a threshold needs to be chosen. The Receiver Operator Characteristic (ROC) curve takes both of these facets into account. In a ROC-curve the true positive rate (TPR) and negative positive rate (NPR) are calculated based on the probability distributions for a given threshold. This is done for an appropriate number of thresholds and the ROC curve is then traditionally visualized as the TPR as a function of FPR. A perfect classification would thus go towards the upper-left corner, and a random classifier will lie along the diagonal. The area of the ROC curve (AUC) is now a measure of how well the model classifies the sample, with AUC = 1 being perfect and AUC = 0.5 being random.
For an example of how the ROC-curve and AUC is used see Fig. 23.

4.4 Graph Neural Networks

Here we introduce the concept of graphs, and the specific frameworks used for learning on them. We will focus on the methods most relevant to the work of the group.

4.4.1 Why Graphs?

In recent years, *Graph Neural Networks* (GNN) have lent themselves to the realm of physics, as the focus on nodes and their mutual relationships appear to constitute a representative manifestation of the real world [36].

Many types of NNs have focused on unvarying grids (like pixels in an image or letters in a sentence), but most struggle with irregular relationships between data points. Especially since IceCube is built from many individual sensors with internal differences, placed in a very peculiar structure, the choice of the much more flexible graph framework seems obvious.

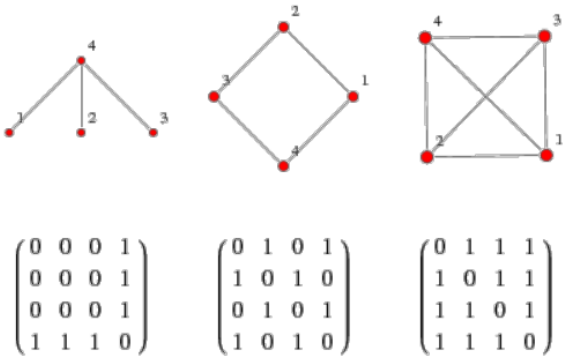

Figure 15: Three simple graphs with their respective adjacency matrices. (Image source: [39])

4.4.2 Graphs

A graph is made up of nodes which are connected by edges. A graph has N nodes with features given by the vector $x_0 ... x_F$ which is connected by edges which can also have features. In the `Tensorflow Spektral` and `PyTorch Geometric` notation ([37],[38]), we encode the information for a single graph in the following way:

- Let N, F, G be scalars referring to the number of nodes (N), number of features for each node (F), and number of features for each edge (G).
- Let $\boldsymbol{X}$ be a $N \times F$ matrix containing the features of the nodes.
- Let $\boldsymbol{A}$ be the adjacency matrix with $N \times N$ entries. The adjacency matrix is defined by $A_{ij} = 1$ if node i is connected to node j, and 0 otherwise. In general $\boldsymbol{A}$ is symmetrical, but it is possible to make a *directed* graph, where $A_{ij} \neq A_{ji}$. For a graph without self-loops, the adjacency matrix must have 0's on the diagonal. The adjacency matrix is most commonly treated as a sparse matrix, where the information is boiled down to a $2 \times N_{edges}$ index matrix and a vector of the values for each entry - all other entries are expected to be 0. An example of simple graphs with their adjacency matrices can be seen in Fig. 15.
- Let $\boldsymbol{E}$ be an $N_{edges} \times G$ matrix containing the edge features.

4.4.3 Aggregation and Pooling

The GNN-framework allows for variable size input, i.e. a model can be independent of how many nodes are in any given graph. If only a prediction for each node is wanted[13], it is straightforward since the mapping consists of: $f : \mathbb{R}^F \rightarrow \mathbb{R}^{N_{targets}}$, however if a prediction for the whole graph is wanted, the mapping becomes: $f : \mathbb{R}^{N \times F} \rightarrow \mathbb{R}^{N_{targets}}$ leading to complications since N can vary for each input. This is dealt with through *aggregation* and *pooling*. The two terms cover the same idea i.e. taking a set and returning a single element. In this project, for the sake of clarity, aggregation is used to denote combining said elements, whereas pooling is used for "boiling down" the collection into its most important parts. A weighted sum is an example of aggregation: $\text{Agg}(m_n) = \sum_n^N \alpha_n m_n$. An example of pooling could be finding the mean, variance, minimum, or maximum of the set. It should be noted that any method generally needs to be permutation invariant since the graph structure usually does not offer any hierarchical structure.

There are a few more complicated methods of pooling and aggregation that could be used to extract other information from the graphs. These include top-k/DiffPool [14] and attention based aggregation. A simple attention based aggregation used in this project is a weighted sum $pool(h_i) = \sum_i^N \alpha_i h_i$ where the scalar attentions, α_i, are calculated from the hidden states. The attentions are normalized by SoftMax.

4.4.4 Message Passing

To create a convolution on a graph, several different approaches can be taken. In the graph literature, a node which is connected to another node is called a neighbour. A set of these are thus a neighbourhood (not to be confused with the k nearest neighbours algorithm). One very general approach of a graph convolution is to build a layer that sends a message from every node to all its neighbours and updates the nodes according to the incoming messages. This is the idea behind the very general

[13] In the graph literature this is called a node-level predictions

[14] which reduces the size of the graph to k nodes or by a fraction

Message Passing (MP) layer [40].

An MP layer is composed of a message function, an aggregation function, and an update function. Given hidden states of a node $\vec{h}_i^t$ and the edge features e_{ij} between the node and its neighbours, a message is calculated according to

$$m_{ij}^{t+1} = M_t(\vec{h}_i^t, \vec{h}_j^t, \vec{e}_{ij}) \tag{27}$$

where M_t is an arbitrary differentiable function (most typically an MLP). Now the messages are aggregated by a function that takes multiple messages and returns a single instance:

$$m_i^{t+1} = Agg(\{m_{ij}\}), \quad j \in N(i) \tag{28}$$

where $N(i)$ is the set of neighbouring nodes to i and Agg is the permutation invariant aggregation function. Lastly, an update function is applied according to:

$$h_i^{t+1} = U_t(h_i, m_i^{t+1}) \tag{29}$$

where the update function, like the message function, can be anything that takes a hidden state and returns another.

4.4.5 Convolutions

While MP is a very general layer, a few specific layers from literature were used in this project, some based on convolutions. Here these layers will be described.

The basic Graph Convolutional Layer, **GCN** is inspired by convolution layers in image recognition, but instead of using neighbouring pixels, the neighbouring nodes are used [41], [37]. Its internal calculations can be defined as:

$$\mathbf{X}' = \mathbf{AXW} + \mathbf{b} \tag{30}$$

Where $\mathbf{A}$ is the adjacency matrix, $\mathbf{X}$ denotes the attributes of the nodes, $\mathbf{W}$ is the weight matrix, and $\mathbf{b}$ is the bias. This isn't directly used in the project but serves as a baseline for more advanced layers.

Another such layer is **GraphSAGE**, [15] where the central idea is to aggregate the neighbourhood of a node and then apply a linear layer to encode the information of the neighbourhood. Mathematically we describe the process of updating the hidden state of node i at time t from h_i^t to h_i^{t+1} as first calculating an aggregated neighbourhood vector:

$$h_{N(i)}^t = Agg(\{h_j | j \in N(i)\}) \tag{31}$$

now the aggregation is concatenated with the node features and a linear layer is applied. So that the next layer can be calculated by:

$$h_i^{t+1} = (h_i^t \parallel h_{N(i)}^t)\boldsymbol{W} + b \tag{32}$$

where $\boldsymbol{W}$ and b are trainable parameters. Now, by applying the GraphSAGE layer multiple times, a bigger neighbourhood can be considered for updating a single point [42].

GATConv implements a technique known as *attention*. Attempting to mimic the awareness of conscious animals, *attention*-layers has in recent years proved to be more precise and versatile than traditional methods [43].

The layer works as a standard convolutional layer (GCN), but uses the attention mechanism to weight the adjacency matrix [44]:

$$\mathbf{X}' = (\boldsymbol{\alpha} * \mathbf{A})\mathbf{XW} + \mathbf{b} \tag{33}$$

Where $*$ is the Hadamard product[16] and $\boldsymbol{\alpha}$ is the attention matrix, which is usually calculated as a normalized response to the inputs. Each attention trainable vector $\boldsymbol{a}$ is called an *attention head*: [17]

$$\alpha_{i,j} = \text{SoftMax}\left(LRELU\left(\mathbf{a}^T(\mathbf{XW})_i || (\mathbf{XW})_j\right)\right) \tag{34}$$

ECCConv implements edge-conditioned convolutions. The idea is to utilize message passing with a dynamic set of weights computed based on the edge-features:

$$\mathbf{x}_i' = \mathbf{x}_i\mathbf{W}_{\text{root}} + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \,\text{MLP}\left(\mathbf{e}_{j \to i}\right) + \mathbf{b} \tag{35}$$

where W_{root} is the base weight matrix and MLP is a multi-layer perceptron that outputs an edge-specific weight as a function of edge attributes [45].

[15] In this section the implementation from *Spektral* [37] which differs slightly from the original idea presented in [42]

[16] Element-wise matrix multiplication: $(A * B)_{ij} = A_{ij}B_{ij}$

[17] There can be multiple heads that "vote" on the final attention

4.5 Custom Layer Implementations

Since the Message-Pass (MP) layer is a general framework, the group implemented our own ideas in a few different ways in this project.

General MP layer is built by having two MLPs and an aggregator. First messages are computed by an MLP taking $m_{ij} = MLP(x_i, x_j, e_{ij})$ as input. For each node the messages are now aggregated by calculating some statics over the incoming messages, such that the: minimum, maximum, mean, and variance are calculated and concatenated. Now another MLP is taking the aggregated features and updating the node.

The KHop MP layer is inspired by the Simple Graph Convolution (SGConv) operator proposed in [46], which attempts to maintain the simplicity of the basic MP layer, while also extending the neighbourhood size for each node. This is done by specifying the K-neighbours each node should aggregate information from, i.e. giving K=2 would mean that one's neighbourhood would take in information for the 2-hop neighbourhood, while K=[1,2] would mean that each node would receive and aggregate information from its 1-hop neighbours first and then relay that information in another encoded message to the the 2-hop neighbourhood.

The motivation behind **GRUConv** is that, by utilizing the "message-pass" analogy, one could instead consider a "conversation" between neighbours where, instead of one-directional information sharing, one could implement a call and response method. The idea is, that instead of two simultaneous one-way message-passes, the second message pass is modified based on the original message and the updated node. A memory unit is thus needed and for this; the Gated Recurrent Unit (GRU) - cell is used. The GRU takes as input a feature vector x and a contemporary hidden vector h and updates the hidden state via the following:

$$\begin{aligned} r &= \sigma\left(W_{ir}x + b_{ir} + W_{hr}h + b_{hr}\right) \\ z &= \sigma\left(W_{iz}x + b_{iz} + W_{hz}h + b_{hz}\right) \\ n &= \tanh\left(W_{in}x + b_{in} + r * \left(W_{hn}h + b_{hn}\right)\right) \\ h' &= (1 - z) * n + z * h \end{aligned} \tag{36}$$

where σ is the sigmoid function. h then plays the role of the message to be passed with x the sender of the message. h' is then the new message calculated based on the receivers updated features and the original message h. However, in this project, GRUConv was only used on graphs with an adjacency matrix where only one of the rows had entries different from 0. Thereby a single node, called the global node, was responsible for all the communication between nodes. The resulting convolution, denoted **Global GRUConv** is then:

$$\begin{aligned} h_i' &= GRU(x_i^t || g^t, h_i^t) \\ g^{t+1} &= U_g(g^t, A(\{h_i' | \forall i\})) \\ h_i^{t+1} &= GRU(x_i || g^{t+1}, h_i') \\ x_i^{t+1} &= U_x(x_i^t, h_i^{t+1}) \end{aligned} \tag{37}$$

Where U is an update function, A is an aggregation function, and g denotes the features of the global node.

4.6 Explainability

Due to the large amount of parameters with complex relationships in any NN, explaining exactly how a given set of observations is turned into a prediction becomes highly non-trivial. NNs are incredible at finding correlations, but are these correlations meaningful?

NN explainability is a highly active research area, and there is not yet any single method that dominates the area. Symbolic regression, SHAP (SHapley Additive exPlanations), and gradient analysis are all possible methods of doing so. Here we focus on SHAP and gradient analysis.

4.6.1 SHAP

SHAP values are built on the game theoretical concept of Shapley values [47] [48]. The core idea is to determine the reward a game-actor should receive according to their contribution, given that contributions are judged fairly. It is calculated for each feature (actor) i by prediction with a model f (playing the game) using the feature space S without i and a feature space $S \cup \{i\}$ which includes the feature i. The Shapley value is now calculated by a weighted sum of the contributions of i.

$$\sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left(f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)\right) \tag{38}$$

Where the sum is over all subsets that do not include the parameter i. In the actual implementations, several assumptions and approximations are made, since evaluating Eq. 38 requires a factorial

amount of terms. Some approaches approximate even further, allowing for creation of a larger statistical sample ([49], [50]).

4.6.2 Gradient Analysis

Another approach, facilitated by the necessity of the use of gradients used in back-propagation (Sec. 4.3.1), is to analyse the gradients of the outputs with respect to some input. This is a much less computationally expensive approach, facilitating analysis of a larger amount of events.

Direct Gradient Analysis. This method is known as *saliency mapping* in computer vision [51]. Here one simply maps $\frac{\partial_i}{\partial x_j}$, but this method suffers from a series of drawbacks. First of all, it is unclear how one can compare gradients that have different units, i.e. the gradient of logarithmic energy with respect to charge is not comparable to the gradient of the zenith with respect to position. Furthermore, the general size of the gradients can differ between both parameters, as well as between events of different size.

Integrated Gradients. This method, developed in [52], builds upon the same idea, but tries to capture the fact that decision boundaries may have happened before reaching the current value of a given parameter. Thus this approach opts for approximating the path integral of the gradient from some baseline x_i' (often 0's for all parameters) to the actual input, i.e. the integrated gradient for output y_j with respect to variable x_i for the neural network denoted F. The distance integrated between the baseline and the input is controlled With the "perturbation parameter" α, where $\alpha = 0$ is the baseline and $\alpha = 1$ is the actual input.

$$\text{IntGrad}(y_j(x)) \equiv (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F\left(x' + \alpha \times (x - x')\right)}{\partial x_i} d\alpha$$

Computationally this is done as a Riemann sum over m steps. This method works axiomatically for classification tasks i.e. $F : R^n \rightarrow [0, 1]$, but for regression tasks the method is relatively unexplored. The main drawback is the choice of baseline, which is highly subjective.

5 Data

This section serves as an introduction to the data used in this project. First the main datasets will be presented. Afterward, an overview of the input and target variables will be provided and the relation between them will be demonstrated through distributions. Lastly, we will present methods for processing the database format to graphs suitable as inputs to GNNs.

5.1 Databases

In this project, the focus was on simulated data in the IceCube experiment. Mainly muons from the MuonGun-simulation and Neutrinos from the Genie simulation (both described in Sec. 3.5) were used. The data was processed and came in two database files, which we will refer to as MuonGun and OscNext. The first is a collection of simulated muons and the latter is a mix of different neutrino flavours and muons. Table 1 shows the total amount of events of each particle type as well as the total dataset size. In this project MuonGun was cleaned at level 2 and OscNext at level 7 (see Sec. 3.6).

5.1.1 Input Features

The databases each consist of two tables; truth and features. The features contains all the information read out by the DOMs, of which the most pertinent parameters are:

- *event_no*: Which event the DOM activation belongs to.
- *x*, *y*, *z*: The position of the DOM that made the readout.
- *time*: The time of the readout in ns. The trigger time is set to $t = 0$.
- *charge_log10*: The charge measured in the DOM.
- *pulse_width*: The width of the measured waveform. Due to the digital nature of the DOMs this parameter is either 1 or 8 ns., depending on the trigger.
- *SRTInIcePulse*: A Boolean variable corresponding to whether the DOM activation survived the SRT cleaning. This parameter is only in

Table 1: Amount of data in the simulated datasets used

(10^6 events)	μ	ν_e	$\bar{\nu}_e$	ν_μ	$\bar{\nu}_\mu$	ν_τ	$\bar{\nu}_\tau$	Total
MuonGun	1.5	0	0	0	0	0	0	1.5
OscNext	0.2	1.3	0.6	3.0	1.4	1.4	0.5	8.3

the MuonGun database since the OscNext dataset only contained readouts that passed the cleaning

- *rqe*: All DOMs are created equal but some are more equal than others: "Relative Quantum Efficiency" is the estimated quantum efficiency of a sensor (OscNext specific).

To see how DOM pulses are distributed in the detector, the density of the x and z coordinates are displayed in Fig. 16. Most considered events pass through the DeepCore part of the detector.
In MuonGun data, the veto module can also be seen around $z \approx 100m$ to $z \approx 200m$, but since this is specifically made to clean the sample from muons, these events do not pass the level 7 cleaning and are not present in the OscNext sample.

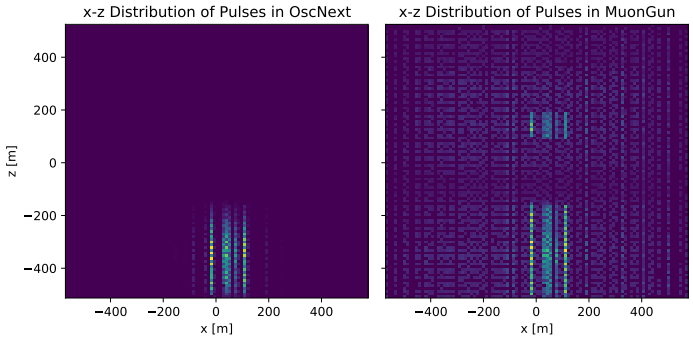

Figure 16: Overview of the pulses' x-z distribution showed in 2D histograms.

5.1.2 Target Features

In the truth table, the parameters used for generating the simulation can be found. For this project the primary focus was on reconstructing the *azimuthal* and *zenith* angle along with the *energy*. The azimuthal angle is lies in the xy- or equatorial plan and goes from 0 to 2π. The zenith angle is the polar angle going from 0 to π with 0 being straight up and the horizon at $\pi/2$. The energy is given in log(E/GeV). The individual distributions and the pairwise density plots can be seen in Fig. 17. The energy of the muons is in general much higher, and they originate from above the horizon while neutrinos come from the full polar range.

From Fig. 17 it can also be seen that the azimuth angles are almost uniform. This is expected since there should be a rotational symmetry around the z-axis. There are a few peaks (most clearly in the MuonGun sample) which can be accredited to the hexagonal structure of the detector layout in the IceCube Experiment. As can be seen in Fig. 35 in the Appendix, none of the target features share any notable correlation.

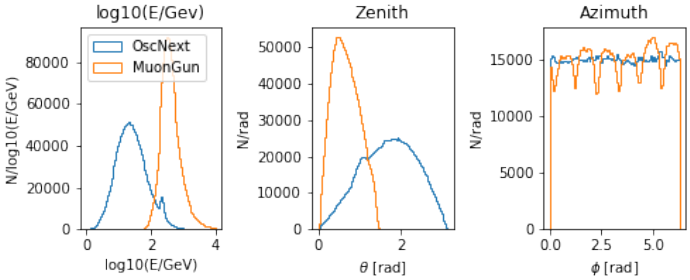

Figure 17: Distributions of the target variables for muons and neutrinos. As OscNext contains multiple particles, the dataset has some quirks, like the muon energy/zenith bumps.

Since the DOMs have a minimum threshold for measurement, a higher energy is associated with more readouts and more data for a single event. For this reason it can often be beneficial to evaluate models in different energy ranges to measure strengths and weaknesses. In data, this is not possible, but one could instead use the total amount of triggered DOMs. In Fig. 18 the energy distribution and number of pulses distribution that pass the SRT cleaning is displayed. A clear correlation is seen between the two features, within each dataset.

5.2 Data Preparation: from Data to Graphs

Before applying the GNN-framework, it is advantageous to perform multiple layers of pre-processing in order to achieve the best prerequisites for the

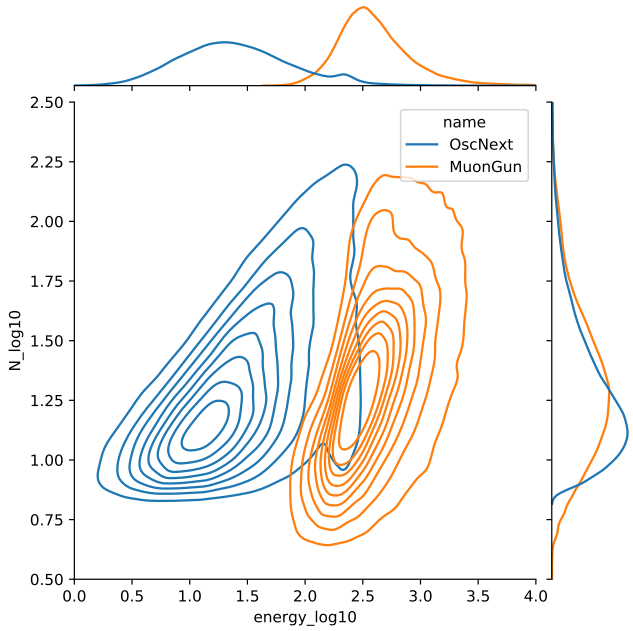

Figure 18: Contour plot of $log_{10}(E)$ and $log_{10}(N_{DOM})$ distributions in MuonGun and Oscnext.

machine learning methods.

5.2.1 Scaling

Contrary to, decision trees, the inputs in NNs are directly used in computing the output. This means that inputs that are too large or too small infuse numerical instability into the learning process.

Therefore, it is advantageous to scale the input. There are multiple ways this can be achieved, where the most well-known methods include: *Min-Max-scaler*, which linearly forces the whole sample to lie in the range [0,1], and thus suffers in the presence of outliers. *RobustScaler* attempts to counter this by translating by the median and scaling by the interval spanned by the innermost 50% of the sample. The *Standard-scaler* is also useful for data that is normal-distributed, where the sample is translated by the mean of the sample and scaled by the variance. Many other more sophisticated methods exist, but are not used in this project.

