Cover page

Exam information
NFYK10020E - Physics Thesis 60 ECTS, Niels Bohr Institute - Kontrakt:121072 (Rasmus Faldt Ørsøe)

Handed in by
Rasmus Faldt Ørsøe
pcs557@alumni.ku.dk

Exam administrators
Eksamensteam, tel 35 33 64 57
eksamen@science.ku.dk

Assessors
Troels Christian Petersen
Examiner
petersen@nbi.ku.dk
+4535325442

Rasmus Mackeprang
Co-examiner
rasmus@mackeprang.com

Hand-in information

Titel: A Graph Neural Network Approach to Low Energy Event Reconstruction in IceCube Neutrino Observatory
Titel, engelsk: A Graph Neural Network Approach to Low Energy Event Reconstruction in IceCube Neutrino Observatory
Tro og love-erklæring: Yes
Indeholder besvarelsen fortroligt materiale: No

UNIVERSITY OF COPENHAGEN

MASTER THESIS

A Graph Neural Network Approach to Low Energy Event Reconstruction in IceCube Neutrino Observatory

Author:
Rasmus F. Ørsøe

Supervisor:
Troels C. Petersen

A thesis submitted in fulfillment of the requirements for the degree of MSc Physics

at

Experimental Particle Physics
Niels Bohr Institute

May 20, 2021

Declaration of Authorship

I, Rasmus F. Ørsøe, declare that this thesis titled, "A Graph Neural Network Approach to Low Energy Event Reconstruction in IceCube Neutrino Observatory" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, references are always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 20-05-2021

"Perhaps man wasn't meant for paradise. Maybe he was meant to claw, to scratch all the way."

Captain Kirk, USS Enterprise (NCC-1701)

UNIVERSITY OF COPENHAGEN

Abstract

Faculty of Science
Niels Bohr Institute

MSc Physics

A Graph Neural Network Approach to Low Energy Event Reconstruction in IceCube Neutrino Observatory

by Rasmus F. Ørsøe

This work investigates a graph neural network (GNN) approach to low energy event reconstruction in IceCube Neutrino Observatory by treating level7 oscNext samples as point cloud graph representations of low energy neutrino events. A GNN model is proposed and tested against current low energy reconstruction methods in both classification and regression tasks in MC data and on IC86.11 measurements. In MC data, this work finds a 15% increase in neutrino signal as compared to the current final level7 neutrino classifier, or equivalently, a reduction of background by 80% at the same neutrino signal strength. In addition, this work records 11.7%, 22.4% and 16.3% improvement in the widths of error distributions for regression targets *azimuth*, *energy_log*10 and *zenith*, respectively, as compared to RetroReco in the neutrino oscillation relevant energy range of 0 to 1.5 $\log_{10}$ GeV. The proposed model is capable of producing reconstructions at speeds of 15.000 events pr. second as compared to 5 - 40 seconds pr. event for RetroReco, which opens the potential for cosmic alerts from low energy neutrinos. Lastly, this work contains a characterization of at least part of the current pulse noise in IceCube Upgrade MC data, and finds these to originate from suspicious same-pmt activation patterns from mDOMs.

Acknowledgements

I would like to express my deep and sincere gratitude to the supervisor of this thesis, Associate Professor **Troels C. Petersen**, Niels Bohr Institute, for the continuous support of this work through opportunity, sound advice and enthusiasm both within and outside regular business hours! Also, a big sincere thank you to **Tom Stuttard**, Post-doc, Niels Bohr Institute, for answering my endless stream of technical questions on everything related to oscNext.

In addition, I'd like to thank former NBI master students **Mads Ehrhorn** and **Bjørn Mølvig** for laying the foundation from which this work was built and PhD student **Martin Minh**, and his co-supervisor Post-doc **Phillip Eller**, both from TUM, for their cooperation and interest.

Last, but not least, I'd like to thank Sara Pinholt, Malte Algren, Lau Mortensen, Christian Michelsen, Tania Kozynets, Kathrine Groth, Kasper Pedersen, Sofus Stray and Jonathan Jegstrup for good office companionship, many laughs and for all the good discussions!

It is the opinion of this stray theoretician that the work presented here would have had neither the scope nor the ambition had it not been for you. So thank you!