Usually features are scaled independently, however, in order to keep the symmetry and not distort distances the x, y and z variables were scaled by the same factor.

5.2.2 Feature Selection and Creation

Most of the node features used by our group were taken directly from the databases. These include positions, trigger time, measured charge and pulse width.
An additional interesting feature, is the SRTInIcePulse property, which was only available in the MuonGun dataset. This is a Boolean variable that refers to whether or not a specific node 'survived' SRT-cleaning. We found that handing the model this information gave a greater increase in predictive power than simply removing the noisy nodes (see Sec. 5.1.1).

For this project all prediction tasks were event-wise, as opposed to e.g. providing a prediction for each DOM. In the GNN-framework this can be treated as the creation of a node that carries the global features of the graph, which are ultimately used to create the prediction. The applications of the global node include either creating it in the early layers and treating it as a part of the adjacency matrix, or simply creating it in the final layers by aggregating the nodes of the graph.

One can also create a set of global features directly from the input, and not treat it as a node, keeping it as separate information to be given to a decoding MLP later. This could be the summary statistics like the mean, minimum, or variance of a feature variable across all DOMs.

Apart from the adjacency matrix, the other unique attribute of graphs is the possibility of using edge features. Since the adjacency matrix comes in a sparse matrix with an index matrix of size $N_{edges} \times 2$, features comparing two nodes can easily be calculated on the fly. With this method, we calculated a set of edge features consisting of difference in log_charge and time, Euclidean distance between the nodes, and the normalized vector between them. Furthermore, we added a Boolean value showing if the pulses were within the speed of light.

5.3 Graph Creation

As there is no inherent edge structure in IceCube, an open question is what graph structure to embed. In this project, two different approaches were ultimately used (though others were attempted, see for example Sec. 13 in the appendix). One was to define the adjacency matrix from k nearest neigh-

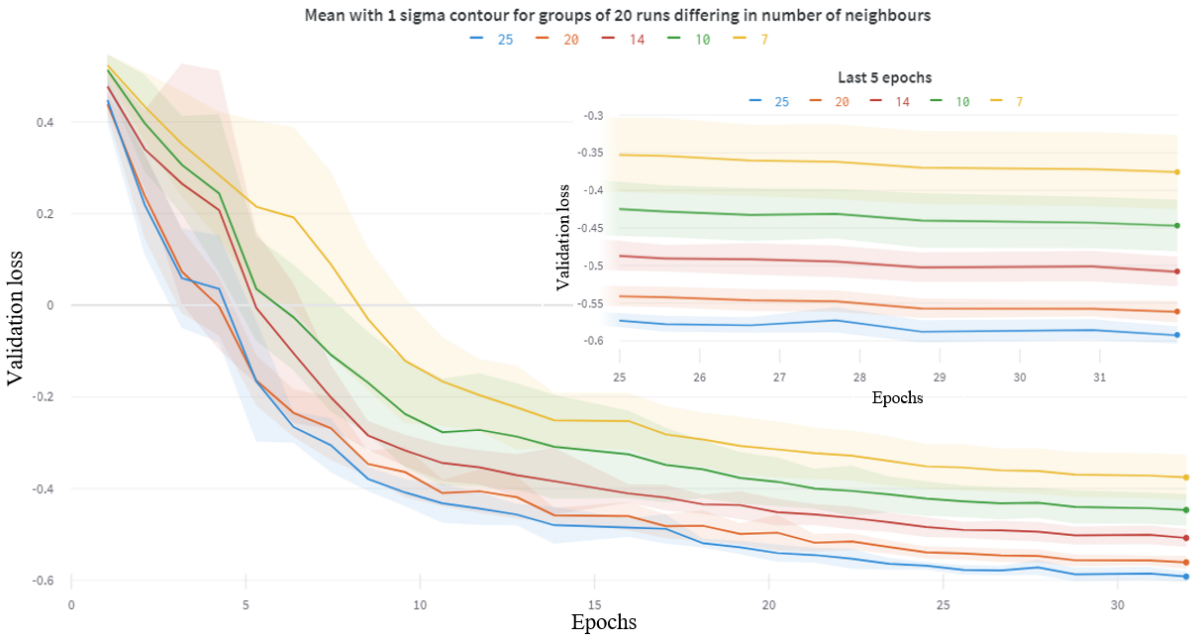

Figure 19: Lines show average loss of group with $\pm 1\sigma$ contours. Doing a sweep of *regularization*, *aggregation method*, *dropout* and *number of neighbours*, we observe that an increase in number of neighbours further decreases validation loss. Each group has 20 runs, with different configurations in the mentioned hyperparameters, implying that the number of neighbours is consistently an important parameter

bours.[18] Here the choice of number of neighbours was paramount. Another approach was to define a global node placed in the center of charge, which is connected to every other node, such as Global GRUConv defined in Eq. 37. The latter method had the advantage that it could easily be parallelized on a GPU and thus offers a significant speed-up and eliminates the need to create edges before training.

5.4 Hyper-Parameter Sweep

As we sought to find the best model configuration, there was a lot of trial-and-error. Answers to questions like "Should I use regularization?" and "How many neighbours should each point have?" can be informed by a heuristic educated guess, but in reality one has to treat model training as an experiment. To find the optimal configuration of parameters you can do a *hyper-parameter sweep*, testing out all permutations of possible parameter choices. This is extremely computationally heavy, since it requires training $n_i!n_j!...n_k!$ models (where n_i is the number of choices for parameter i and so on).[19] The result of such a sweep for the StateFarm architecture training on the OscNext data can be seen in Fig. 19. All runs are grouped by the number of neighbours desired when creating the adjacency matrix, while other parameters are varied. This shows that the higher number of neighbours consistently improves the models when regressing on OscNext.

6 Models

In this project, each group member created a GNN architecture. This section will present the different approaches to this task. For ease of readability and comparison, we have kept a consistent color coding for the different models in both text and plots. The models presented here are the versions used for predicting in the OscNext sample.[20]

6.1 Global GRU Convolution (GGConv)

This model is the result of a pursuit of a graph representation for which the adjacency matrix is trivially given, such that no pre-processing of each individual event is necessary. Ultimately, the choice was to define a central node placed at the center of charge and containing the sum, mean, variance,

[18] through Euclidean distance measure between the spatial coordinates

[19] Unless more sophisticated search method like Bayesian optimization are used

[20] Only small modifications are made to the models when predicting on MuonGun data or performing classifications

maximum and minimum of the input features across all nodes in the graph. The connections in the graph are then only between every input node and this new central node. The central node is encoded separately from the other nodes, after which the convolution takes place. The convolution is done with **Global GRUConv** as described in Eq. 37. The nodes all send a message to the central node which are aggregated with attentions and the central node is updated. A message for each node is then constructed based on the original message and the updated central node, with which each node is updated. Every iteration of the central node is 'saved' and concatenated together, from the beginning to the last update, along with a pooling of the last node representations. A single decoder is then used to turn this into the wanted prediction. For a model diagram, see the Appendix Fig. 34.

6.2 Extended Neighbourhoods (StateFarm)

The idea behind this model is to extend the neighbourhood of each node as much as possible while avoiding over-smoothing, using the KHop MP described above. The adjacency matrix is created by a kNN algorithm. The model uses a mixture of KHop and GraphSAGE layers for the graph encoding, focusing on reaching the 2-hop neighbourhood in an optimal way. The graph encoding is pooled with max/min/mean-pool and combined with extracted global graph statistics, and decoded by two densely-connected MLP layers (`Tensorflow` Dense layers), and then fed to three separate decoder heads that decode log(E), (θ, ϕ) and $(\kappa_\theta, \kappa_\phi)$. A schematic of the model architecture can be seen in Fig. 20.

6.3 Stepwise Message Encoding (AntHill)

In this model, the adjacency matrix is found by using the kNN implementation. The inputs are then aggregated using min, max, mean, and a sum of each feature and stored for later. Now two General MP layers are applied. The updated nodes are again pooled. Hereafter, two GraphSAGE layers are applied and the graph is pooled one last time. The output from the three pools are now concatenated and sent through a wide MLP with two layers and afterwards split up into a smaller MLP for each output variable. This model has primarily been used to predict directional unit vectors. An illustration of the model can be found in the appendix in Fig. 33.

6.4 Attention-based GCN (LifeGuard)

As an attempt to focus on *attention*, poolings, and a minimal model-size, the idea of LifeGuard was conceived. First, edge features are calculated and stored. The graphs are then encoded by virtue of three message-passing layers with max, min, and mean aggregation respectively. To further convolve the edge features an ECCConv-layer is applied after which the individual batches are normalized. For a final encoding, the graphs are sent through two GraphSAGE layers and a single head attention layer. Between each of these last three layers, a pooling is made - all of which are concatenated. This group of poolings is sent to a three layer deep decoder made of simple Dense layers. Each output (energy, angles, uncertainties) is then routed to a separate smaller Dense decoder before the final predictions are made. A schematic of the architecture can be seen in Fig. 32 in the Appendix.

6.5 Ensemble Models

While the UAT states that a sufficiently wide and deep NN can approach the information limit, local minima in the loss function means this is unlikely. Ensemble models can help mitigate the complications of any single model, as is seen in the popular boosting used in decision tree forests. The theory is that if any single model has a certain chance of getting the right answer, by combining more models the resulting ensemble will be better than any of the individual models.

Much like having multiple people in a jury increases their ability to make an informed decision[21], training multiple independent models will bring their product closer to the truth. The main drawback (apart from the time used training and building multiple models) is that their predictions are rarely independent, making the ensemble inefficient [54].

Since the models proposed in this thesis are of differing architectures, they have different strengths and weaknesses. Therefore, we propose an ensemble model architecture that takes the output from

[21] Often called *the Jury Theorem* [53]

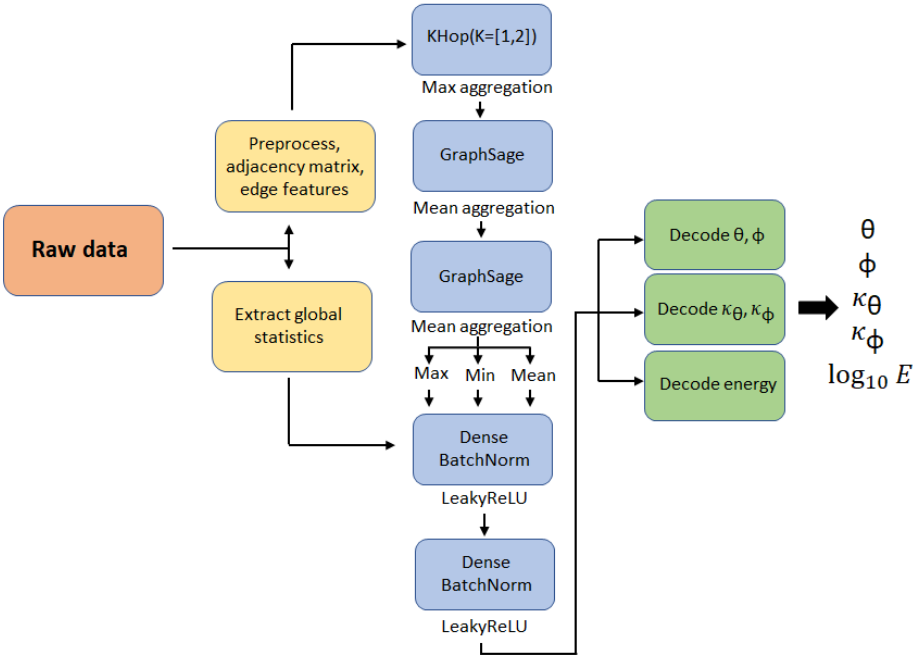

Figure 20: Illustrated StateFarm architecture. All connections are shown, as well as the global aggregated statistics passed around the graph encoder and directly to the decoder.

each model combined with a particle classifier and returns a set of predictions for *energy_log10*, *zenith* and *azimuth*. The ensemble consists of a simple MLP framework with three layers using dropout and batch normalization between them. This is further decoded by a thinner, two-layer MLP for each output variable.

7 Results and Analysis

In this section, the usefulness and performance of the models described in Sec. 6 is evaluated. Firstly, the results on the OscNext dataset, which contains neutrinos, will be evaluated, since this is the primary interest of the IceCube experiment. Secondly, the usefulness of GNNs for muons will be investigated. Primarily, the focus will be on regression of energy and directional parameters. Secondly, on the ability to get a clean sample of stopped muons, and lastly on distinguishing between particle types. Furthermore, it is considered how one could use this method to calibrate the experiment trying to see the effect from the moon on atmospheric muons detected by IceCube and reconstructed by GNNs.

7.1 Neutrino Reconstructions

The main focus of the IceCube experiment is to measure neutrino oscillations. Thus the important parameters for reconstruction are zenith, energy, and particle classification (see Fig. 2). Furthermore, if IceCube were to be a cosmic event alert system, the best possible angle reconstruction, i.e. including azimuth, is important.

7.1.1 Angle and Energy

The models were trained on the OscNext sample with a common train/test split of 80%/20% with target variables consisting of energy, zenith, and azimuth angle. To make the comparison between the models fair, all models were allowed to train for 20 epochs (training on around 132 million events), however further training would improve the scores of the GNNs.[22] A summary of the performance statistics is given in Tab. 2 and the energy dependent performance is displayed in Fig. 21.

In Fig. 21 the performance metrics are shown as a function of energy. All tested models as well as Retro perform better on angle predictions for higher energies up until $\approx$ 100 GeV where the dataset contains significantly fewer events.

The interactions in the detector differ between neutral and charged current as well as between different neutrino flavours. To measure if this had an effect on the accuracy of the GNN-architectures,

[22] Training Statefarm for an additional 30 epochs increased median accuracy by about 6% across all target variables

Table 2: Comparison of the performance metrics for the four models and Retro. The θ, ϕ and Ω are given as the median of the angular difference while the $\log_{10}(E)$ is given as the width of the residuals. The speed is the average inference speed measured on an Nvidia Geforce GTX 3090. LifeGuard does not converge on azimuth angle and is for this reason left blank. The last row shows the average uncertainty for the column, calculated as in [25]. See Fig. 24 for a predicted uncertainty dependent performance plot. The entries in Loss (Angle) use the shorthand defined in Sec. 7.4

	Loss $(\theta, \phi/x, y, z)$	Loss $(log(E))$	θ	ϕ	Ω	$log(E)$	Speed	Params (10^3)
StateFarm	2xPvM	MAE	**12.92** °	31.42 °	34.36 °	**0.2082**	$9 \cdot 10^3$	643
GGConv	vG	Normal	15.94 °	**29.65** °	**32.84** °	0.2105	$3 \cdot 10^4$	176
AntHill	2xPvM/SvM	Normal	14.20 °	33.53 °	36.37 °	0.2357	$1 \cdot 10^4$	2,218
LifeGuard	2xPvM+σ	MAE	15.91°	-	-	0.2634	$2 \cdot 10^4$	2.2
Retro	-	-	15.00 °	37.93 °	40.56 °	0.2390	$O(10^{-2})$	-
σ	-	-	0.01 °	0.05 °	0.04 °	0.0003	-	-

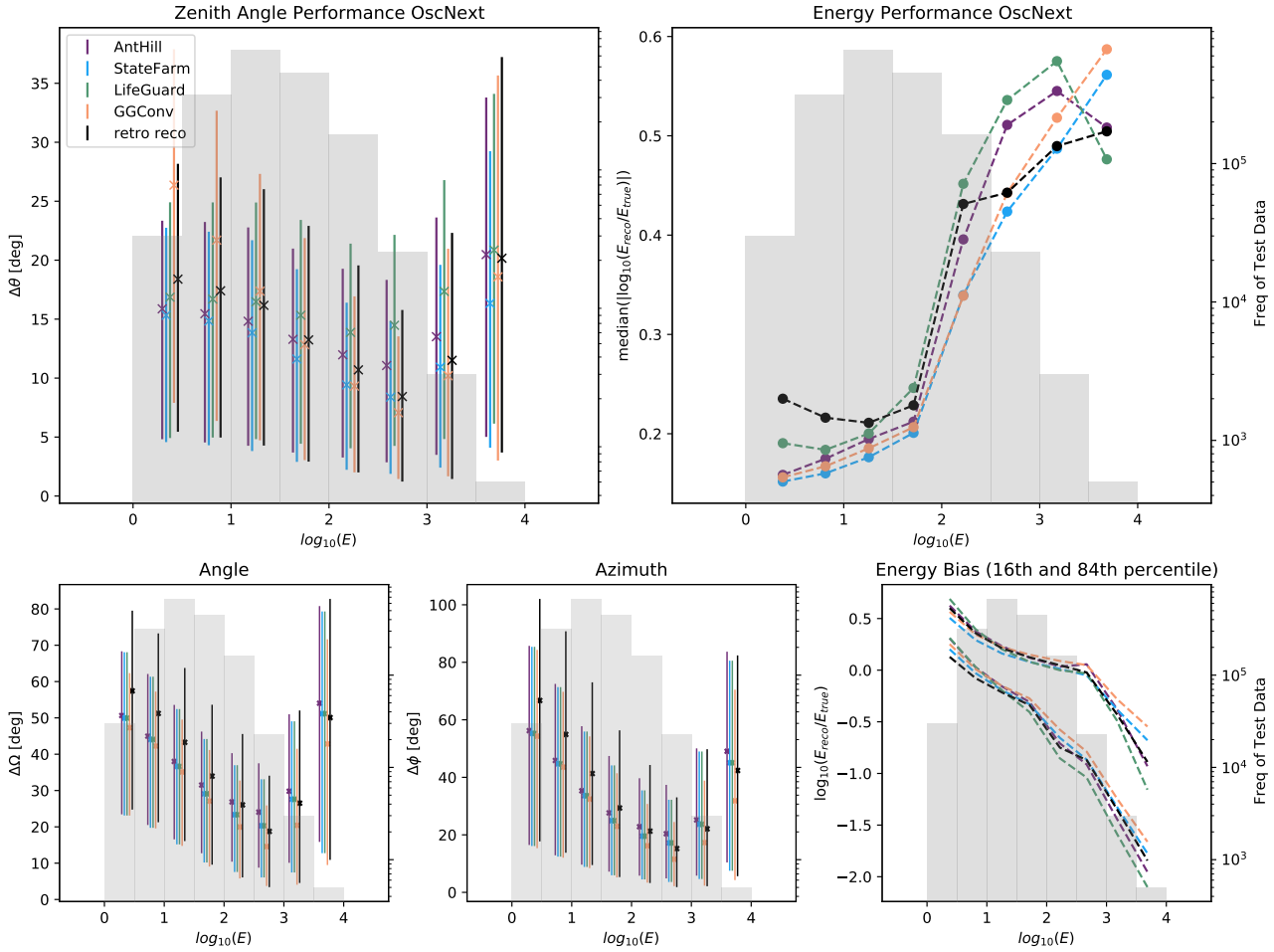

Figure 21: Overview of the performance of the architectures described in Sec. 6 as a function of log(E/GeV). The used metrics can be found in Sec. 4.3.8. In general, a GNN model outperforms Retro in every energy range, for every metric. A similar plot for muon performance can be found in the Appendix, Fig. 36.

StateFarm was evaluated with different subsets of the dataset, split into charged/neutral current as well as the different particle types seen in the experiment. The results can be seen in Tab 3.

Table 3: Overview of the performance score of StateFarm when masking after neutral and charged current (NC and CC) and different particle types.

	Ω	ϕ	θ	$log_{10}(E)$	E - bias
NC	41.24	37.98	15.84	0.2612	-0.25
CC	33.78	30.70	12.56	0.1825	0.02
μ	27.53	22.40	9.71	0.3812	-0.59
ν_μ	31.69	28.65	11.65	0.1827	0.02
ν_e	40.03	37.05	15.34	0.1709	0.02
ν_τ	38.70	35.30	14.97	0.2037	-0.16

7.1.2 Particle Classification

Using GGConv, we attempted to determine the particle identification number. The type of particle was one-hot-encoded and weighted in accordance with Tab. 1. No difference between particles and their corresponding antiparticles were made. The resulting AUC can be seen in Fig. 22.

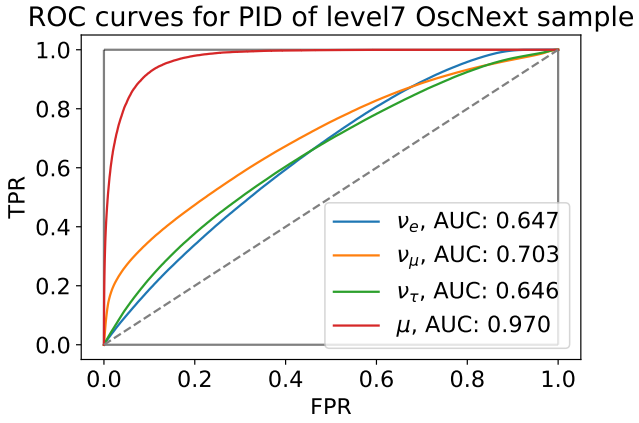

Figure 22: Classification performance for particle type in the OscNext dataset. The performance displayed in a ROC-curve. Particles and anti-particles were not distinguished.

7.1.3 Ensemble Predictions

The results in Sec. 7.1.1 and Sec. 7.1.2 show the individual performance of the models; the models differ a lot and for this reason the reconstructions from the four models as well as the particle classification from GGConv were combined in an ensemble model (as describe in Sec. 6.5) to improve the performance further.

To train the ensemble, the test sample from the OscNext dataset was used. The test data was split into a new train/test split of 80 % / 20 % and the ensemble was trained using MAE[23] and stopped when the performance gain plateaued. The performance of the ensemble is shown in Tab. 4. For a visual comparison of the predicted variables with the truth, refer to Fig **??** in the appendix.

Table 4: Table of performance metrics for the ensemble model, best performance of any GNN and Retro. For energy the width, $w(\Delta \log_{10}(E))$ is used, while the angles are reported as median of absolute difference. The best GNN is marked by the colour of the model with that performance and the best measure in each row is marked by bold.

	Ensemble	Best GNN	Retro
$\log_{10}(E)$	**0.21**	0.21	0.24
θ (deg)	**12.6**	12.9	15.0
ϕ (deg)	**29.6**	29.6	38.0
Ω (deg)	33.4	32.8	40.6

7.2 Muon Reconstruction

Although the focus of IceCube is on neutrinos, muons make up a large fraction of the events detected. This makes them a high priority, since they can both be used for calibration of reconstruction algorithms and to clean a sample of neutrinos from muons. Directional reconstruction is desired in order to calibrate our predictions against the moon shadow.

The performance of the different models is compared in Tab. 5 and in more depth in Figs. 36 and 41 in the Appendix. The first displays the energy dependent performance while the second is an overview of the uncertainty predictions.

Table 5: Comparison of the performance metrics for the three models used for muon reconstruction. Metrics are the same as Tab. 2. AntHill was not used for predicting muon energy and is thus left out.

	θ	ϕ	Ω	$log_{10}(E)$
StateFarm	**1.72** °	**5.49** °	**4.09** °	**0.1340**
GGConv	2.77 °	7.52 °	5.77 °	0.1376
AntHill	1.88 °	5.57 °	4.19 °	-
σ	0.01°	0.02 °	0.01 °	0.0007

[23] periodic for the azimuth predictions

7.2.1 Stopped Muons

As previously mentioned, some muons expend all of their energy and stop within the confines of IceCube.

Combining a GNN-architecture with a single output and a sigmoid activation layer, one can obtain a probability, $p \in (0,1)$ from a graph. Now, this model can be trained with the cross entropy loss function in order to predict the probability p that a muon is stopped in the ice. For readability, the probability p is transformed using $\text{logit}(p) = \log(p/(1-p))$, making it easier to compare probabilities near 0 or 1.

Trying to obtain a stopped-muon classifier with AntHill, it was possible to achieve an AUC-score of 0.927. The ROC-curve of the classification as well as the distributions for the labeled data is displayed in Fig. 23. Even though the two distributions have an overlap, it is possible to make a cut in logit-scores which leaves very few non-stopped muons in the sample. The base fraction of stopped muons is 68.2%, but making a cut at $\text{logit}(p) = 2.0$, the fraction of stopped muons is increased to 98.2%, while only removing 35% of the stopped.[24]

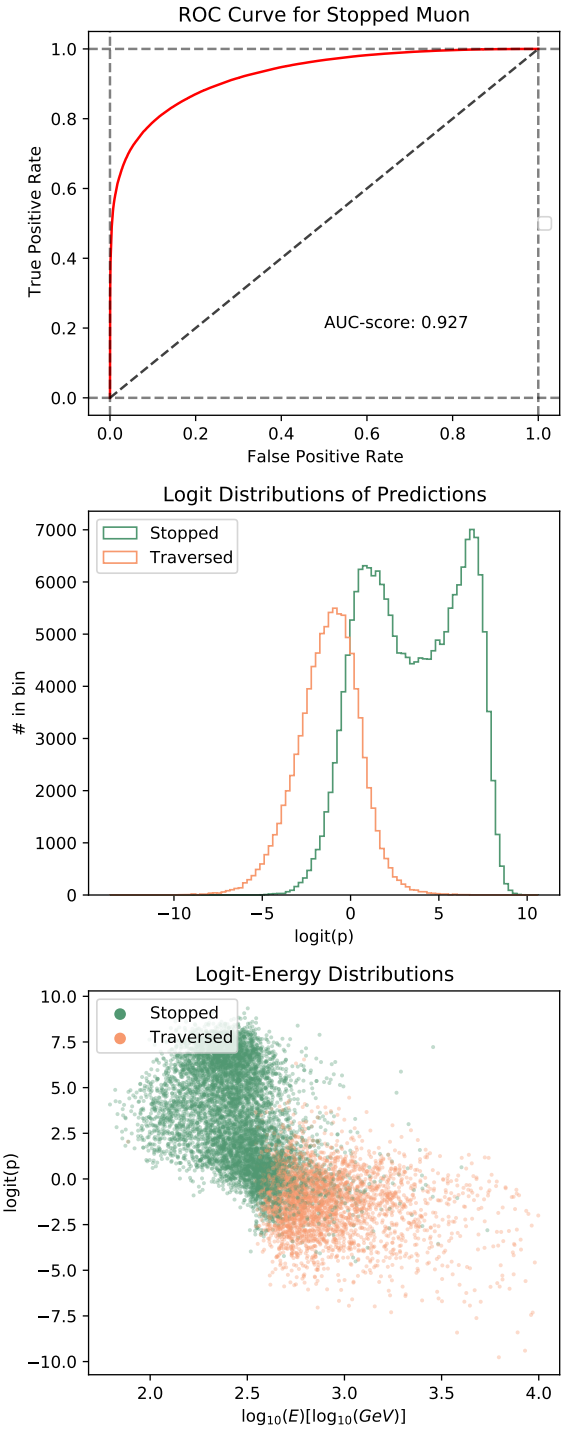

Figure 23: Overview over AntHill performance on stopped muon classification in the MuonGun dataset. The first plot shows the ROC-curve for the classification. The second plot compares the distribution of logit score for stopped and non-stopped muons respectively. The third plot shows how the logit scores are dependent on energy.

7.2.2 Information in SRT Cleaned Pulses

The MuonGun dataset contained a variable telling if a pulse "survived" the SRT cleaning or not (see Sec. 3.6). To test if any information is lost doing the SRT cleaning for muons, AntHill was trained with and without the SRT cleaning for angular reconstruction. The median of the angle differences for total, azimuth, and zenith angles can be seen in Tab. 6, showing a 30% improvement when including all pulses over the cleaned data. The energy dependent performance is displayed in Fig. 42. Note that the uncleaned events perform significantly better in the $2.5 \leq \log_{10}(E) \leq 3.25\,\log_{10}(\text{GeV})$ range, but both sets perform almost identically at high energies.

7.3 Uncertainties from Probabilistic Loss

In this section, the uncertainties resulting from employing the loss functions described in Sec. 4.3.6 will be analyzed to see if they are reliable in describing the accuracy of the model.

To get a sample of accurate events, it is possible to cut in the predicted uncertainty of the models only considering e.g. the 50% most certain events, hopefully improving the accuracy. Furthermore,

[24] In other words, a true positive rate of 0.982 at a false positive rate of 0.35

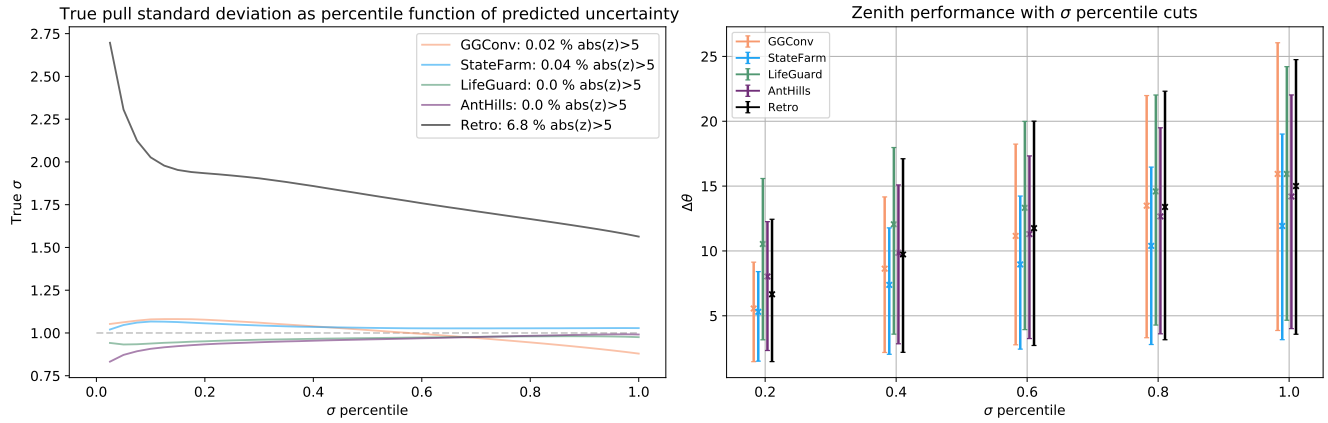

Figure 24: Zenith sigma analysis for our models and Retro. First plot shows the true standard deviation of pulls masked as below percentile of predicted uncertainty. Note that we only consider pulls below 5 since Retro sometimes has pulls of order millions, ruining the mean as a statistic. The percentage of pulls discarded is shown in the legend. Second plot shows the performance (16th, 50th, and 68th percentile) for predictions masked as below certain percentiles of predicted uncertainties.

Table 6: Table of performance metrics for angular differences in the prediction from AntHill on MuonGun data with and without SRT cleaning of events. All values are median of the angular differences given in degrees. For the energy dependent performance, see Fig. 42 in the appendix

[deg]	2D	3D	2D clean	3D clean
θ	**1.88**	2.02	2.60	2.96
ϕ	5.57	**5.38**	8.89	8.37
Ω	4.19	**4.14**	6.52	6.42

as the used probabilistic distributions are supposed to be Gaussian on the surface of the hypersphere, it is beneficial to investigate whether or not the uncertainties actually render Gaussian distributions. In Fig. 24, this is tested by considering events only below a certain percentile in our predicted uncertainty. The metrics described in Sec. 4.3.8 are considered for different uncertainty percentiles. The Gaussianity of the pull distribution is tested by considering the actual standard deviation of the pulls ($z = \frac{y_{true} - y_{pred}}{\sigma_{pred}}$), which should be 1 if the errors are actual Gaussian. The more complete Fig. 40 can be found in the Appendix.

Overall, we see that our models generally perform well and have better uncertainty estimates than Retro. We see that the Gaussianity condition is fulfilled and that cutting in the predicted uncertainties vastly improves precision, e.g. for StateFarm, the median residual is reduced by a factor of 3 going from the full sample to the 20% most certain.

7.4 Comparison of Angular Loss Functions

The choice of loss function is critical in model development. In this section a comparison of the loss functions for directional regression is provided, and their accuracy across the full night sky is evaluated. All loss functions were used to train GGConv. We consider the following (see Sec. 4.3.5 and 4.3.6):

- **MSE+MAE**: The sum of the MSE and MAE, which can be considered the baselines for any regression problem. Only non probabilistic loss is used here.
- **SvM**: The Spherical von Mises Fisher distribution.
- **2xPvM**: The product of 2 Polar von Mises Fisher distributions.
- **SvM/2xPvM**: A combination of SvM and 2xPvM, where for each batch, the loss is computed for both functions but it is randomly chosen which one will dominate the sum of the two losses, by a factor 10^7.[25]
- **vG** A multivariate normal distribution for the vector components. vG predicts the x, y, z components of the directional unit vector and provides an uncertainty for each component. When testing, the azimuth and zenith angle are calculated from the un-normalized vector predictions.

[25] The randomization is 50/50 and the factor is arbitrarily chosen.

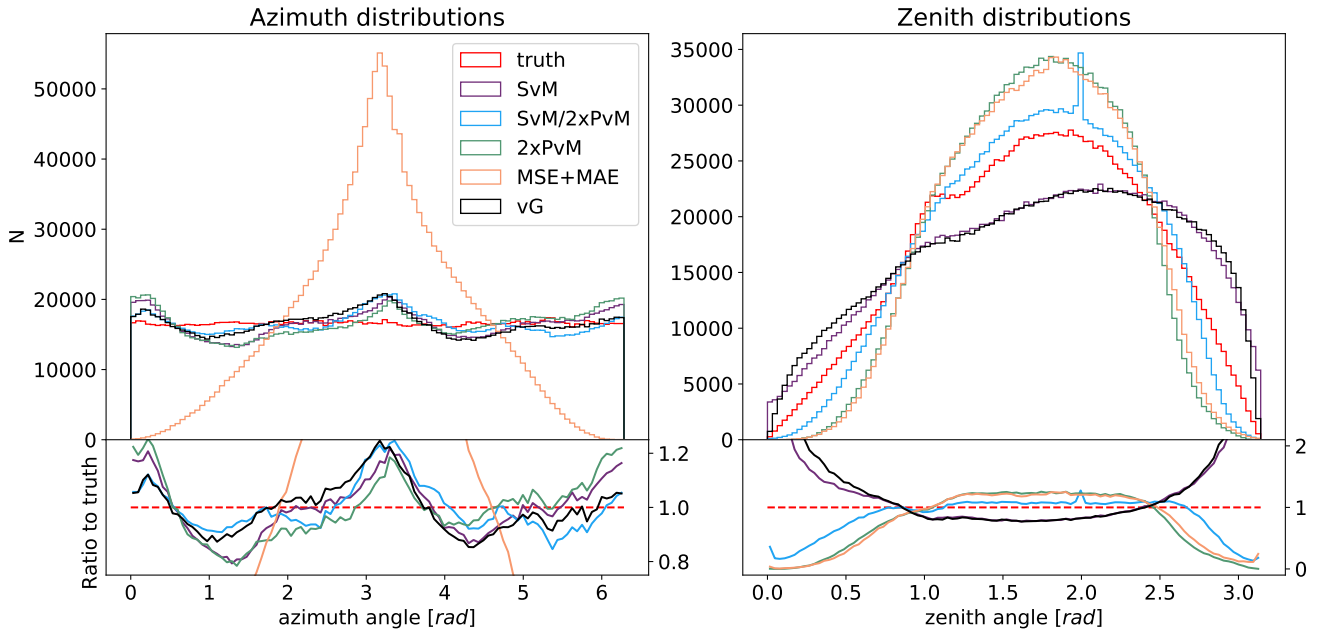

Figure 25: Comparison of the distributions of the azimuth and zenith prediction for each loss function. For a visualization of the full night sky see Fig. 46 in the Appendix.

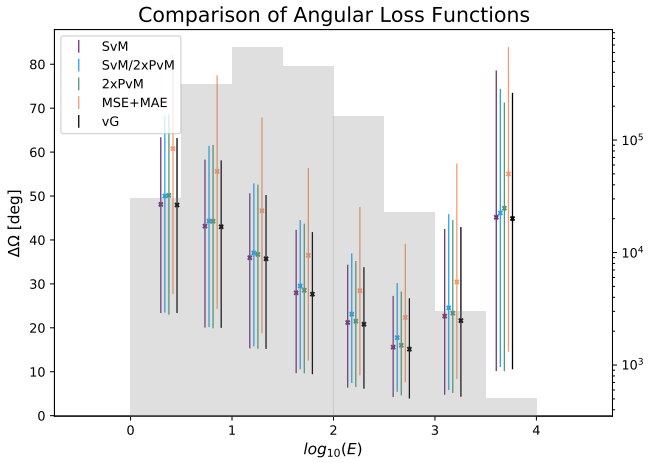

Figure 26: Performance measure of the angle prediction at (16th, 50th, and 68th) percentile for different loss functions. In general, the probabilistic loss functions perform better across the entire energy range, with the SvM/2xPvM being the worst of the four.

A comparison of the performance for the different loss functions for directional regression can be found in Fig. 26. MSE+MAE is clearly lackluster which is due to the functions not being periodic as seen in the distributions of the predictions in Fig. 25 (or the full night sky in the Appendix in Fig 47). SvM and vG provide similar results since they both essentially minimize $\boldsymbol{\mu}^T \boldsymbol{x}$. Both has a bias toward "poles", whereas 2xPvM and MSE+MAE has a bias towards the horizon. This was the motivation for SvM/2xPvM which can be seen to approximate the true distribution better, at a minor cost in directional prediction accuracy.

7.5 Moon Shadow Reconstruction

Since cosmic rays consist of atomic nuclei, they can be blocked by the moon on their path towards the atmosphere. Thus, given a precise enough reconstruction, a deficit of muons can be seen from the direction of the moon. This renders the moon shadow a useful calibration source, since it is possible to quantify the precision of a given reconstruction algorithm in data with a pseudo-label. The full analysis thus requires a gathering of many such events for which the moon is visible. It is also necessary to apply a translation with regards to the moon position. This is feasible, but not encompassed in this project. Instead, a toy dataset is simulated where the appertaining reconstructions are based on our model performance. There is evidence to suggest that this approach is acceptable, since Retro has already performed the moon analysis, and our models maintain the correlation with Retro in real data as can be seen in Fig. 27. For a discussion of the used real observations see the Appendix Sec. 12.9.

7.5.1 Simulating the Dataset

Creating the toy dataset consists of generating a true distribution and applying deviations gener-

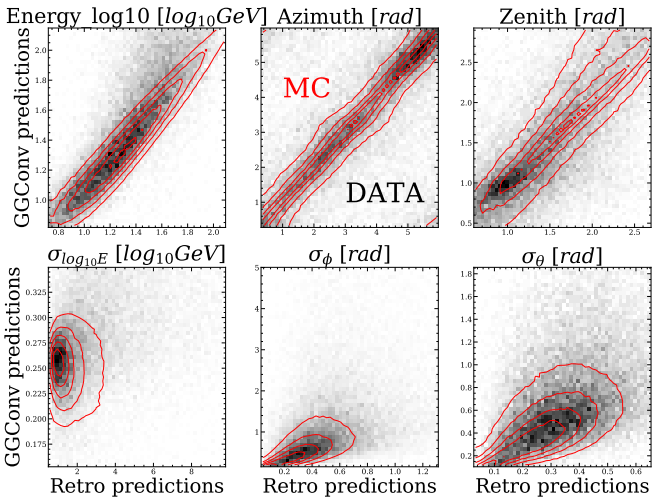

Figure 27: Applicability of the Monte Carlo trained ML methods in real data. Shown are the contour plots of the correlation between predictions made by GGConv and Retro for energy, θ, ϕ, and their uncertainties in MC. The underlying heatmap contains the same but in real data. The data sample size is significantly smaller. The interval of the axes are chosen to include the inner 90% of the sample and are equal for MC and data. A bigger version can be found in the Appendix in Fig. 47.

ated from the model residual distribution. The points are originally ascribed an azimuthal and zenith angle on the sky, taking as our average fluence 312 events per square degree. The origin is set to be the moon position. Points within a radius of 0.5° are then excluded, as to simulate the moon radius. This is slightly contrary to the true size of the moon which has a radius that varies between 0.245° and 0.284° [55]. This factor of 2 is arbitrary and unimportant for the analysis, but was included for ease of testing. The remaining points are then translated by an angle, which is sampled from the model residuals. The points are also ascribed the associated predicted uncertainty. The direction of the translations are distributed uniformly.

In practice, the spherical geometry of the sky is easily accounted for when translating the events with regards to the moon position. However, if the translation is already done and the considered field of view has sidelengths of $< 10°$, it is simplest to assume the geometry in question is Euclidean. This convention is adopted from here on.

These "observations" immediately render themselves susceptible to an unbinned Likelihood fit. However, in order to account for the uncertainty of each point, a generalization of the maximum likelihood method is needed. A sketch of a derivation is attempted in the following section.

7.5.2 Unbinned Likelihood on Data with Uncertainties.

Maximum Likelihood Estimation describes the method of adjusting the parameters of a model such that the likelihood of data being from a given model is maximized. The method does not necessarily describe how well the model fits the data, but given a specific model it determines a good estimate of what the parameters should be.

Usually it is certain what observations have been made, and the likelihood can be computed by multiplying the probabilities of each event under the assumption that they were sampled from the model:

$$L(\boldsymbol{\theta}|\mathbf{x}) = \prod_i^N P(x_i|\boldsymbol{\theta}) \tag{39}$$

However, it can be advantageous to generalize the framework for uncertain data. Instead of multiplying over the data, Eq. 39 can instead be written as multiplying over the entire domain of possible observations, Ω_x, and exponentiate the probability by how many times any observation occurred, denoted N_x. In that case the likelihood is:

$$L(\boldsymbol{\theta}|\mathbf{x}) = \prod_{x\in\Omega_x} P(x|\boldsymbol{\theta})^{N_x} \tag{40}$$

But Eq. 40 is not practical since there is no reason to check the entire domain.[26] However, one can now consider the case, where the observations are uncertain and are instead described by some set of parameters θ_x (eg. mean, variance, and so on), such that the observation x_i is described by its Probability Density Function $x_i \sim PDF(x|\theta_{x_i})$.
The likelihood can then be extended to the following:

$$L(\boldsymbol{\theta}|\mathbf{x}) = \prod_{\theta_x\in\Omega_{\theta_x}} P(x|\boldsymbol{\theta},\theta_x)^{N_{\theta_x} P(x|\Omega_x,\theta_x)} \tag{41}$$

The product is now over all possible data parameters Ω_θ. $P(x|\boldsymbol{\theta},\theta_x)$ is the probability of observing x given model parameters $\boldsymbol{\theta}$ ***and*** the data parameters θ_x. N_θ is the number of observations in the dataset with θ_x as parameters[27] and $P(x|\Omega_x,\theta_x)$ is

[26] This is not even feasible for a continuous domain Ω_x
[27] For the continuous case this is practically always 1

the probability of observing x in the domain given θ_x
Now it is more appropriate to revert back to considering a product over the dataset instead of all the possible configurations Ω_{θ_x}.

$$L(\boldsymbol{\theta}|\mathbf{x}) = \prod_i^N P(x|\boldsymbol{\theta}, \theta_{x_i})^{P(x|\Omega_x, \theta_{x_i})} \tag{42}$$

Comparing Eq. 42 to Eq. 39, we see that now the data $\boldsymbol{x}$ does not consist of absolute observations x_is but of the θ_{x_i}s describing the PDFs of the observations. Now, the probability of an observation being in Ω_x is not 0 or 1 but between 0 and 1. The interpretation is that $P(x|\Omega_x, \theta_{x_i})$ is a weight factor, where a lower probability means lower consideration is made for that point when adjusting the model parameters.