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 The History of the Neutrino 3

2 IceCube Neutrino Observatory 13

3 Data From IceCube 19

4 Machine Learning 25

5 Graph Neural Networks 33

6 Model Development 37

7 The Final Model: dynedge 61

8 Pre-Upgrade MC Results 65

9 Application on Upgrade Data 83

10 Application on Real Measurements From IceCube 89

11 Conclusion & Outlook 97

A Prototyping Dataset Distributions 103

B Quick guide: NBI HEP Cluster 113

C Notes on i3, CVMFS and Shovel 117

D The NBI Intergalactic IceCube Predictions Treaty 121

E SQLite Databases 125

F Making Databases 127

G Geometric Model 131

Bibliography 133

In Memory of

My loving grandmother,
Norma Bentsen, 94

My dear cousin,
Lars Ørsøe, 33

Reader's Guide

This is a reading guide that seeks to provide context to the reader from which the work can be understood.

The Problem

The motivation for this thesis comes from a few observations that was made both within and outside IceCube, and by providing you with these I hope the structure of this work becomes logical. Let's start with the observations made outside IceCube prior to the start of this work:

1): Machine Learning is often many orders of magnitudes faster than traditional, statistical methods, and it is already used in physics experiments to analyze data.
2): A new type of machine learning algorithm called **graph neural networks** (GNN's) have been shown to work well with irregularly spaced data in classification problems, but (at the time) nearly no papers existed on applications of GNN's in physics.

Within IceCube, the following observations were made:
1): Reconstruction of low-energy neutrino events in IceCube is done using a statistical model named **RetroReco**, which takes around 5-40 seconds to reconstruct a single event. This is considered to be slow.
2): The IceCube neutrino detector is built as an irregular shape.
3): Low-energy events are important because they are used to study neutrino oscillations, a still relatively ill understood physical phenomenon.

The core question of this work then naturally became:
Can a GNN be used to reconstruct low-energy neutrino events in IceCube, and if so, how does it compare to RetroReco?

Because the RetroReco algorithm is slow, it would be considered a success if comparable, even if slightly inferior, results could be obtained at increased reconstruction speed.

Structure of This Work

In Chapters 1 and 2, the reader is introduced to a short overview of a few historic neutrino experiments, a description of IceCube and it's research goals. In Chapters 3 to 5 follows a description of the data from IceCube, an introduction to Machine Learning and a brief explanation of graph neural networks. This establishes the foundation for the work.

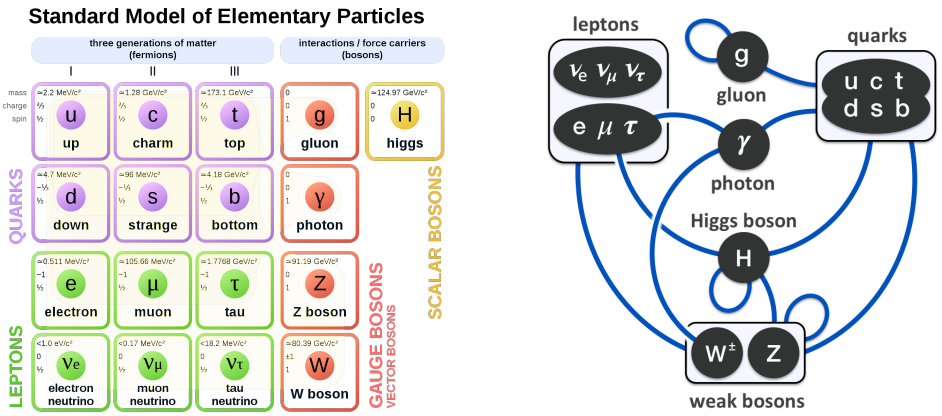
In Chapter 6 is a detailed description of the development process of the GNN, which can be skipped completely. In Chapter 7 is a description of the final model. In Chapter 8, a comparison between RetroReco and the model developed in this work is shown in level7 MC data. In Chapter 10 a real data comparison is shown on the level7 IC86.11 sample.