Further clarification is given by expressing the probabilities in Eq. 42 in terms of the model $P(x|\theta)$ and the PDF of the observations $PDF(x|\theta_{x_i})$.

$$\begin{aligned} P(x|\boldsymbol{\theta}, \theta_{x_i}) &= \textstyle\int_{\Omega_x} P(x|\theta) PDF(x|\theta_{x_i}) dx \\ P(x|\Omega_x, \theta_{x_i}) &= \textstyle\int_{\Omega_x} PDF(x|\theta_{x_i}) dx \end{aligned} \tag{43}$$

If instead the domain is discrete, the integrals become sums since $PDF(x|\theta_{x_i})dx \to PMF(x|\theta_{x_i})$ where PMF is the Probability Mass Function for x_i.

7.5.3 Application on Moon Toy Dataset.

In this analysis we assume the observations to be normally distributed with mean μ_i and variance σ_i^2

$$PDF(x, y|\theta_{\boldsymbol{x}_i}) = \mathcal{N}(x, y|\mu_i, \sigma_i)$$

μ_i and σ_i are predicted by a GNN model.
The moon model is then given by:

$$P(x, y|\theta) = \frac{1}{N}\left(1 - fe^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}}\right) \tag{44}$$

Closed form expressions can be found for both the normalization constant N in Eq. 44 and the integrals in Eq. 43.[28]

The free parameters of the model are the placement μ_x, μ_y, the width σ and the depth f. The minimization is carried out with iminuit [56] and yields the results shown in Fig. 28. As can be seen,

[28] These can be found in Jonas Vinther's GitHub

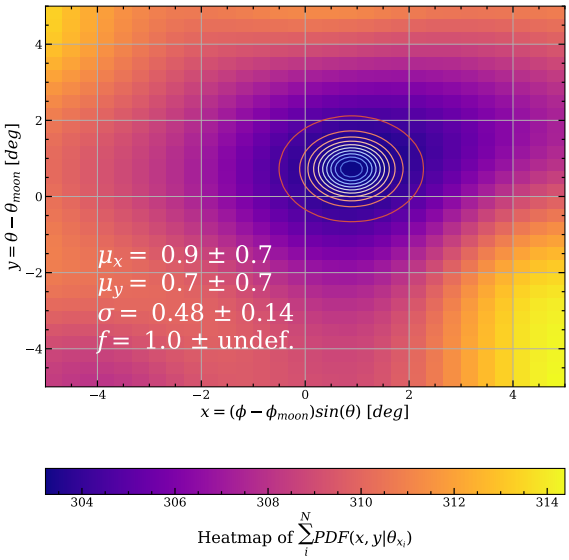

Figure 28: The results of the unbinned likelihood fit which accounts for uncertainties of the data points. The heatmap is the sum of the PDFs generated by the positions of the points and their uncertainties. The uncertainty on f is undefined because the value reached the end of its interval $f \in [0, 1]$. For a comparison to the no moon shadow case, see Fig. 38 in the Appendix.

all the true parameters are well within the errorbars of the estimates. Also notably, the ratio of the shadow to the background, as estimated from the heatmap, is around $2 - 3\%$. However, an f of 1 is estimated by the minimization. This can be attributed to the fit accounting for the uncertainty of each point, since no point had its entire PDF under the moon shadow. This results in no error estimate for the parameter f and allows us to remove it as a free parameter and fix $f = 1$, in contrast to the standard unbinned likelihood method, for which the results are shown in Fig. 29.

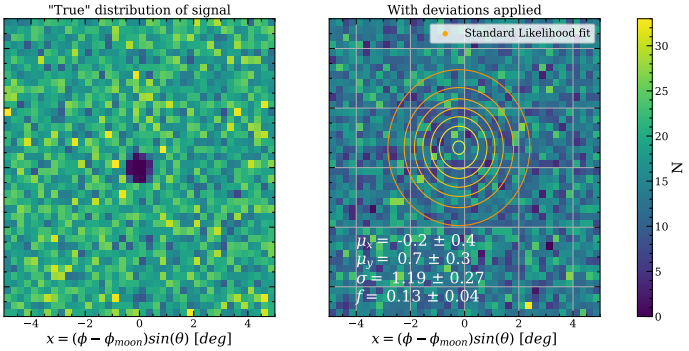

Figure 29: Prior distribution and observed distribution. The results of the standard unbinned likelihood fit can be seen, but for a clarification of why the shadow is placed as it is, see Fig. 37 in the Appendix. The same analysis, but with artificially better accuracy, can be found in Fig. 39 in the Appendix.

Both the distance from the origin and the width is off by about $\sim$ 2.4 and 2.6 sigma respectively, which are both barely acceptable deviations.

To compare the two methods, a hypothesis test is carried out. **Wilk's Theorem** states that given an alternative hypothesis which has likelihood $\mathcal{L}_{alt}$ and has the null hypothesis nested, the test statistic $D = 2ln\left(\frac{\mathcal{L}_{alt}}{\mathcal{L}_{null}}\right)$ is approximately χ^2-distributed with degree of freedom $df_{alt} - df_{null}$. It is not given that D follows a χ^2 - distribution, and often it is advantageous to carry out a simulation to estimate a distribution for D. However, here Wilk's theorem is assumed to be applicable. The null hypothesis is then: $f = 0$ with $df_{null} = 0$. For the method that accounts for uncertainties the alternative hypothesis is the result of the minimization with $f = 1$ such that $df_{alt} = 3$. It is the same for the standard method but with $df_{alt} = 4$ since f is still a free parameter. The resulting p-values are 2.6% and 0.033%, respectively, indicate that both are significant, but with a notably higher significance from the standard method.

Eq. 43 can thus be said to better estimate the parameters of the true distribution prior to the added noise and give more conservative p-values in accordance with the rather low sample size and the introduction of uncertainty.

8 Discussion

Throughout this thesis, GNNs were tried on different aspects of the IceCube experiment. In the following section the usefulness and performance of this framework will be discussed, along with proposed solutions to some of the problems presented in the previous sections.

8.1 Graph Neural Networks

As a part of this project, GNNs have successfully been employed and optimized to work with data from the IceCube Experiment, and we have gained several insights during this process. In this segment some of the most interesting outcomes will be highlighted.

Many points are relevant to most NN types, but some are specific to GNNs.

8.1.1 Problems with Graph Neural Networks

Regressing large samples of data in order to adjust the weights and biases is very effective, and provides a data-driven way of predicting features of new data. However, since the networks aim to minimize the average loss, they will often tend to simply regress to the mean of population-wide statistics.

Subjective choices of hyperparameters means that reaching the global minimum is by no means guaranteed, even if the UAT holds. This means that countless experiments have to be made to get an estimate of the actual best performance of any model. For example Fig. 19 took 210 GPU hours to complete.

Lack of explainability means that GNNs are used as black box algorithms, which can make it hard to draw conclusions from or have confidence in the inner workings of a GNN, even if it is very precise.

Possible lack of extrapolation ability is a problem that is present in any NN model, since the learning method is a basic "monkey see, monkey do" framework. If any truly spectacular and extraordinary events happen, it is unlikely that a Deep Learning approach will catch it.

Labeled data is required to do supervised learning, but not always available and any model thus relies heavily on correct simulation.

Speed for GNNs is subpar compared to other NN approaches with more rigid data structures.

Global features can be hard to capture efficiently, since GNNs by nature are constructed to focus on neighbourhoods. One solution to this is to simply extend the neighbourhood to include every point, but for large graphs this quickly becomes computationally infeasible and leads to oversmoothing, since a node considering all its neighbours equally will risk not gathering relevant information from the nodes that are most relevant. This can be mitigated by using attention networks but this has not been as successful as we had hoped. One ad-hoc work-around that has worked very well was to simply gather graph-level information, like the mean, variance, min, and max of each feature across all nodes, and feeding it directly to the decoder after the graph encoding. This consistently improved performance.

8.1.2 Advantages with Graph Neural Networks

Highly flexible inputs is one of the principal advantages of the GNNs. One of the main motivations behind choosing this framework is that GNNs work on any node structure. This means that there is no reason to pad or select sub samples of the DOM pulses from an event and all the data can thus be utilized.

Speed for GNNs is still of order 10^5 faster than the currently employed classical algorithms, even if they are slower than other NN based algorithms.

Edge features allow graphs to encode relations between data points more readily than other frameworks.

Theoretical Versatility As can be seen in the very different custom layer and model implementations, the GNN framework is extremely versatile, and can be combined with already existing Deep Leaning frameworks.

Practical Versatility As IceCube will soon see the implementation of the new DOMs in Upgrade, new algorithms will need to be implemented, taking into account the somewhat peculiar new structure. Given a reasonably accurate simulation of the new sensor array, the models proposed in this project (and an ensemble) will most likely be able to begin accurate and reliable reconstructions within hours. This would be the same for any other upgrade.

We are only getting started. GNN research is still getting started, but is already the most popular topic at e.g. the ICLR conference.

8.2 Neutrinos

For the zenith angle, the general distribution was missed by quite a bit, which was mainly due to the loss function tending to either regress uncertain points towards the poles or towards the middle. This could be fixed by sacrificing some precision and implementing a correcting term in the loss function or to do a combination of e.g. the polar and spherical likelihood functions, since the spherical likelihood pulls towards the poles and the polar likelihood towards the average zenith angle (SvM / 2xPvM, see Fig. 26). The KDE loss (Sec. 4.3.7) can correct the distribution if the training batch size is above a few thousand, but for single models sub par precision cannot be avoided

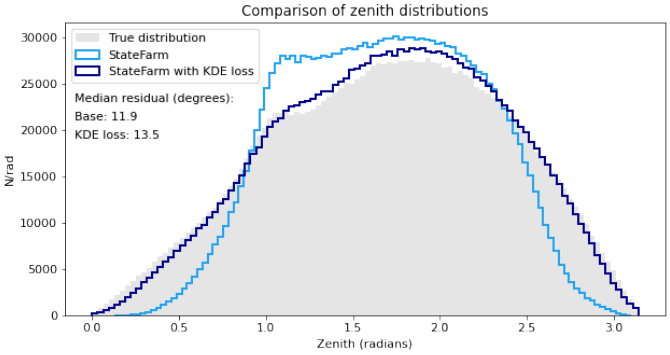

Figure 30: StateFarm polar likelihood misses the neutrino zenith distribution, but amending it with a KDE loss term reduces the precision significantly.

as can be seen in Fig. 30. Notably, Retro hits the target distribution well, but with lower accuracy compared to the 2xPvM+KDE-trained StateFarm.

As described in Sec. 2, the neutral and charged current interactions are quite different and we expected to see better performance for CC interactions. We also expected muon neutrinos to be easier to be regress, since they create track-like events and we have a larger training sample of muon neutrinos. As can be seen in Tab. 3, these expectations were correct. StateFarm performed slightly better on tau than electron neutrinos.

The model ensemble also models the target distributions better, and additionally improve upon the accuracy.

8.2.1 Classification Tasks

When testing the particle classification in the OscNext dataset with GGConv, the muons were easily separated from the neutrinos. However, it is important to keep in mind, that the two types of events come from different simulations and this could be the reason for the performance. If the performance were to translate to data, this classification algorithm would significantly reduce the muons (in this case noise) from a neutrino sample.

The neutrinos were harder to separate (AUC was best for muon-neutrinos with 0.703), and this capability would have to be improved in order to recreate the oscillation plot seen in Fig 2. Even though the classification was not perfect the information could still be used in the ensemble to improve the knowledge of the particle type when reconstructing energies and angles.

8.2.2 Comparison with Retro

The speed improvement is mainly due to the nature of the algorithm, where we employ a series of matrix multiplications instead of a look-up table. The speeds quoted in table 2 are subject to implementation differences, and thus should all be able to achieve speeds of $5 \cdot 10^4 - 10^5$. In general our predictions are more accurate and our uncertainties better when compared to Retro (Figs. 24 and 40). It can be discussed whether or not the uncertainties from the probabilistic loss functions are Gaussian by construction or inherently Gaussian. Other uncertainty measures could be more appropriate.

Although we show significant improvements compared to Retro, we postulate progress can still be made, although we have seen some evidence of hitting the information limit:

- Different architectures all achieve similar accuracies (albeit with different strengths and weaknesses).
- Vastly larger networks gave no notable improvement.

However, the ensemble was a quick implementation that improved the overall precision, which has led us to believe that there are still improvements to be made.

8.3 Muons

Compared with the performance on the MuonGun dataset, it becomes evident that a much better angular precision can be reached. For example, half the muons reconstructed with AntHill, attain an angular residual of 4.2° while it is 36.4° for OscNext. The improved accuracy is credited to the muons being a charged particle, which are observed at higher energies and directly instead of through their decay products.

Furthermore, the MuonGun dataset came with the pulses which were not removed by the SRT-cleaning. Including this information in the data improved the median accuracy with a factor $\approx \frac{2}{3}$. This demonstrates the versatility of the GNN architecture, where more information can simply be included in the algorithms in exchange for slightly longer train time. Unfortunately, the OscNext sample was already SRT-cleaned, which removed the option for trying the same for Neutrinos.

The GNN was capable of finding a clean sample of muons stopped in the detector. However, this is not surprising when comparing the energy distribution with the predicted probability, demonstrating that low energy muons are often stopped within the detector (Fig. 23). For this reason a good energy prediction would be a good indicator that a model can separate the stopped muons. Due to the ensemble performance gain, we expect an ensemble to perform well when classifying stopped muons.

8.3.1 Moon Shadow Reconstruction

The developed likelihood method, which accounts for the uncertainties is useful for not only examining the predictive power of a reconstruction but also the correctness of uncertainties. While one could do the moon analysis for different bins of σ, one can now instead keep the whole sample and see how well the estimated parameters align with the expected values, since this method seemingly is able to estimate the true distribution given that the uncertainties are correct. However, for more conclusive and quantifiable results a more thorough analysis is necessary.

8.4 Interpretability

We compared the Integrated Gradients (IG) and SHAP methods of explaining events, and found no correlation between the two.[29] We compared these across GGConv and StateFarm. For StateFarm, using IG, the DOM positions were most important, with the z position being of particular importance (lower plot in Fig. 31). For GGConv, using SHAP, the DOM positions were less important, while the width parameter became by far the most important (upper plot in Fig. 31).

These two explanations seem contradictory but could reflect the two models working very differently, although this seems somewhat dubious, considering that they have similar performance which we believe is close to the information limit. Further

[29] Taking into account the computationally heavy SHAP method, only 100 random events were compared

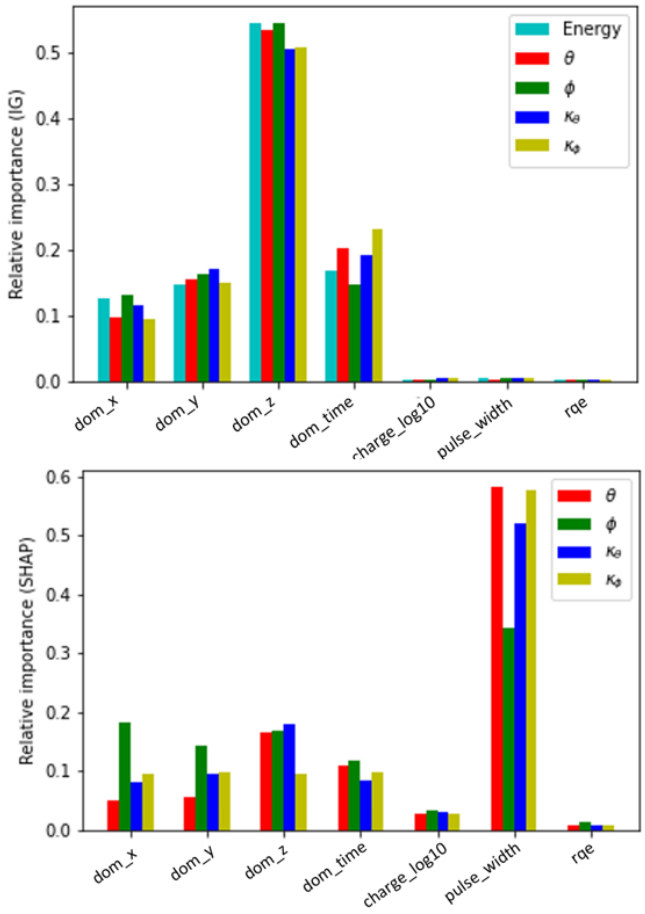

Figure 31: Average relative importance for output variables with respect to input variables for SHAP for the upper plot and Integrated Gradient for the lower.

work must be done to explain the networks. The correlation plots can be found in Appendix 12.7.

9 Conclusion

In this project, Graph Neural Networks were implemented to do reconstruction of low energy particles in the IceCube Experiment. Four different architectures were proposed by the authors, which all possess similar or better reconstruction capabilities compared to the current *Retro* algorithm. The best models improved accuracy over Retro by $\approx$ 14% reduction in the median angular difference for zenith, azimuth, and total angle, while the width of the energy residual distribution was reduced by $\approx$ 13%. When combined in an ensemble, the reconstruction was improved further to $\approx$ 16% for the zenith metric. Furthermore, the models provided well behaved uncertainty estimates when trained with probabilistic loss functions. The authors hope that this will facilitate the usage of low energy neutrinos at IceCube.

For muons, the reconstruction proved very accurate with a median absolute zenith residual $\approx 2^\circ$ and median absolute angular residual $\approx 4^\circ$, which according to our analysis should be enough to see the moon in data, allowing for physical calibration. When applying the GNN framework to classify if muons stopped within the detector, a very pure sample could be achieved containing 98% stopped muons while only reducing the amount of data by 35%. The AUC for this classification is 0.927. In addition, muons could easily be classified from neutrinos with an AUC of 0.970. However, classifying the flavour of neutrinos proved more difficult and AUC scores around a mere 0.67 were achieved.

We have demonstrated that our GNN architectures have multiple other advantages. First, the computation time compared to Retro is faster by five orders of magnitude. Furthermore, the GNN framework is very flexible and can easily be adjusted to new IceCube upgrades with new parameters, even if the DOMs are placed weirdly or have other properties than the existing DOMs, provided simulations can accurately capture these aspects.

We have also shown that GNNs handle noisy data well, actually performing better when noise was passed along, so the many levels of cleaning could no longer be relevant.

10 Further Work

Explainability could be improved since our first attempts pointed in very different directions.

Speed could be improved since the biggest bottleneck in reconstruction speed is data-loading. Having access to the raw data and time to implement it better, we hypothesise the speed of regression could be greatly optimized.

Creating an ensemble model, the overall performance improved compared to a single GNN. This process was however started late in the project since it required all group members to have predictions on the test set. For this reason the ensemble is not well-optimized and the idea as a whole has a lot more potential. It would be interesting to combine the idea of explainability and SHAP values with the simpler ensemble model. This could lead to nuanced comparisons of the models.

The idea of doing GNNs and ensembles could also be expanded to encompass the entire IceCube framework. Throughout this thesis GNNs have proved useful in both classification and reconstruction, where even DOMs previously deemed noise helped accuracy. One could imagine having a bigger collection of GNNs doing everything from determining the trigger to cleaning and reconstructing the direction, energy, and particle type. We hypothesize that this could keep up with the events in IceCube in real-time.

11 Acknowledgements

We would like to thank:
Rasmus F. Ørsøe for extracting the data from the raw i3 files, as well as for his inspiration and camaraderie.
Bjørn Mølvig and Mads Ehrhorn for providing fantastic master theses as inspiration and a general data framework.
Professor Troels C. Petersen for supervising us, and getting us in touch with so many great people.
Finally, to our families, girlfriends, and friends, for supporting us throughout the experimentation and writing process, allowing us to talk about whimsical particles, frustrating code, and silly ideas.

References

[1] Christoph Scholz Frank Zetsche Bogdan Povh, Klaus Rith and Werner Rodejohann. "Particles and Nuclei – An Introduction to the Physical Concepts". 7th edition edition, 2015.

[2] Raymond Davis Jr. Nobel lecture: A half-century with solar neutrinos. Reviews of Modern Physics, 75(3):985, 2003.

[3] Z. Maki, M. Nakagawa, and S. Sakata. Remarks on the Unified Model of Elementary Particles. Progress of Theoretical Physics, 28(5):870–880, November 1962.

[4] Markus Ahlers. Lecture slides - introduction to nuclear and particle physics - nbi, university of copenhagen, 2020.

[5] The website for the NBI IceCube Collaboration. 2016. https://nbi.ku.dk/english/research/experimental-particle-physics/icecube/neutrino-oscillation/.

[6] The official website for the IceCube Collaboration. 2016. https://icecube.wisc.edu/.

[7] Benjamin Cheymol. Development of beam transverse profi le and emittance monitors for the cern linac4. 12 2011.

[8] CA Argüelles, AJ Aurisano, B Batell, J Berger, M Bishai, T Boschi, N Byrnes, A Chatterjee, A Chodos, T Coan, et al. New opportunities at the next-generation neutrino experiments i: Bsm neutrino physics and dark matter. Reports on Progress in Physics, 83(12):124201, 2020.

[9] Carlotta Giusti, Andrea Meucci, and Franco Davide Pacati. Charged-current and neutral-current neutrino-nucleus scattering in a relativistic approach, 2009.

[10] Hadiseh Alaeian. An introduction to cherenkov radiation. http://large.stanford.edu/courses/2014/ph241/alaeian2/, 2014, Accesed 04/06/2021.

[11] M.G. Aartsen, M. Ackermann, J. Adams, J.A. Aguilar, M. Ahlers, M. Ahrens, D. Altmann, K. Andeen, T. Anderson, I. Ansseau, and et al. The icecube neutrino observatory: instrumentation and online systems. Journal of Instrumentation, 12(03):P03012–P03012, Mar 2017.

[12] Delia Tosi and Kyle Jero. IceTop as Veto for IceCube. PoS, ICRC2015:1086, 2016.

[13] Learned through private conversation with our supervisor.

[14] N. E. Bramall, R. C. Bay, K. Woschnagg, R. A. Rohde, and P. B. Price. A deep high-resolution optical log of dust, ash, and stratigraphy in south pole glacial ice. Geophysical Research Letters, 32(21), 2005.

[15] Official icecube diagrams. https://icecube.wisc.edu/gallery/diagrams/. Accessed: 2021/13/06.

[16] R. Abbasi, M. Ackermann, J. Adams, M. Ahlers, J. Ahrens, K. Andeen, J. Auffenberg, X. Bai, M. Baker, S.W. Barwick, and et al. The icecube data acquisition system: Signal capture, digitization, and timestamping. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 601(3):294–316, Apr 2009.

[17] Doms - university of wisconsin-madison physical sciences lab. http://www.psl.wisc.edu/services/electrical/dom, 2018, Accessed 09/06/2021.

[18] The Ice Cube Collaboration. The design and performance of icecube deepcore. Astroparticle Physics, 35(10):615–624, 2012.

[19] Aya Ishihara. The icecube upgrade – design and science goals, 2019.

[20] M.G Aartsen, M. Ackermann, J. Adams, Juan Antonio Aguilar Sánchez, M. Ahlers, M. Ahrens, David Altmann, Karen Andeen, Travi Anderson, I. Ansseau, G. Anton, Carlos Arguelles Delgado, J. Auffenberg, S. Axani, Pascal Backes, H. Bagherpour, Xiaoqiong Bai, A. Barbano, and J.P Barron. Measurement of atmospheric tau neutrino appearance with icecube deepcore. Physical Review D, 99, 02 2019.

[21] Phillip Eller for the IceCube project. An introduction to cherenkov radiation. Private conversation with Troels trough Zoom.

[22] M. Aartsen, M. Ackermann, J. Adams, Juan Antonio Aguilar Sánchez, M. Ahlers, M. Ahrens, David Altmann, Travi Anderson, C. Arguelles, Timothy Arlen, J. Auffenberg, Xiaoqiong Bai, S. Barwick, V. Baum, Ryan Bay, J. Beatty, J. Tjus, K.-H Becker, S. BenZvi, and M. Zoll. Atmospheric and astrophysical neutrinos above 1 tev interacting in icecube. Physical Review D, 91:022001, 01 2015.

[23] D. Heck, J. Knapp, J. N. Capdevielle, G. Schatz, and T. Thouw. CORSIKA: A Monte Carlo code to simulate extensive air showers. 2 1998.

[24] C. Andreopoulos et al. The GENIE Neutrino Monte Carlo Generator. Nucl. Instrum. Meth. A, 614:87–104, 2010.

[25] Bjørn H. Mølvig. Low-Energy Neutrino Reconstructions using Recurrent Neural Networks. Masters Thesis at NBI - University of Copenhagen, 2020.

[26] Julien Aublin, Giulia Illuminati, and Sergio Navas. Searches for point-like sources of cosmic neutrinos with 11 years of antares data, 2019.

[27] Marek Kowalski. Neutrino astronomy with icecube and beyond. Journal of Physics: Conference Series, 888:012007, 09 2017.

[28] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2020.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[32] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[33] Tensorflow batch normalization documentaion. https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization. Accessed: 2021/13/06.

[34] R.J. Barlow. Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences. Manchester Physics Series. Wiley, 1993.

[35] KV Mardia and Peter Edmund Jupp. Directional Statistics. John Wiley and Sons, United States, 2000.

[36] Hexin Bai, Peng Chu, Jeng-Yuan Tsai, Nathan Wilson, Xiaofeng Qian, Qimin Yan, and Haibin Ling. Graph neural network for hamiltonian-based material property prediction, 2020.

[37] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral, 2020.

[38] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[39] Eric W. Weisstein. "Adjacency Matrix." - MathWorld. A Wolfram Web Resource, 2016. https://mathworld.wolfram.com/AdjacencyMatrix.html.

[40] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. CoRR, abs/1704.01212, 2017.

[41] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. CoRR, abs/1609.02907, 2016.

[42] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. CoRR, abs/1706.02216, 2017.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[45] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017.

[46] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.

[47] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 4765–4774. Curran Associates, Inc., 2017.

[48] Harold William Kuhn and Albert William Tucker. Contributions to the Theory of Games, volume 2. Princeton University Press, 1953.

[49] Javier Castro, Daniel Gómez, and Juan Tejada. Polynomial calculation of the shapley value based on sampling. Computers Operations Research, 36(5):1726–1730, 2009. Selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X).

[50] E. trumbelj and I. Kononenko. An efficient explanation of individual classifications using game theory. J. Mach. Learn. Res., 11:1–18, 2010.

[51] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[52] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017.

[53] Marie Jean Antoine Nicolas Caritat marquis de Condorcet. Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. L'imprimerie royale, 1785.

[54] Lior Rokach. Ensemble-based classifiers. Artif. Intell. Rev., 33:1–39, 02 2010.

[55] Saskia Philippen. Observations of the Moon Shadow in Cosmic-Ray-Induced Muons with the IceCube Neutrino Observatory. Masters Thesis at 'III. Physikalischen Institut B' - RWTH Aachen University, 2019.

[56] F. James and M. Roos. Minuit: A System for Function Minimization and Analysis of the Parameter Errors and Correlations. Comput. Phys. Commun., 10:343–367, 1975.

[57] J. Olivares, Pablo Martín, and E. Valero. A simple approximation for the modified bessel function of zero order i 0 (x). Journal of Physics: Conference Series, 1043:012003, 06 2018.

[58] Rasmus F. Ørsøe. A Graph Neural Network Approach to Low Energy Event Reconstruction in IceCube Neutrino Observatory. Masters Thesis at NBI - University of Copenhagen, 2021.

[59] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling, 2019.

12 Appendix

12.1 Notes on Numerical Implementations

12.1.1 Implementing the 3 dimensional vMF

It is advantageous to extend the domain of the 3-dimensional von Mises distribution to $\pm\infty$, since the network otherwise won't account for the bounded domain. For ϕ it is already given since the range of ϕ is ordinarily $[0, 2\pi]$. For extending θ to $\pm\infty$, it is necessary to take the absolute value of the sine functions where θ is input, since the range of θ ordinarily is $[0, \pi]$. Including some trigonometric identities the cosine of the angle is then:

$$\begin{aligned} cos(\Omega) = &\frac{1}{2}[cos(\theta_x - \theta_\mu) + cos(\theta_x + \theta_\mu)] \\ &+\frac{1}{2}|sin(\theta_\mu)|[sin(\theta_x + \phi_x - \phi_\mu) + sin(\theta_x - \phi_x + \phi_\mu)] \end{aligned}$$

The negative log likelihood can then be written as:

$$-ln(vMF) = -\kappa cos(\Omega) - ln(\kappa) + \kappa + ln(1 - e^{-2\kappa})$$

Where sinh is rewritten as $sinh(\kappa) = \frac{1}{2}e^{\kappa}(1 - e^{-2\kappa})$ in order to mitigate the numerical instability of $ln(sinh(\kappa))$ for large κ, and where an irrelevant constant is left out.

12.1.2 Implementing the 2 dimensional vMF

In the current implementation the gradient of I_0 is not defined, so as of yet, a close approximation of I_0 is used [57]:

$$I_0(\kappa) \approx cosh(\kappa)\frac{1 + 0.24273\kappa^2}{(1 + \frac{1}{4}\kappa^2)^{\frac{1}{4}}(1 + 0.43023\kappa^2)}$$

Again cosh should be rewritten as $cosh(\kappa) = \frac{1}{2}e^{\kappa}(1 + e^{-2\kappa})$ in order to mitigate the numerical instability of $ln(cosh(\kappa))$

In order to extend the domain of $cos(\Omega_\theta)$ for the zenith angle θ to $\pm\infty$ it's sufficient to take the absolute value of θ_μ such that $cos(\Omega_\theta) = cos(\theta_x - |\theta_\mu|)$

12.2 Model Architectures

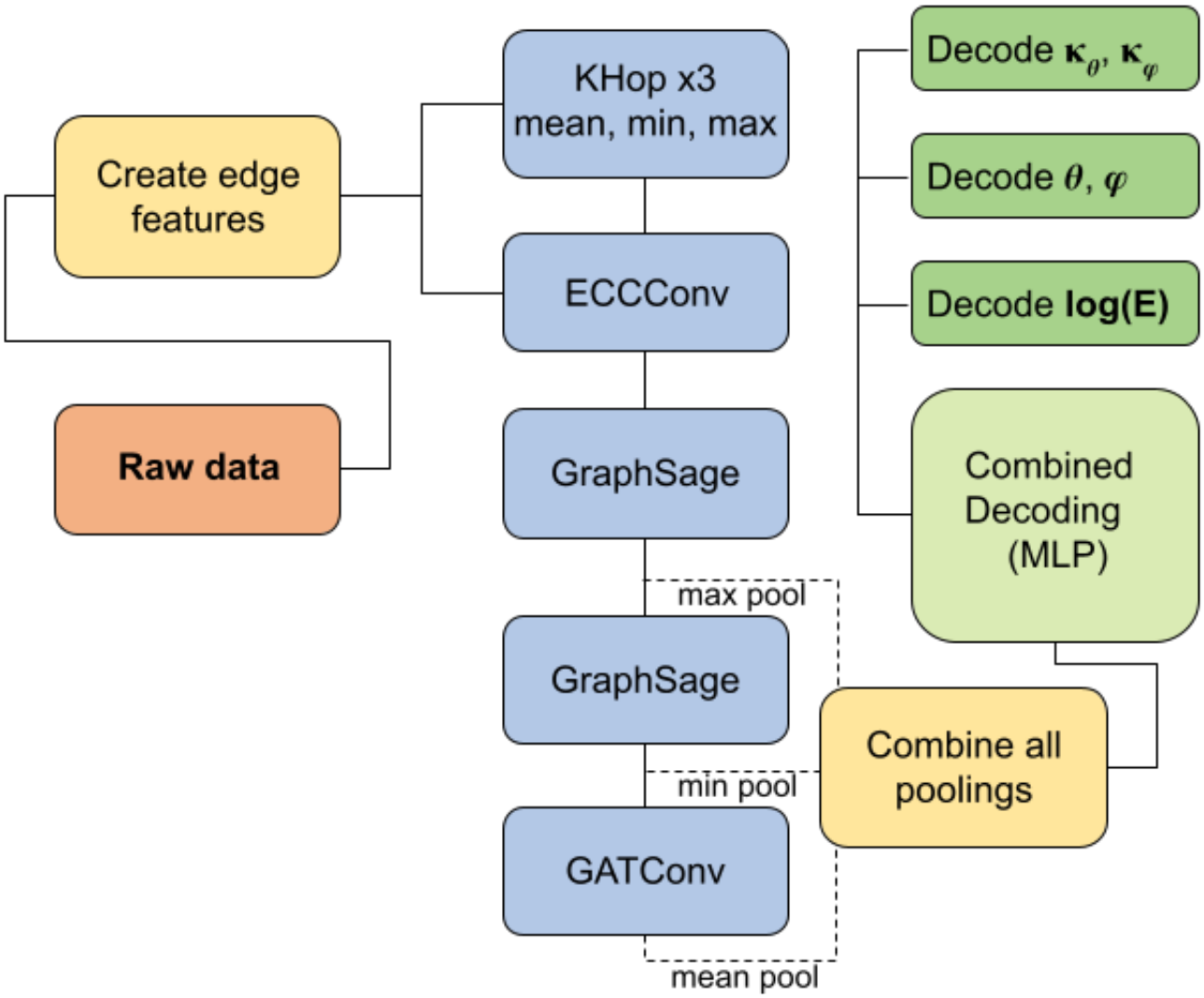

Figure 32: This is what LifeGuard ended up looking like. Red is input, dark green is output. Dotted lines are poolings.

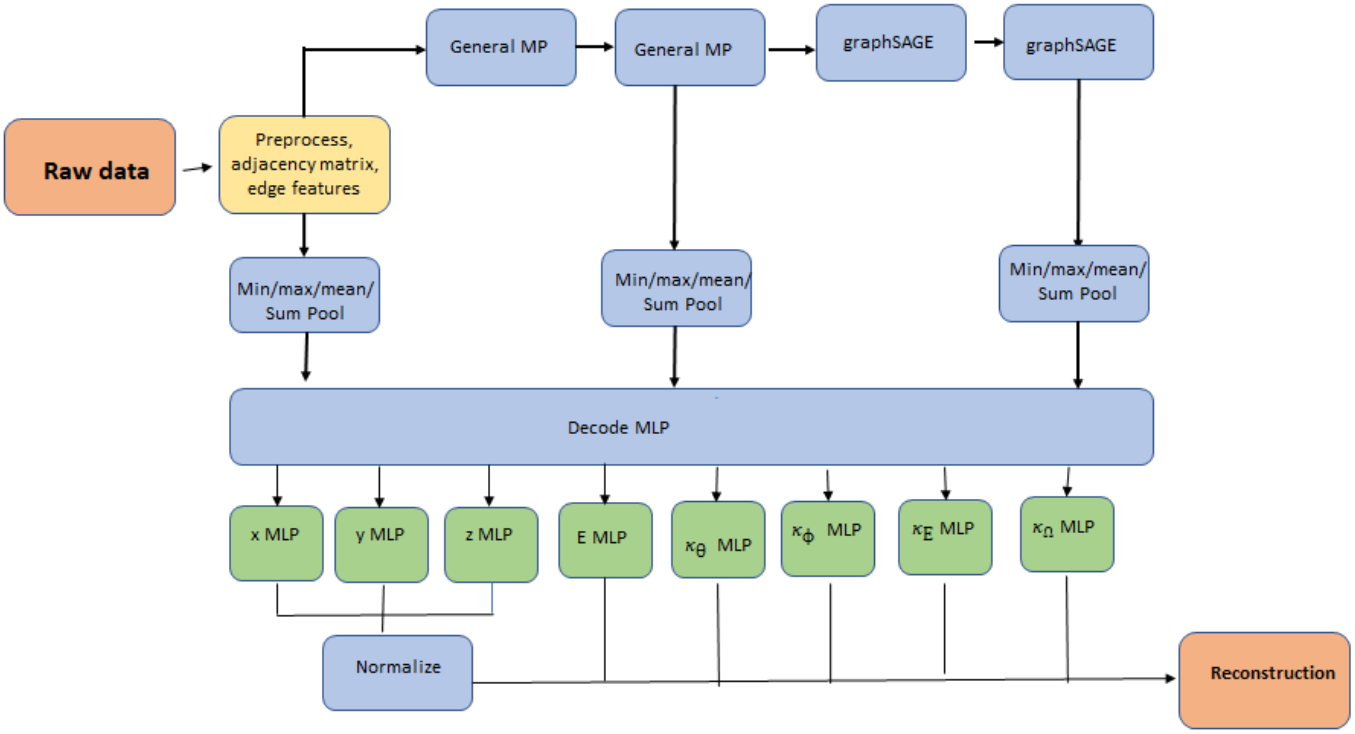

Figure 33: The architecture of AntHills

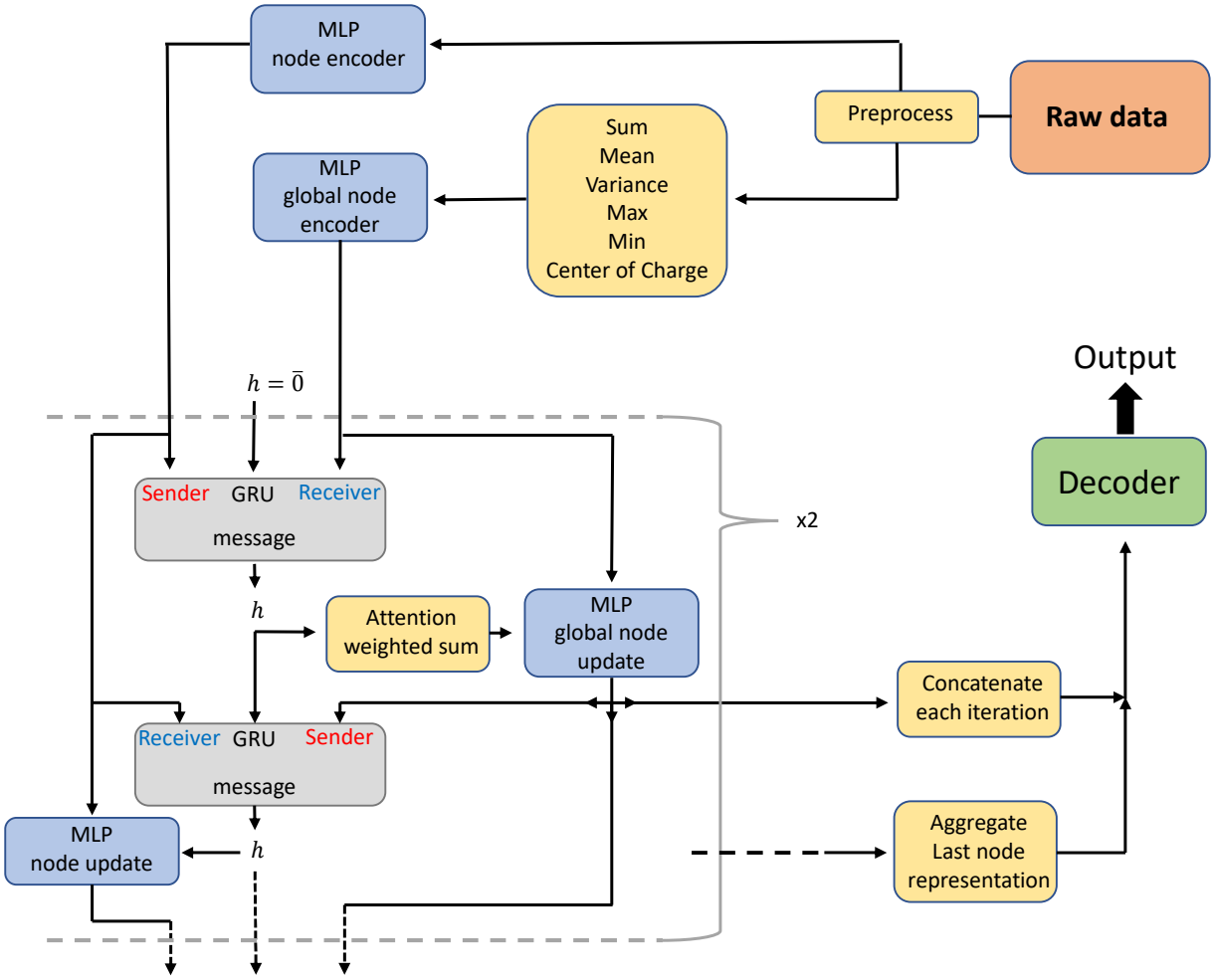

Figure 34: The architecture of GGConv

12.3 Target Feature Correlation

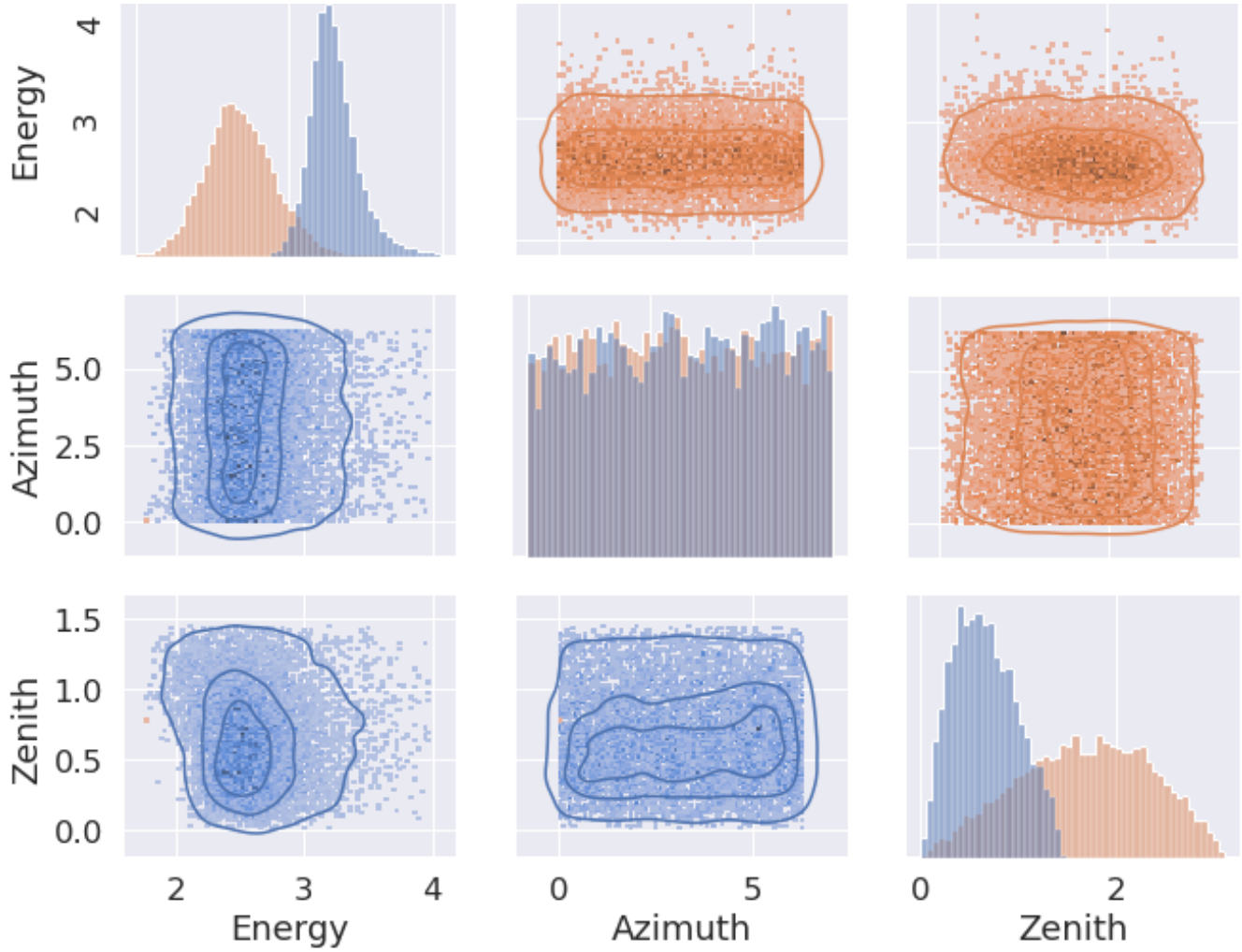

Figure 35: Correlation plot for the different output-features. MuonGun is blue. OscNext is orange.