Code related to this work is available at: https://github.com/RasmusOrsoe/GNNIceCube

Chapter 1

The History of the Neutrino

The standard model is a modern take on the conclusion that Leucippus and Democritus arrived at sometime in the 5th century, when they philosophically deduced that if one were to keep dividing a substance, one would eventually arrive at something that is indivisible [1]. That *something* is today referred to as *elementary particles* and the standard model of particle physics is the current framework in which these are understood. Within the scope of the standard model (**SM**) such particles are divided into two main categories: **fermions** and **bosons**. On the technical side of things, Fermions are half-integer spin particles that obey Fermi-Dirac statistics and are divided into two further categories: **quarks** and **leptons** and their respective anti-particles. From a philosophical perspective, it is perhaps the quarks that resembles the ancient Greek ideas best as quarks are both massive and indivisible. Not all Leptons are described by the standard model as massive, since all current neutrino flavors are considered to be massless - a direct consequence of the choice of description - which leads to scientific anomalies at the very center of the theoretical foundation of this work.



(A) label 1

(B) label 2

FIGURE 1.1: **(A)**: **(B)**:

While the Greek Atomos theory correctly envisioned that the configuration of elementary particles played a role in a substance's properties, it failed to shed much light on the ways in which matter can influence matter. In the SM interactions are handled by bosons. In technical terms, Bosons are integer spin particles that obey Bose-Einstein statistics and are further divided into two sub-categories: **Gauge bosons** and **Scalar Bosons**. It is the gauge bosons that acts as **mediators** or **carriers** of the fundamental interactions described by the SM, which currently include:

electromagnetic interaction, mediated by the photon γ, the **weak interaction**, mediated by W and Z, and the **strong interaction**, mediated by gluons g. Quarks are also divided into different types; the whole zoo is depicted in purple in 1.1. Quarks, because they carry color charge, cannot exists as an individual, free particle and must therefore as minimum come as a pair. This effect is named **color confinement**. In addition, the energy associated with the color field between two quarks increase as a function of distance between the quarks in the pair, much like the energy deposited in a rubber band increases when it's stretched. When the quark pair is stretched sufficiently, the energy in the color field is converted to new particles, that didn't exists before the stretching. This mechanism is exploited in large scale at particle accelerators such at LHC at CERN to probe the fundamental building blocks of the universe[1].

However, in this work, the part of the standard model with highest relevance is the sub-categories of leptons, illustrated in green in Figure 1.1, here specifically the **muon**, **tau** and **electron** neutrinos. Below is a condensed review of the history of the neutrino - from it's theoretical inception to it's role in frontier research.

The Theoretical Inception

Henri Becquerel's discovery of radioactivity in 1896 is often attributed as the birth of nuclear physics. Soon after the discovery of radioactivity, Ernest Rutherford identified two different kinds of radioactive emissions, now known as β^+ and β^- decays, where a neutron in a radioactive nucleus is converted to a proton, and where a proton is converted into a neutron, respectively. In the early stages, the decay processes were studied as a two-body process, e.g:

$$p^+ \longrightarrow n^0 + e^+ \tag{1.1}$$

$$n^0 \longrightarrow p^+ + e^- \tag{1.2}$$

where n^o and p^+ are still bound to the nucleus, and the particle being emitted is the electron or positron. The assumption that the decays were a two-body problem directly suggested that the energy of the emitted electron/positron should take on a distinct value. To see this, consider a radioactive source X at rest that decays to Y and emits an electron:

$$X_z^a \longrightarrow Y_{z-1}^a + e^- \tag{1.3}$$

By conservation of energy, one has that

$$E_X = E_Y + E_{e^-} = (m_Y + m_{e^-}) \cdot c^2 + T_Y + T_{e^-} \tag{1.4}$$

Since the radioactive source X is at rest, this directly implies that

$$T_{e^-} = (m_x - m_y - m_{e^-}) \cdot c^2 - T_Y \tag{1.5}$$

Since the electron is much lighter than Y, it can be safely assumed that the electron would have the bulk of the released energy, which allows for the approximation $T_{e^-} >> T_Y$:

$$T_{e^-} \approx (m_x - m_y - m_{e^-}) \cdot c^2 \tag{1.6}$$

[1] This is also referred to as **jet physics**.

However, early experiments from 1907 and into mid 1910 suggested that the energy spectrum of the decay products were not discrete, as energy conservation would dictate[2], but that the velocity distribution of the decay products were continuous! [2].

The continuous spectrum puzzled physicist for the next two decades, where a wide variety of ideas were proposed. Niels Bohr suggested that perhaps the conservation of energy were only true in a statistical sense [3], but the bounds of the energy distributions ruled this out. At last German physicist Wolfgang Ernst Pauli suggested that perhaps the decays were not two-body problems at all but that there were a 3rd particle involved [3]. Simply by viewing the decays as three-body problems would allow for the electron/positron to take on a continuous energy spectrum - and by applying other conservation laws, one could characterize the properties of the elusive particle. By conservation of charge, it would have to be neutral - by conservation of angular momentum, the spin would have to be 1/2, and so on. In conclusion the particle theorized is what is today called the **Neutrino**, which had to be a massless particle that only interacts via the weak force[4].

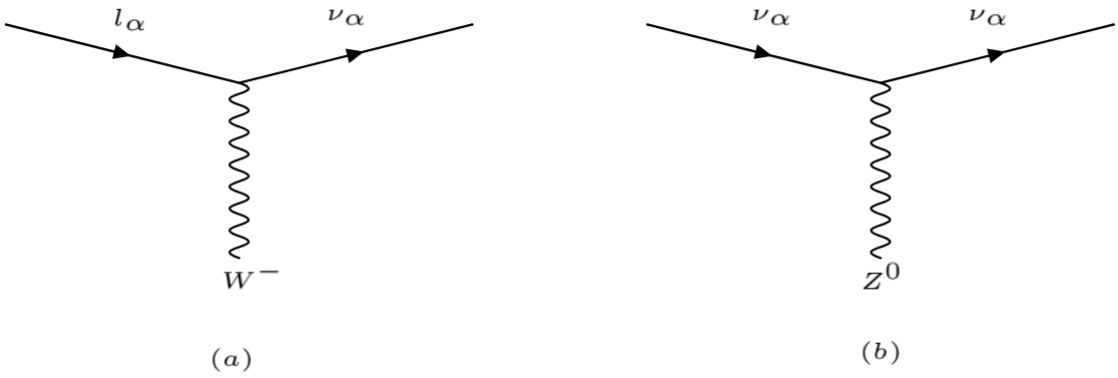


FIGURE 1.2: **(a):** The charged current (CC) interaction vertex. **(b):** the neutral current (NC) interaction vertex.

Such a characterization of it's interaction abilities means that, within the scope of the standard model, the neutrino will only be able to interact with the W and Z bosons in Figure 1.1. These interactions are in modern language represented by Feynman diagrams, an illustrative tool used to carry out calculations on particle interactions in quantum field theories. The fundamental interaction verticies of the neutrino is depicted in Figure 1.2, where α denotes lepton flavour. In experimental physics, one often distinguishes between **charged current** and **neutral current** interactions as these can gives rise to different detector readouts on a nucleus interaction level[4].

Finding the Neutrino

The Savannah River Experiment was conducted in 1956 by Clyde L. Cowan and Frederick Reines. The aim of the experiment was to test the neutrino hypothesis

[2]Energy conservation is not the only conservation law broken in the two-body assumption. Since spin conservation is broken, it's directly seen that angular momentum is not conserved in the process.

[3]This is interestingly the current interpretation of the quantum vacuum

[4]It's later been confirmed that the neutrino is indeed not massless as this would make it impossible for it to oscillate between flavours, which in turn means it also interacts via gravity.