12.4 Muon Angle Performance

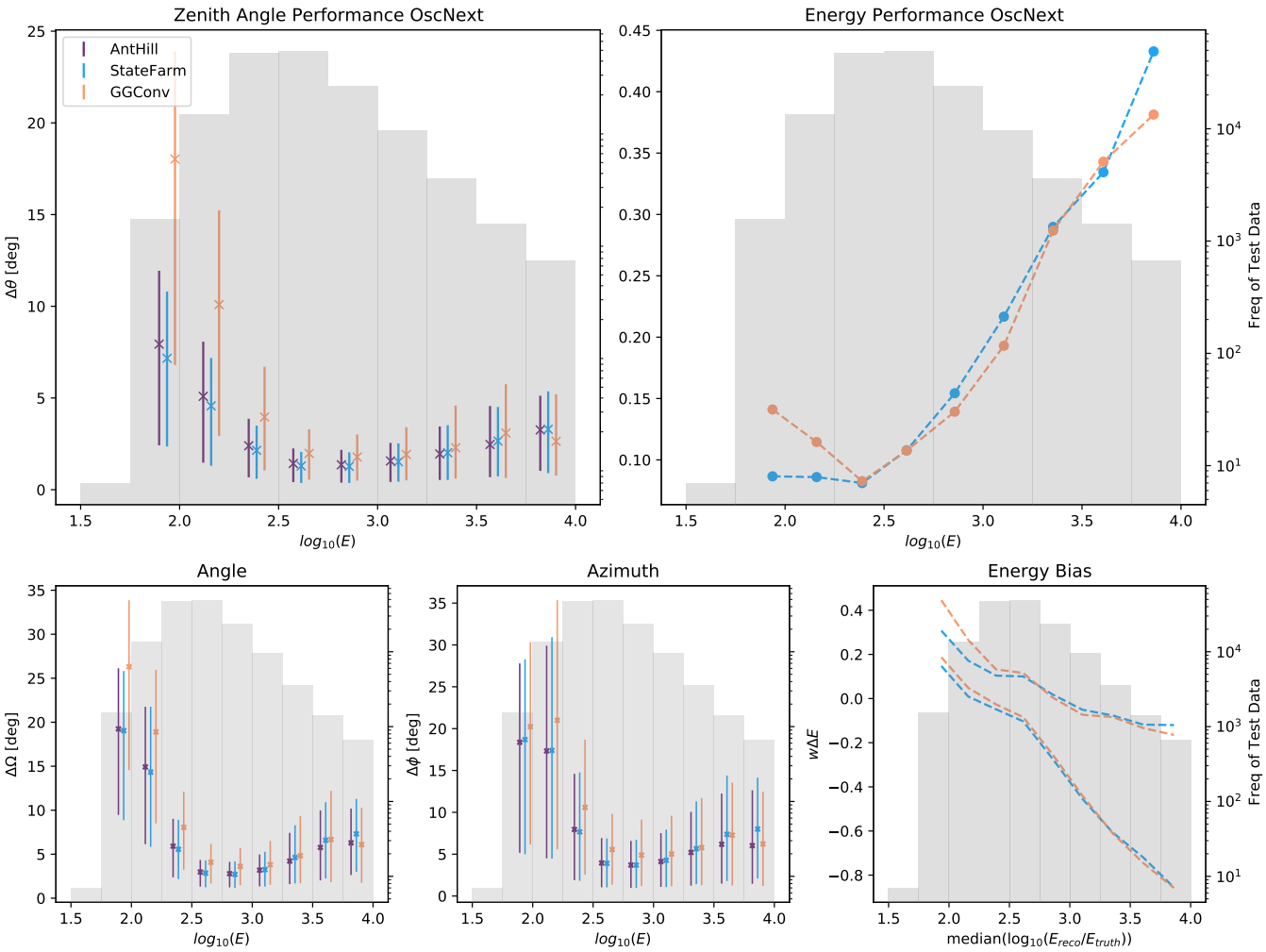

Figure 36: An energy-dependent view over the performance of the architectures on muons.

12.5 Moon Shadow Analysis

In Eq. 44, $N = \int_{\Omega_{x,y}} (1 - fe^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}}) dxdy$. Keep in mind x and y are angle differences, but as mentioned earlier, they are treated as cartesian coordinates. This model is a uniform distribution with a Gaussian subtracted, so $f \in [0,1]$ is the ratio of the darkest part of the shadow and the background and μ is the placement of the shadow and σ is the width.
For a square domain of length and height $2a$ and with the standard definition of the error function $erf(z) = \frac{2}{\pi} \int_0^z e^{-t^2} dt$ the normalization constant N can be shown to be:

$$N = 4a^2 - \frac{1}{2}\pi f\sigma^2 \left[\left(erf\left(\frac{\mu_x + a}{\sqrt{2}\sigma}\right) - erf\left(\frac{\mu_x - a}{\sqrt{2}\sigma}\right)\right) \left(erf\left(\frac{\mu_y + a}{\sqrt{2}\sigma}\right) - erf\left(\frac{\mu_y - a}{\sqrt{2}\sigma}\right)\right)\right]$$

Similarly, by using the error function, a closed form expression can be found for $P(x,y|\boldsymbol{\theta},\theta_{x_i}) = \int_{\Omega_{x,y}} \frac{dxdy}{N} \left(1 - fe^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}}\right) \mathcal{N}(x,y|\mu_i,\sigma_i)$. The result of this integral is much too complicated and long to be useful to write here, but it is useful when carrying out the minimization of Eq. 42 and can be found in https://github.com/Vinther901/Neutrino-Machine-Learning/blob/master/MoonAnalysis/Moon_Shadow_MC_testing-Copy1.ipynb as "expr2"

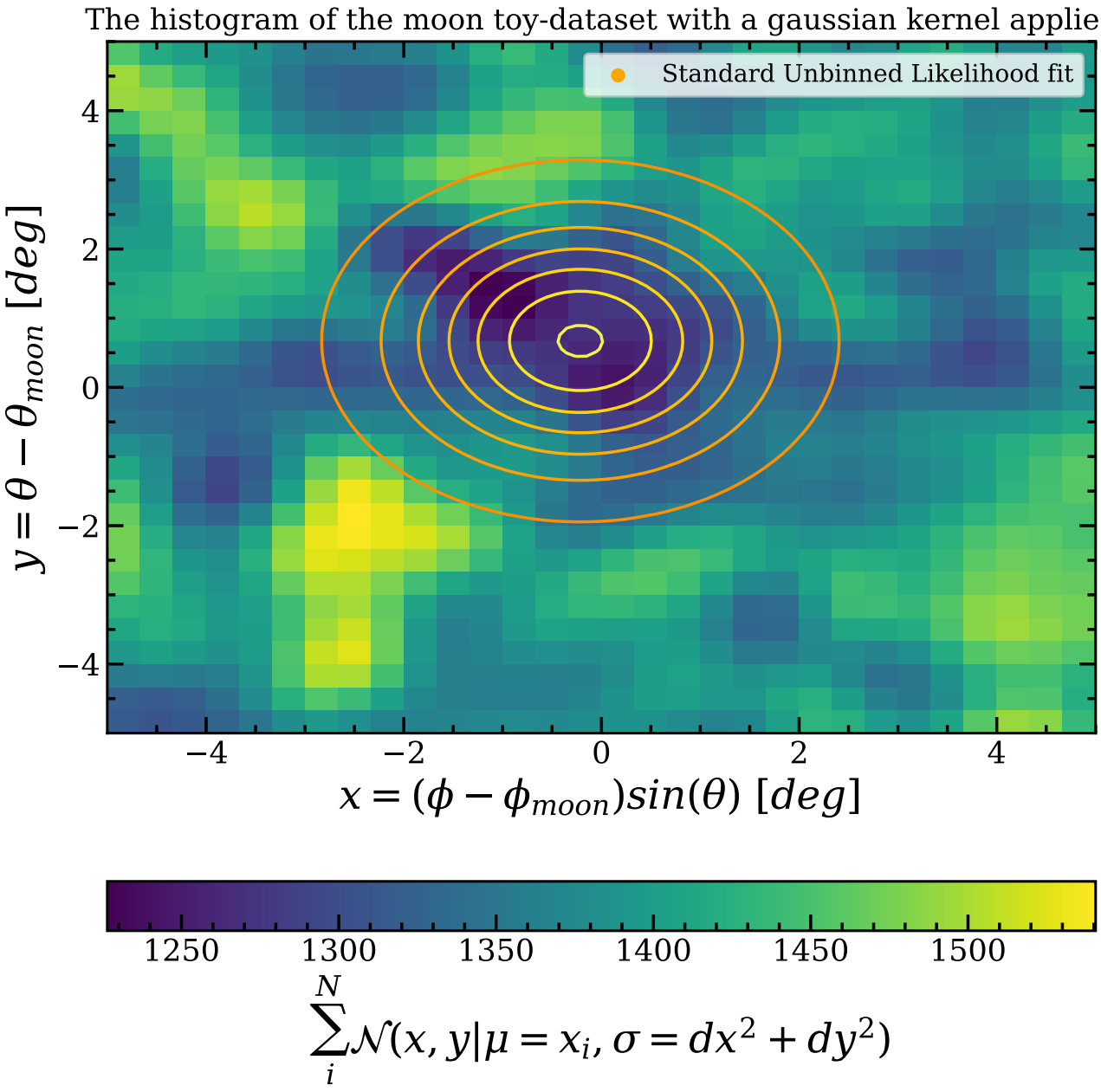

Figure 37: The 2d-histogram of the moon toy-dataset with a Gaussian kernel applied. This reveals why the standard unbinned likelihood places the fit function as it does. The standard method estimates a shadow ratio of $f = 0.13 \pm 0.04$, which can be explained by the ratio estimated from this and turns out to be $\sim (1 - \frac{1230}{1420}) \approx 13\%$. Furthermore, both the distance from the origin and the width is off with about $\sim$ 2.4 and 2.6 sigma, respectively, which are acceptable deviations.

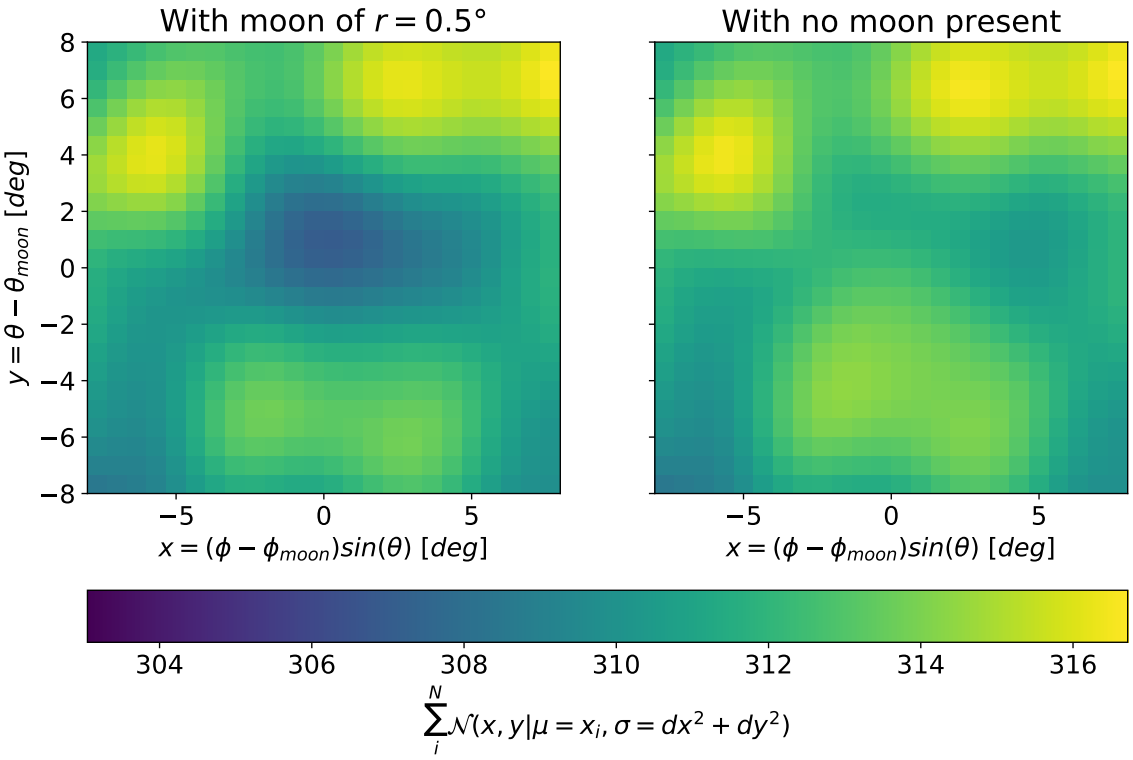

Figure 38: Another simulation with a moon present compared to if there were no moon shadow.

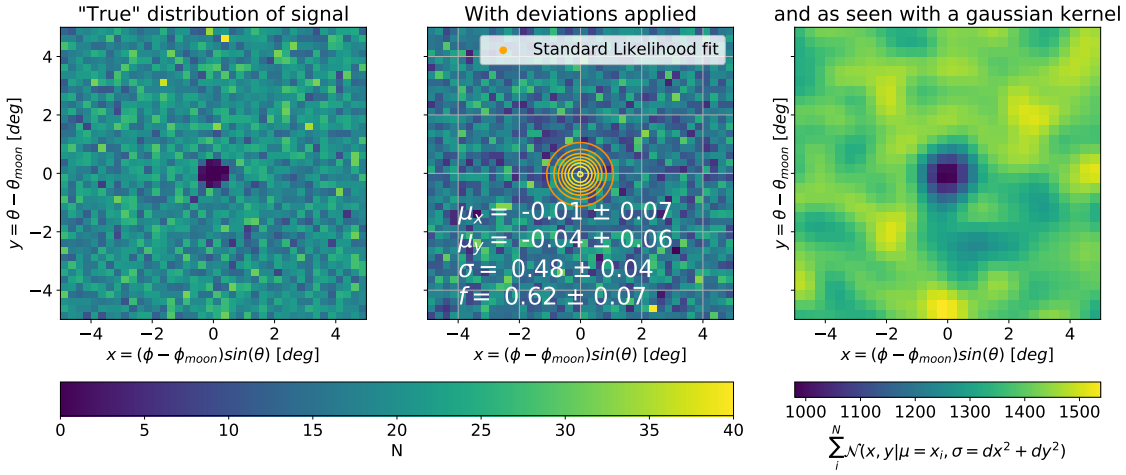

Figure 39: A less difficult case, where the deviations and uncertainties are scaled by a factor $\frac{1}{5}$. Using Wilk's theorem a pval of 7e-21 over the null hypothesis of a uniform background is obtained.

12.6 Performance Figures

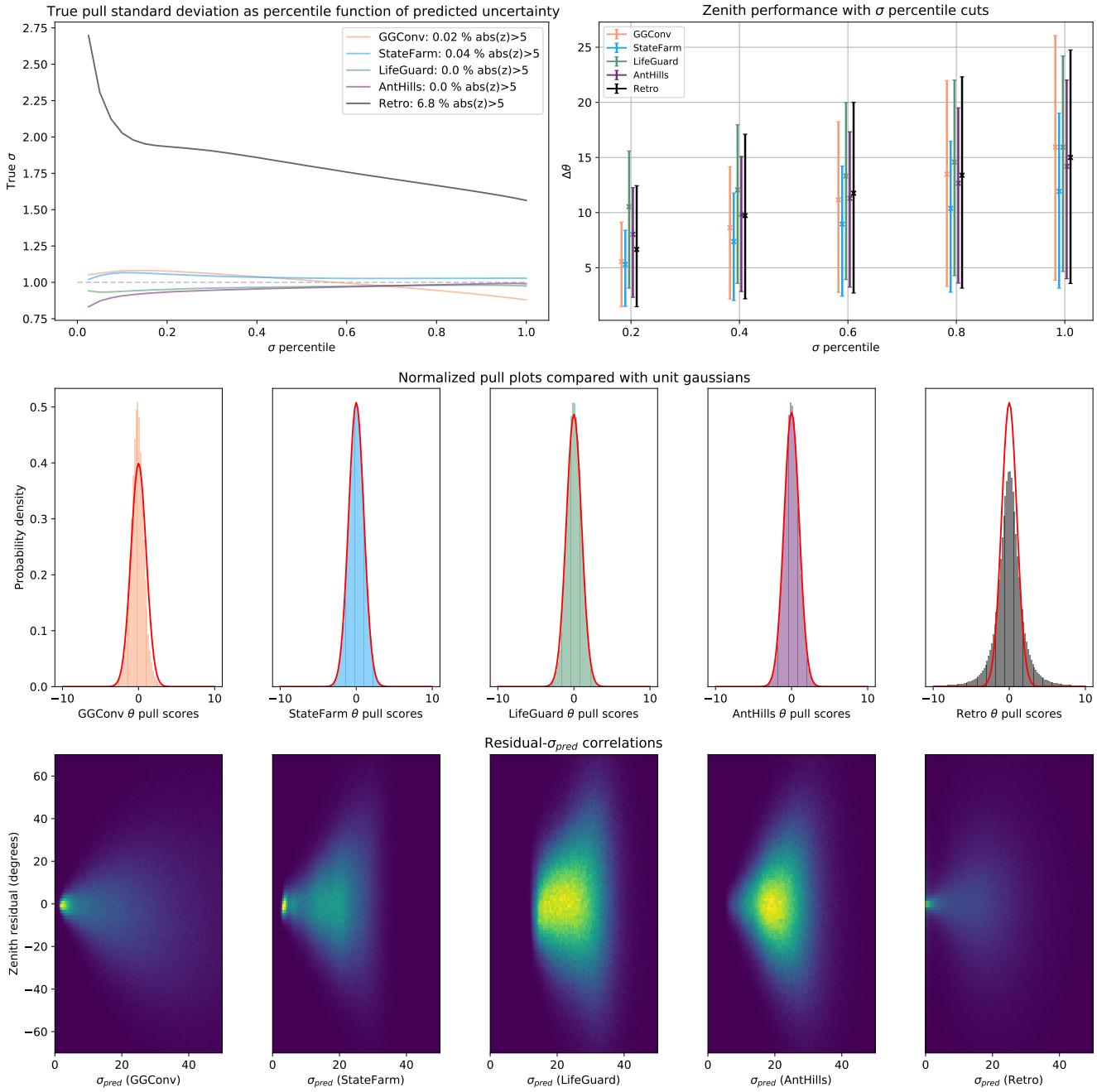

Figure 40: Zenith sigma analysis for our models and Retro. First row shows the true standard deviation of pulls masked as below percentile of predicted uncertainty and the performance (16th, 50th and 68th percentile) for predictions masked as below a certain percentile of predicted uncertainty. Second row shows pull plots for and unit gaussians for reference. Third row shows the correlation between predicted uncertainty and zenith residual.

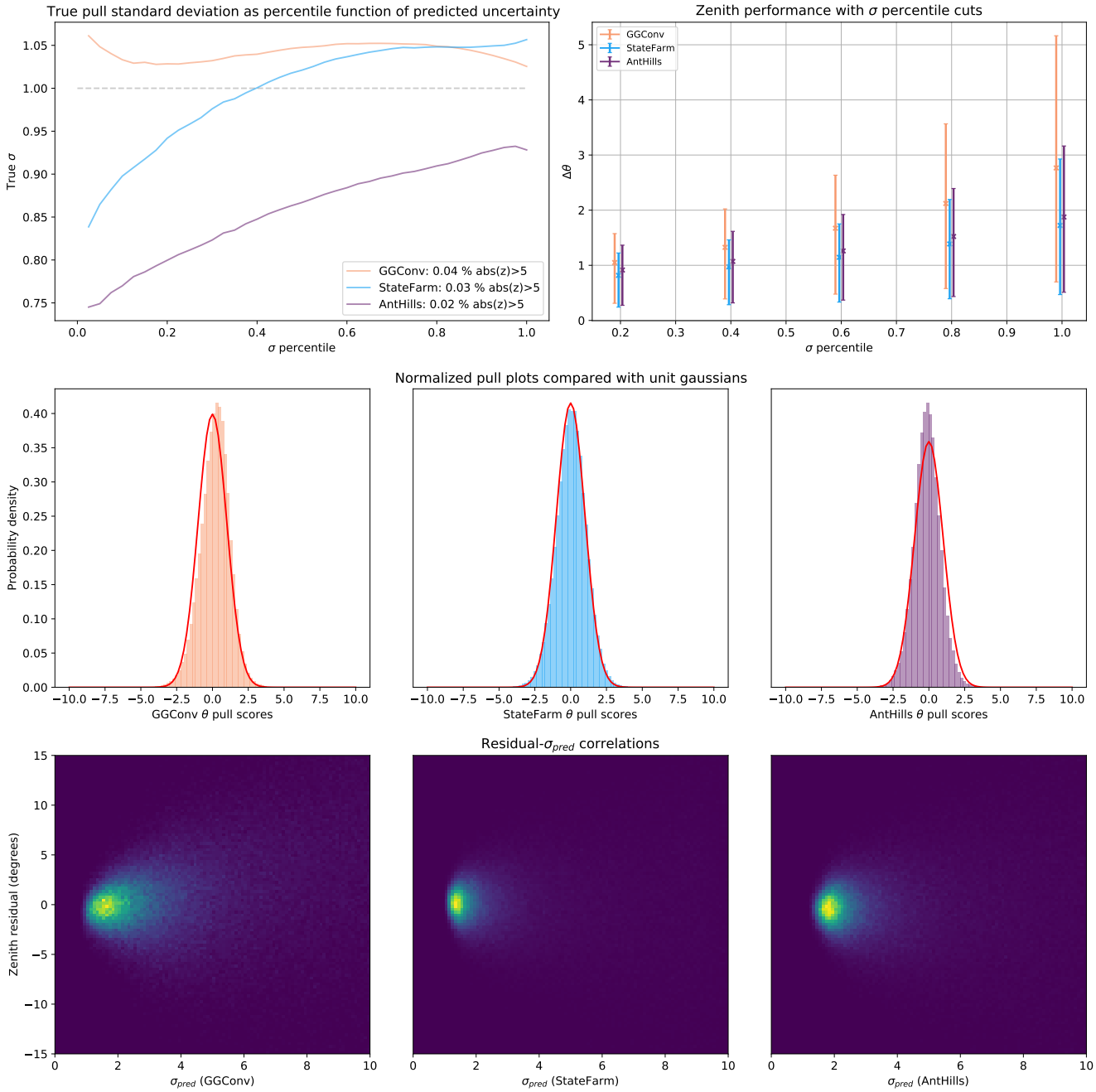

Figure 41: Zenith sigma analysis for three of our models on MuonGun data. First row shows the true standard deviation of pulls masked as below percentile of predicted uncertainty and the performance (16th, 50th and 68th percentile) for predictions masked as below a certain percentile of predicted uncertainty. Second row shows pull plots for and unit gaussians for reference. Third row shows the correlation between predicted uncertainty and zenith residual.