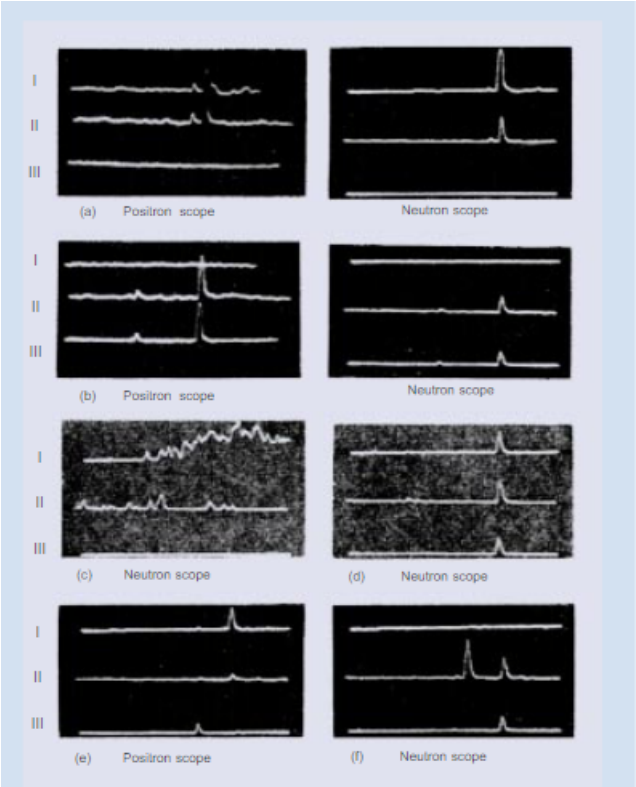


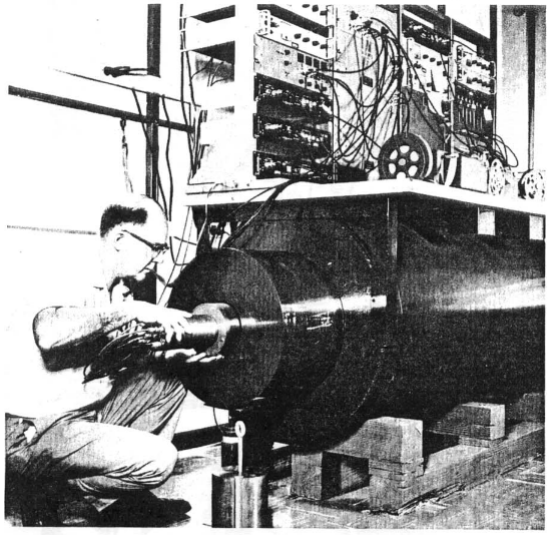
FIGURE 1.3: Oscilloscope readings from the Savannah River Experiment. The three detectors (representing the three scintillator tanks) are labelled I, II, III. a) and b) are examples of accepted signals. d) to f) represents rejected signals. Kindly borrowed from [5].

proposed to explain the continuous energy spectrum of beta decays. The detection method was indirect via inverse beta decay, where an anti electron neutrino collides with a proton to form a neutron and a positron, i.e :

$$\bar{\nu_e} + p \longrightarrow e^+ + n \tag{1.7}$$

The positron would quickly react with an electron, producing gamma-ray photons, which are detectable by **photo-multiplier tubes** (PMT's). The neutron could then bind to a nearby nucleus and produce a third detectable gamma-ray. The reasoning was then that if all three gamma-rays were detected within a short period of time, an inverse beta-decay must have taken place and therefore an anti electron neutrino must have existed.

After flirting with the idea of using nuclear weapon tests as a neutrino source for their experiment [5], Cowan and Reines eventually settled on using the nuclear reactor at Savannah River as their neutrino source. The experiments utilized water tanks with dissolved cadmium chloride, which is a compound that was added to better facilitate the neutron capture. The water tanks was placed between other tanks containing liquid scintillator, a liquid material that would emit light when struck by the gamma-rays. This light would then be captured by PMT's. The experiment kept running under a planned shut down of the reactor to check if the neutrino flux decreased. The results of the experiment was published in 1956, making it the first confirmation of neutrinos [6].



Dr. Ray Davis of Chemistry is shown placing a low level counter in a cut-down navy gun barrel which acts as a shield from stray cosmic radiation. This equipment is used in the Brookhaven Solar Neutrino Experiment.

FIGURE 1.4: Dr. Davis by the 12-inch naval gun

The Homestake Experiment

Brookhaven Solar Neutrino Experiment, also often referred to as The Homestake Experiment, was a solar neutrino experiment lasting from the late 1960's to the late 1990's. The experiment, headed by Raymond Davis, Jr. and John N. Bahcall, had the purpose of measuring the flux of neutrinos emitted from fusion reactions taking place inside of the sun. The experiment was located within a mine in South Dakota, nearly one and a half kilometers underground. Within the mine, a special tank containing approx. 460.000 liters of perchloroethylene, a dry-cleaning fluid rich in chlorine, had been placed. The neutrino detection was indirect in the sense that the chemical reaction