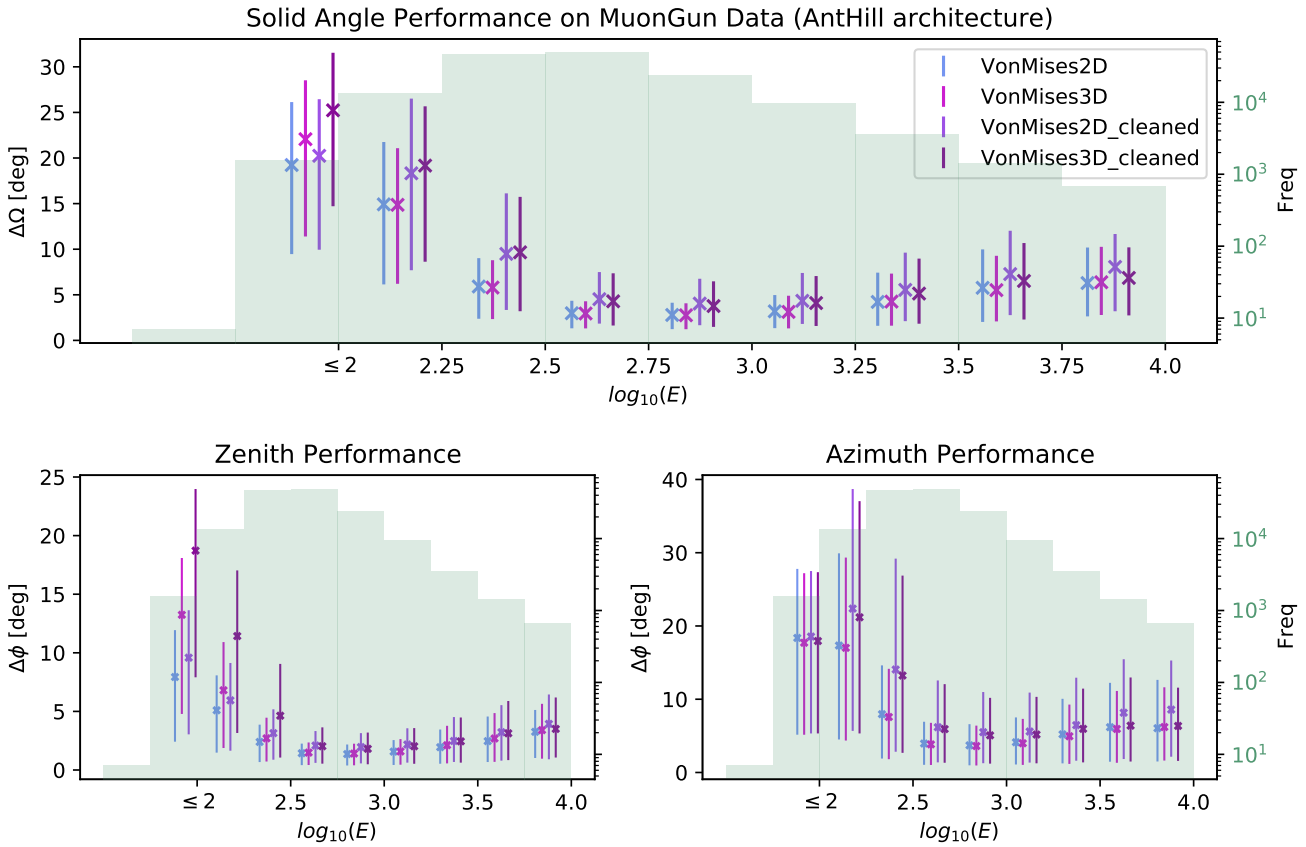

Figure 42: Prediction performance on the angles in MuonGun.

12.7 Explainability

Due to numerical instability in the gradient, we implemented taking the average of 50 runs of the Integrated Gradient method.

The heatmap showing the correlations are included here.

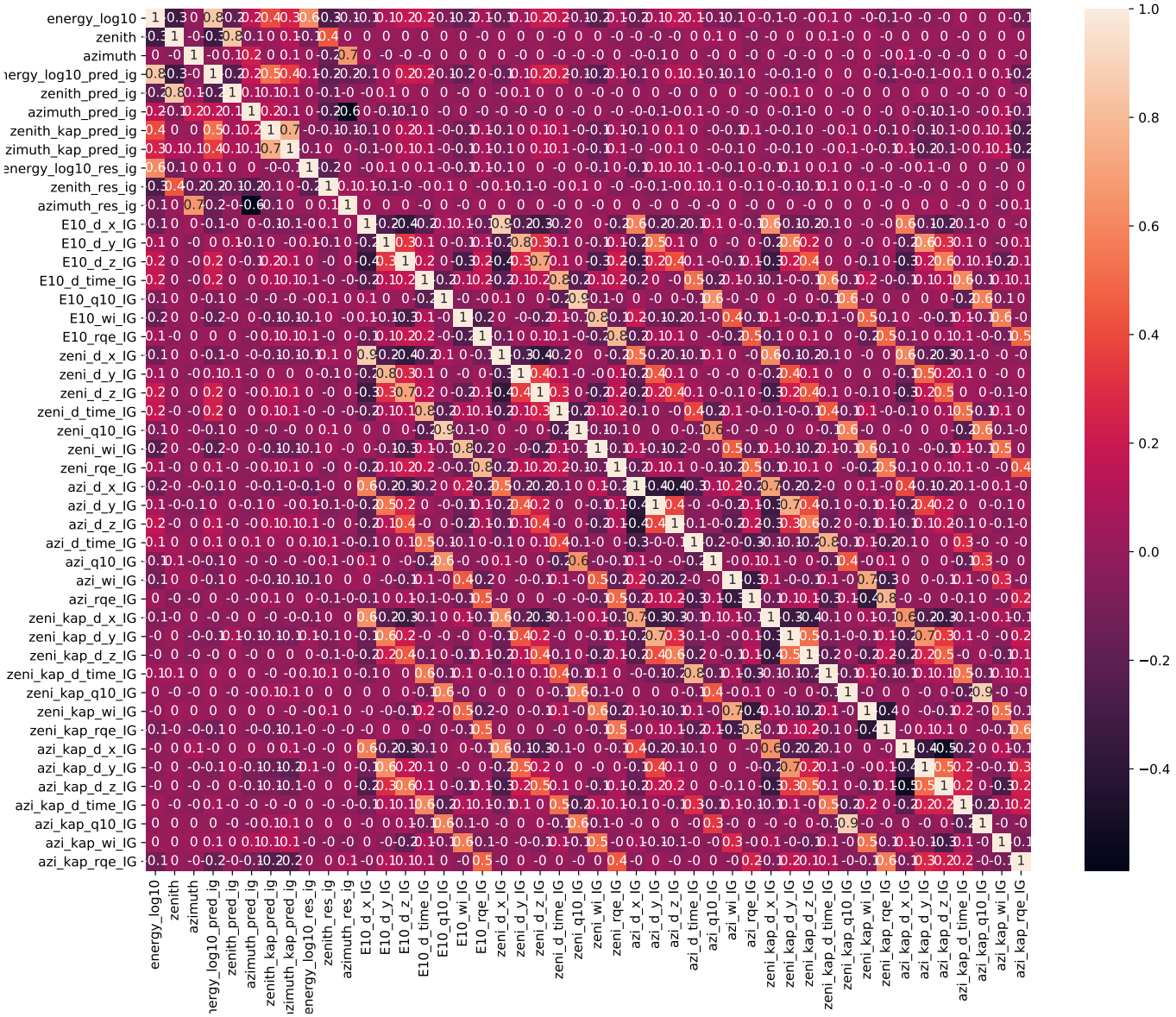

Figure 43: Correlations between parameters for Integrated Gradients

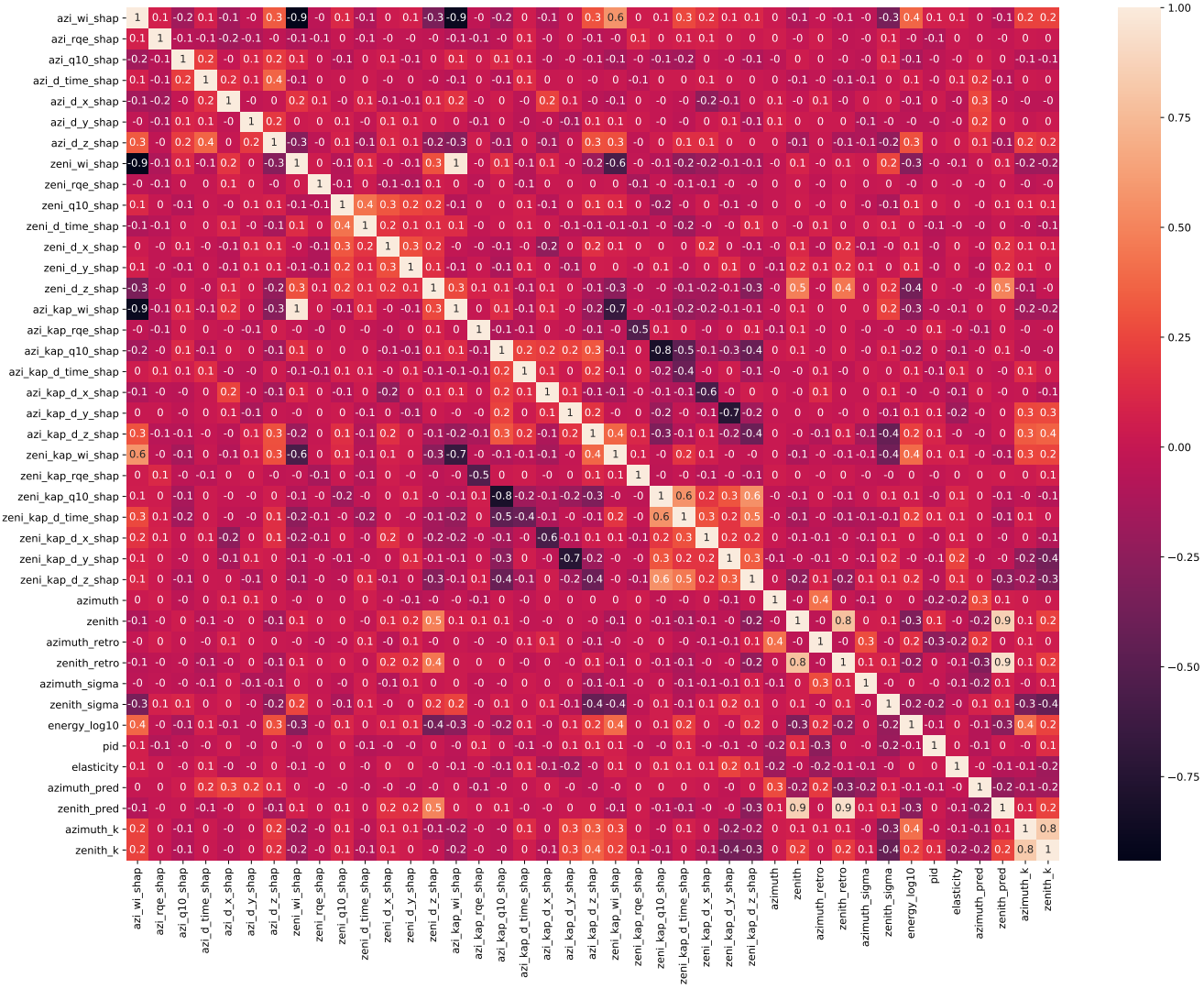

Figure 44: Correlations between parameters for SHAP

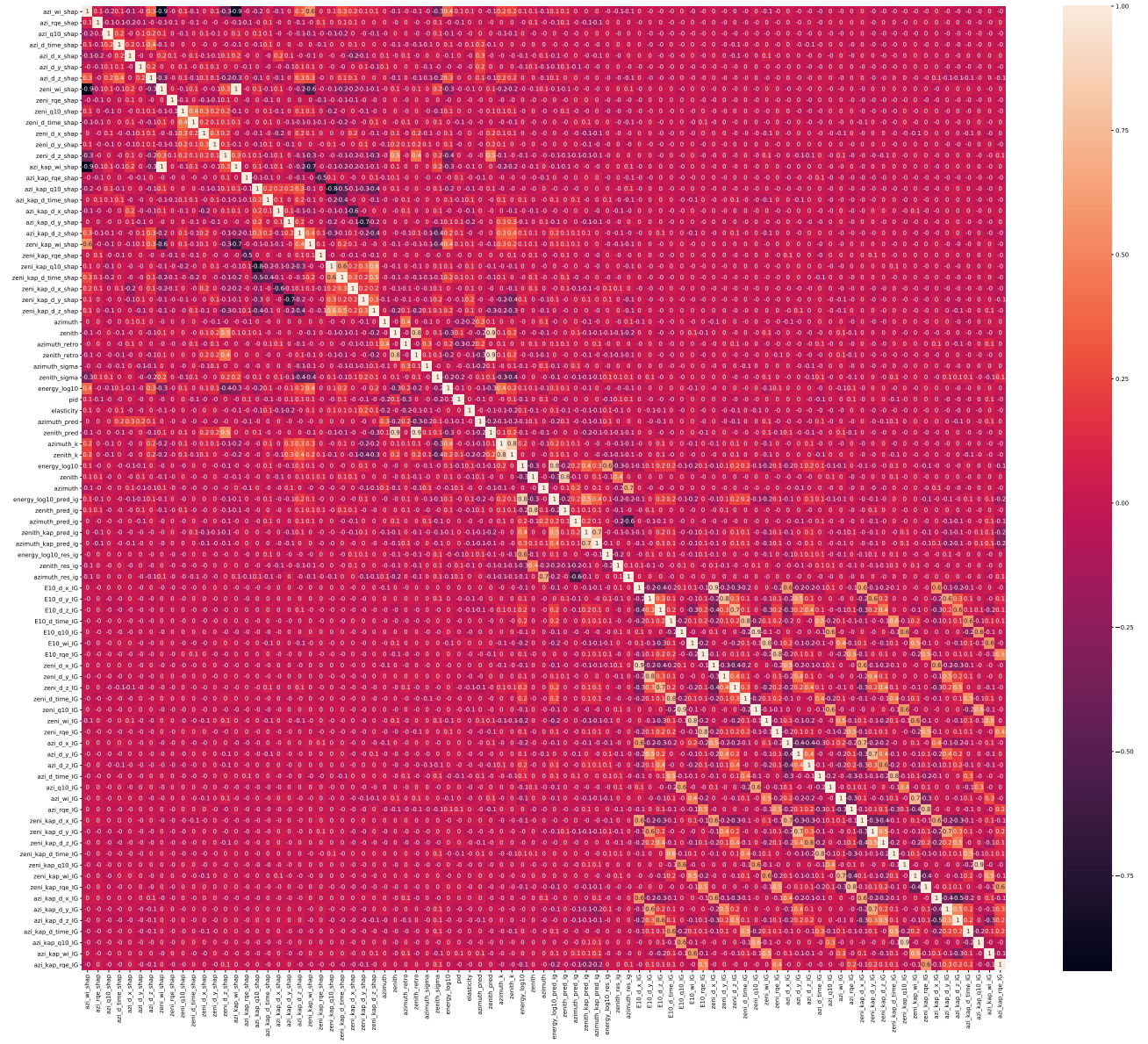

Figure 45: Correlations between parameters for SHAP and IG

12.8 Comparison of Angular Loss Functions

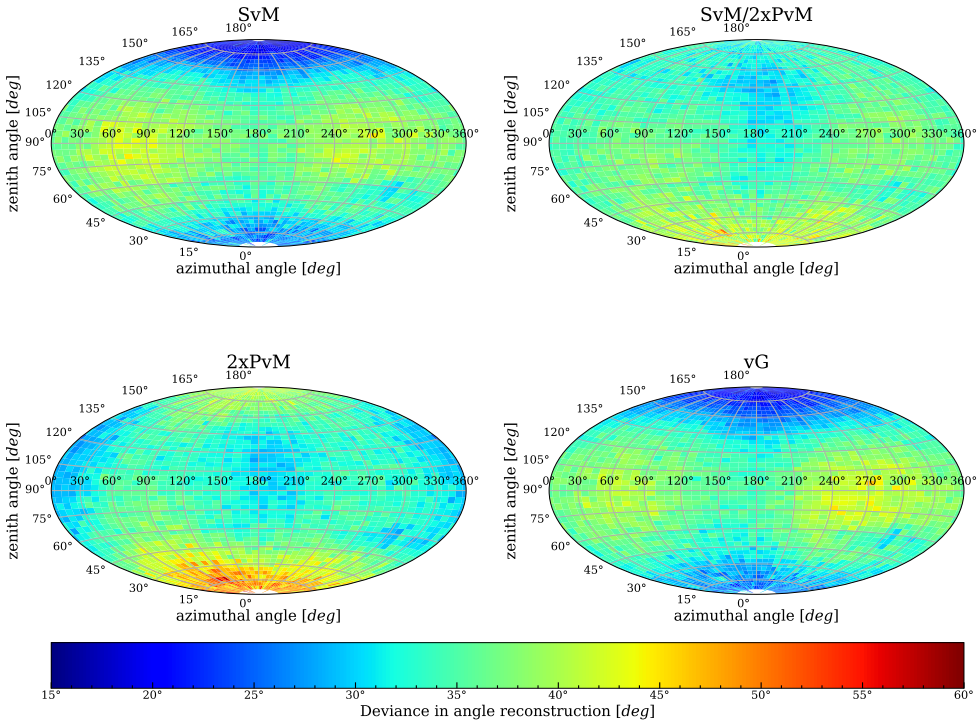

Figure 46: Directional reconstruction performance as a function of where the particle came from for the considered loss functions

12.9 From MC to Data

Generalizing Monte Carlo trained ML-methods to real data is far from trivial. In this project, no special effort to achieve this have been made, however, a simple examination of whether our models are applicable to data is made. In Fig. 27 the correlation between predictions made by GGConv and Retro is compared for MC and data predictions. Ie. for our models to be applicable to data the red contours should lie on the black heatmatp, which it does. This gives an indication of that our results carry over to data. No changes were made for the predictions for data, except for the feature pulse_width, which was scaled by a factor 2 since it, rather peculiarly, lay predominantly at the values 2 and 16. Furthermore, since the only values pulse_width can take in MC is 1 and 8, the data sample was sub-sampled for events for which no DOM had a pulse_width feature other than 2 or 16, effectively cutting the sample in half. The data sample used is the full **IC86.11** data sample, which totals $\sim$ 64.000 events [58].

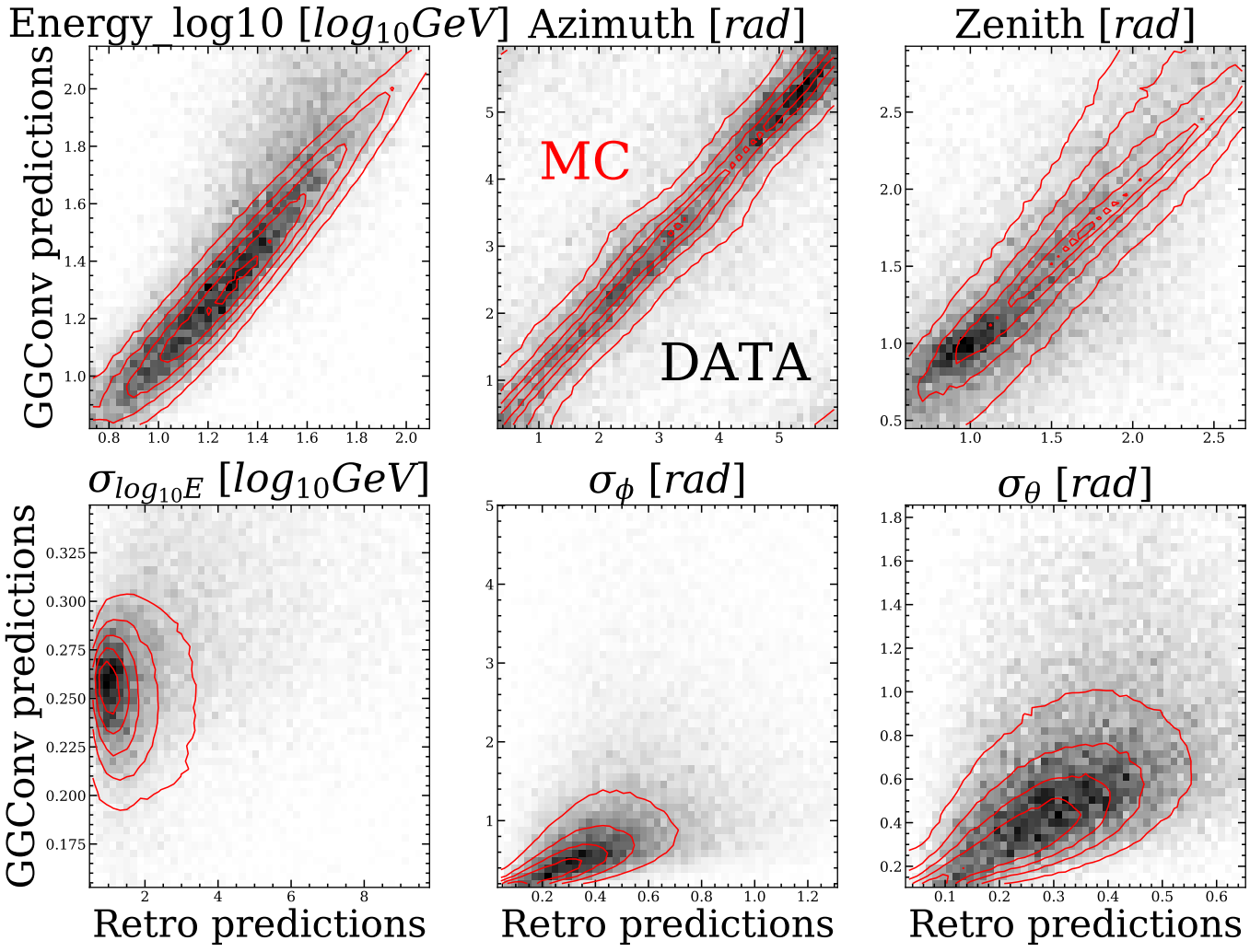

Figure 47: Applicability of the Monte Carlo trained ML methods in real data. Shown are the contour-plot of the correlation between predictions made by GGConv and Retro for energy, θ, ϕ and their uncertainties in MC. The underlying heatmap contains the same but in real data. The data sample size is significantly smaller. The interval of the axes are chosen to include the inner 90% of the sample and are equal for MC and data

13 Honorable Mentions

Opening Angle Limit

One desired aspect of the project that we did not accomplish was to have a true information limit to work with. This could have been done by using the `Pythia` package (https://pythia.org/, to estimate the initial opening angle as a function of energy. This opening angle would then act as a hard information limit.

Covariance Prediction

We attempted to also predict the covariance between the directional reconstruction predictions, on top of the individual uncertainties, however, we did not have time to finish it. It would have been included in the moon shadow analysis, had it been successful.

Self-Attention Graph Pooling

Due to a gut feeling, the group worked tirelessly on an implementation of the SAGPool architecture [59]:

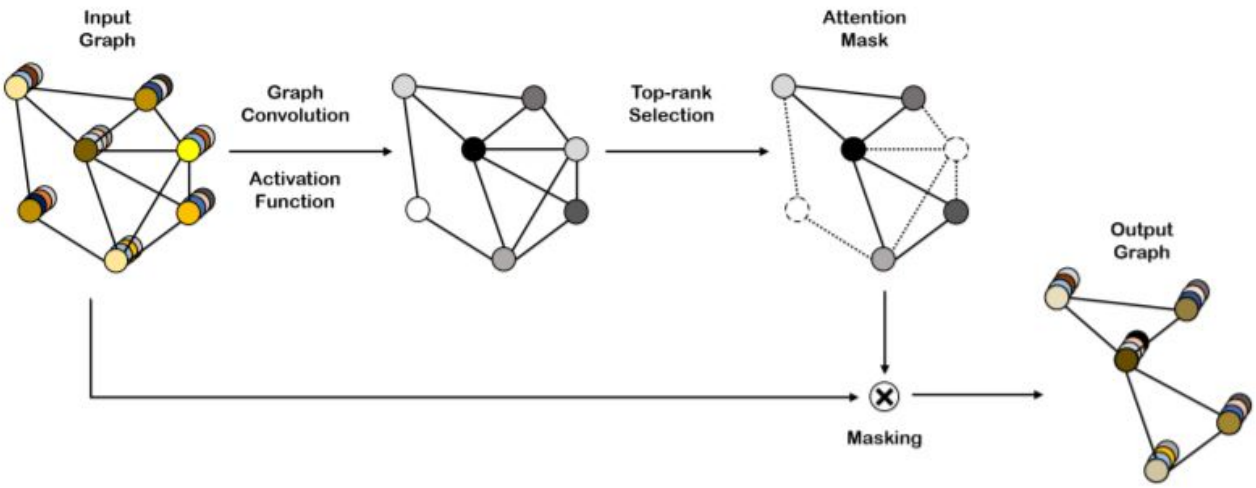

Figure 48: The SAGPool layer

As this lies within the *Honorable Mentions*-section, it can be surmised that the work was ultimately unsuccessful.

Cartesian Space and Number of Triggered DOMs

When considering the extremes of uncertainties, some interesting structure shows. As can be seen in Figs. 49- 52, the events that are most uncertain mostly lie outside DeepCore. We hypothesise the events that only trigger the lower part has particles travelling outside the detector, leaving only an uncertain Cherenkov-wake coming in from the bottom. It can also be seen that the most certain events has a high number of triggered sensors. This was expected as we usually see a proportionality between regressional abilities of our models and number of DOMs.

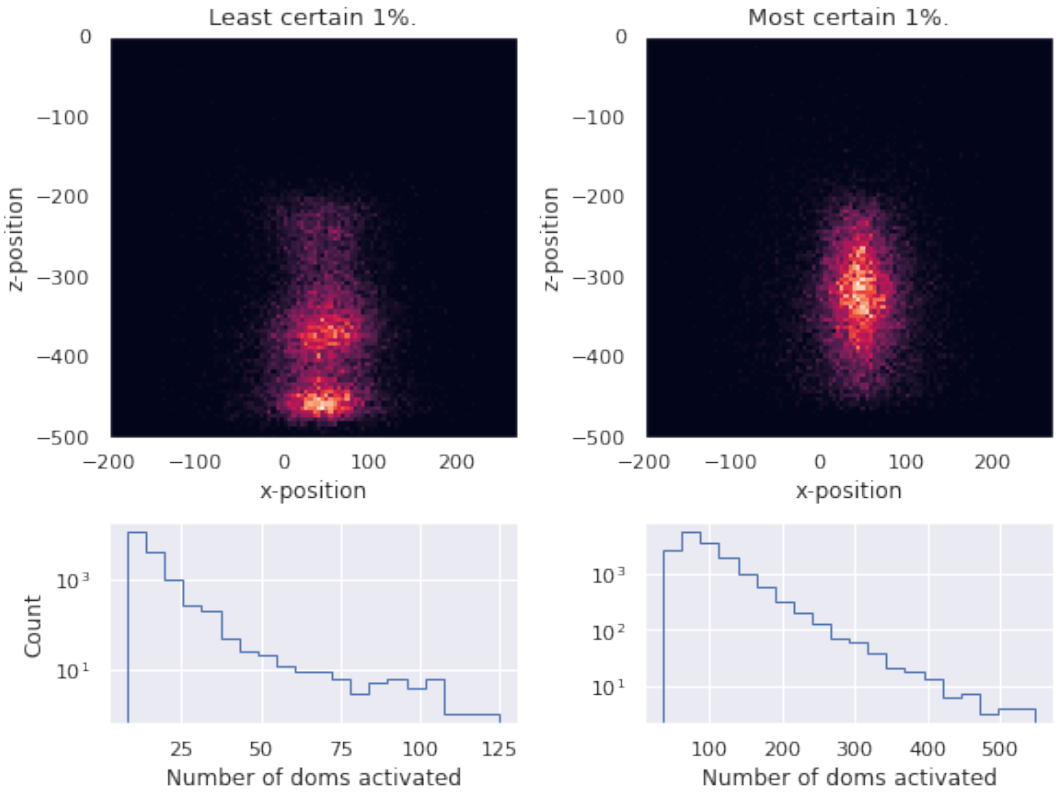

Figure 49: The density of center of charge in x-z for the best/worst percent of regressions from StateFarm

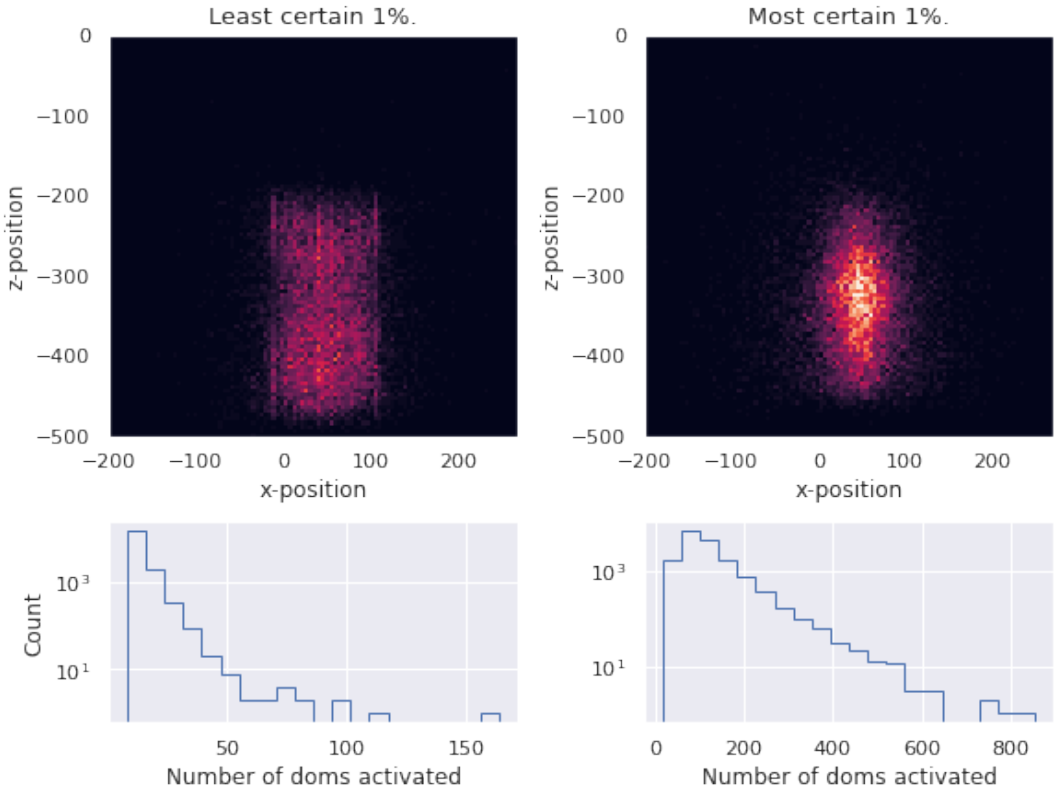

Figure 50: The density of center of charge in x-z for the best/worst percent of regressions from GGConv

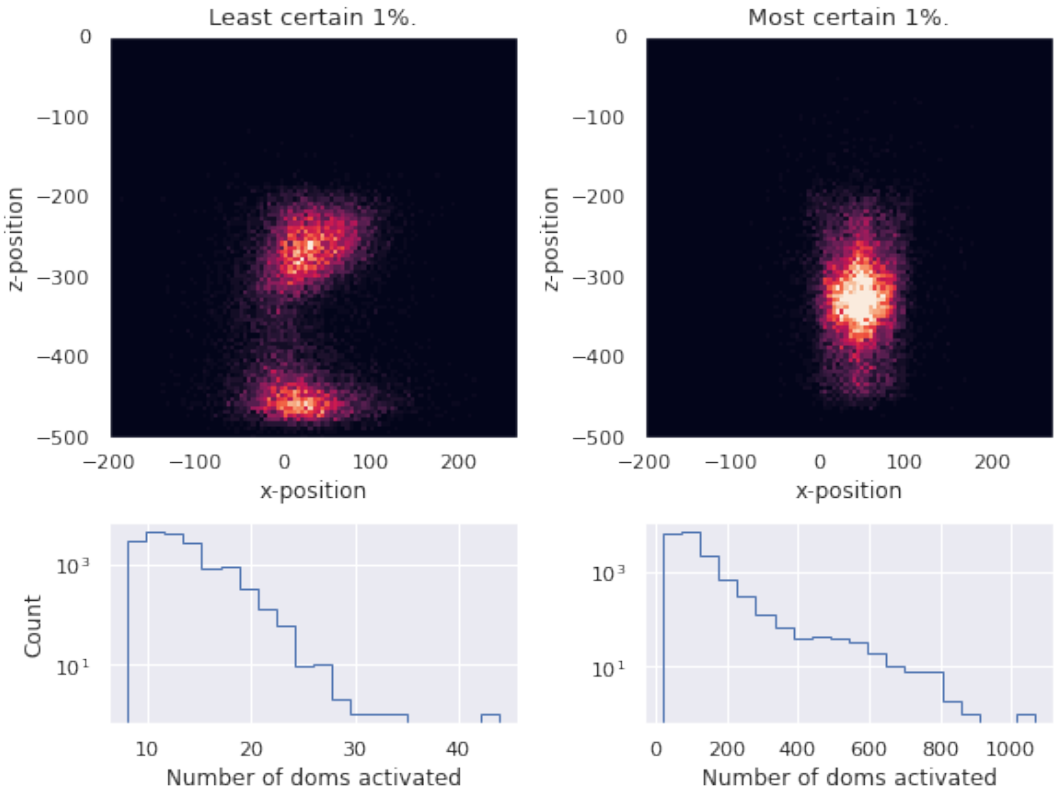

Figure 51: The density of center of charge in x-z for the best/worst percent of regressions from LifeGuard

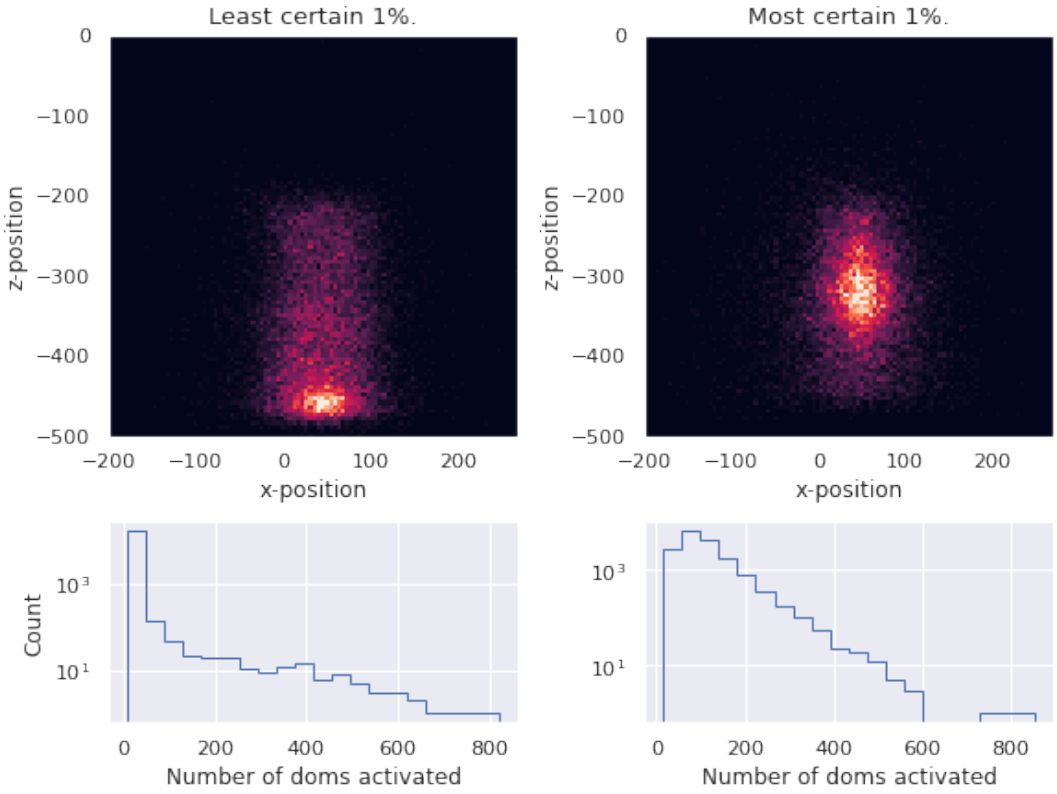

Figure 52: The density of center of charge in x-z for the best/worst percent of regressions from AntHill

Training 1000 Monkeys with Typewriters

Due to a lack of computational power this did not work: Copenhagen Zoo would only lend us 5 gorillas but due to high demand on the institute the gorilla quota to bachelor students was very limited.

Nearest Neighbours on a PCA axis

In this project the Nearest neighbours was used to generate the adjacency matrix. However since there is no given edges in the actual data, the choice of generating them is arbitrary and can be chosen to give the best results. One other investigated method was to use the Principle Component Analysis to find the primary axis from the position of the activated DOMs and the projection on this axis could now be used, such that pulses close to each other along this axis was connected in the graph. This method was tried on the MuonGun data and gave approximately the same accuracy as the nearest neighbour approach, but no improvement. Thus this idea was demoted to an honorable mention.

Elasticity

Classifying antineutrinos from neutrinos, could have been helped by regressing the elasticity in the decay (Fig. 53). This was looked at, but in trying to keep this project streamlined it was not worked on. Given more time this seems like an obvious next step for the ensemble model, as the addition of a PID-identifier was an improvement.

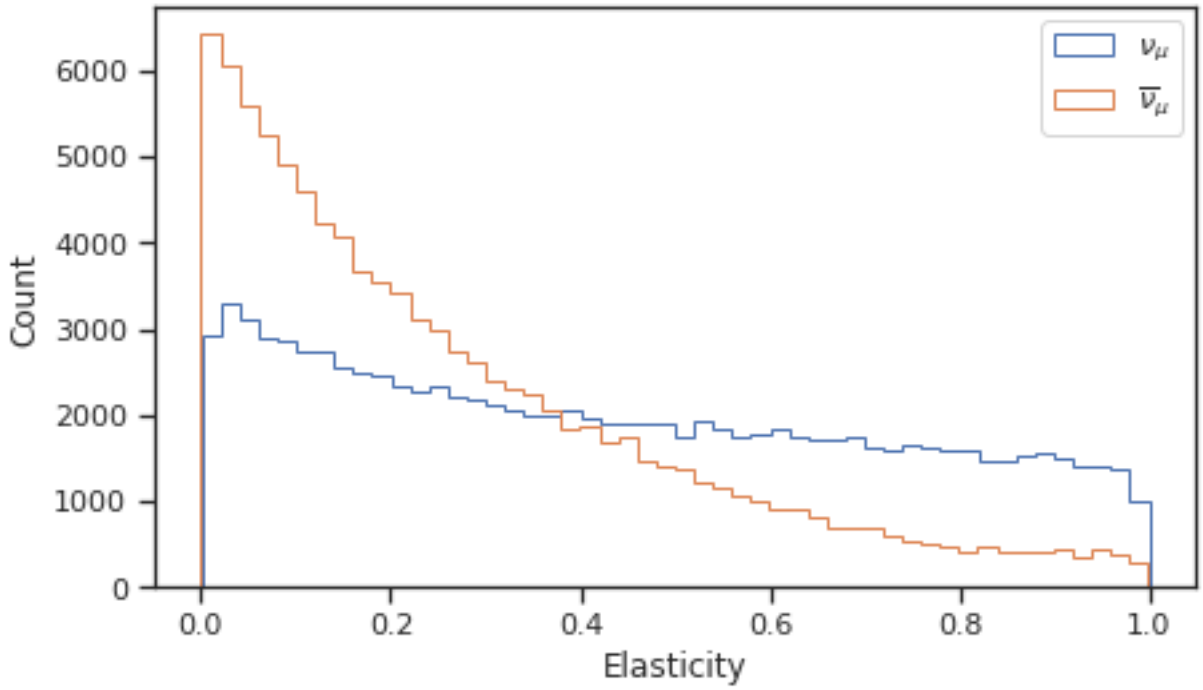

Figure 53: Distribution of elasticity, showing that **if** one could regress elasticity, one could most likely distinguish between ν_α and $\bar{\nu_\alpha}$.

Wrapped Cauchy Distribution as Angular Loss

A loss function that can be employed in order to mitigate the trouble with not hitting the edges of the zenith distribution is the Wrapped Cauchy Distribution, an analogue to the Polar vMF but for the Cauchy Distribution given by.

$$f_{WC}(\theta;\mu,\gamma) = \sum_{n=-\infty}^{\infty} \frac{\gamma}{\pi(\gamma^2 + (\theta - \mu + 2\pi n)^2)} \qquad -\pi < \theta < \pi \tag{45}$$

Here however, the overall precision goes down drastically, and so was not included in the main project

Reweighting

In an attempt to fix predicted neutrino zenith distribution, we attempted to weight the azimuth and zenith such that the phase space had uniform density, but unfortunately this was unsuccessful.

Directed-in-time Graphs

A possible add-on is to make the graph *directed*, by having nodes only be connected one way, with some kind of physical rationale behind this choice. One such method that was attempted was a *forward-in-time* graph, where any pair of neighbours were asymmetricized in time such that only the one that had a signal first in time could connect to the other. Thus, it would be impossible to do message passing backwards in time, effectively letting it work like a temporal CNN.

Kolmogorov-Smirnov Correctional Loss Term

Another approach to force the predicted distribution to approach the target distribution was to add a Kolmogorov-Smirnov statistic term to the loss function. The idea is to compare the cumulative probability distribution of the predicted with the truth. Now the Kolmogorov-Smirnov statistic is given as the biggest difference of the two distributions. To do this differentiable and numerically, this statistic was done by using search sort of both distributions with respect to the sorted list of them combined. Now the biggest difference in the sorting can be converted to the Kolmogorov-Smirnov and was added to the loss. The implementation is heavily inspired by the `scipy` 2-sample KS-test.