$$\nu_e + \mathrm{Cl}_{37} \longrightarrow \mathrm{Ar}_{37} + \mathrm{e}^- \tag{1.8}$$

allowed for counting the amount of detected neutrinos by counting the amount of argon-37 atoms in the tank. Given the elusive nature of the neutrino, namely that it's interactions with the chlorine atoms are dominantly via the weak force, means that even over a period of a few weeks of measurements, the amount of argon atoms in the tank was expected to be tiny, estimated to be between 1.5 and 5 neutrino captures pr. day. Increasing the sample time was difficult because the argon-37 atoms are radioactive with a half-life of approx. 35 days. To capture the argon-37 atoms, helium would be used to bubble through the tank to trap the argon atoms in a special charcoal filter that would then be analysed. The actual counting of the atoms was done by measuring the radioactivity of the sample, and to ensure the accurate counting, the counting device was placed inside of a 12-inch navy gun barrel to shield the counting process from background radiation.[7].

The Brookshaven Solar Neutrino Experiment is significant in the history of science as it was the first time that the solar neutrino flux as predicted by the *standard solar model* was tested and, more importantly, the experiment showed a deficit of

neutrinos by about one third compared to the theoretical value. By the time the experiment started, two different flavours of neutrinos had been experimentally found, the electron neutrino ν_e and the muon neutrino ν_μ. During the experiment, sometime in the 1970's, a third flavor was theorized, the tau neutrino ν_τ but it was not experimentally verified at the time.[5] Eventually the physical interpretation of the neutrino deficit was settled around the idea of **neutrino oscillation**, first theorized by Italian Physicist Bruno Pontecorvo 1957[8]. The conceptual idea of neutrino oscillation is that the state of any neutrino can be written as a linear combination of it's flavour states, and that the probability of measuring the neutrino as being in any one of those states oscillates as the particle moves through space. In effect this means that an electron neutrino emitted from the sun could be in any of it's three flavour states by the time it reaches earth, leaving experiments sensitive to only one neutrino state at a flux deficit. This was exactly the issue of the Homestake Experiment as the chemical reaction used to produce argon-37 could only be facilitated by electron neutrinos.

Neutrinos Oscillate!

The standard model of particle physics describes neutrinos as massless point particles. If one goes beyond the standard model and assumes that the neutrino does have mass [6], and that the mass of a neutrino is not definite but a linear combination of mass eigenstates [7] v_1, v_2, v_3, neutrino oscillation becomes theoretically possible. To explain the concept of neutrino oscillations and the implications it has in the simplest of terms, I will in the following focus on a two-neutrino system, and effectively neglect the third lepton multiplet. It is also assumed that the neutrinos are located in vaccum.

Let's denote two arbitrary neutrino flavour states as v_a and v_b. We can now write the flavour states as linear combinations of the mass eigenstates

$$v_a = v_i \cos \text{ø}_{ij} + v_j \sin \text{ø}_{ij} \tag{1.9}$$

$$v_b = v_j \cos \text{ø}_{ij} - v_i \sin \text{ø}_{ij} \tag{1.10}$$

where ø_{ij} is the **mixing angle**, which is a parameter that needs to be found experimentally. Suppose such these particles existed with momentum $\mathbf{p}$ at $t = t_0$, we'd have:

$$|v_a, \mathbf{p}\rangle = \mathrm{e}^{-\frac{E_i \cdot t}{\hbar}} |v_i, \mathbf{p}\rangle \cos \text{ø}_{ij} + \mathrm{e}^{-\frac{E_j \cdot t}{\hbar}} |v_j, \mathbf{p}\rangle \sin \text{ø}_{ij} \tag{1.11}$$

$$|v_b, \mathbf{p}\rangle = -\mathrm{e}^{-\frac{E_i \cdot t}{\hbar}} |v_i, \mathbf{p}\rangle \sin \text{ø}_{ij} + \mathrm{e}^{-\frac{E_j \cdot t}{\hbar}} |v_j, \mathbf{p}\rangle \cos \text{ø}_{ij} \tag{1.12}$$

When $t = 0$ the resulting state is considered a *pure* neutrino state, but as time evolves the pure state becomes mixed between the mass eigenstates. One can make the observation that the flavor states $|v_b, \mathbf{p}\rangle$ and $|v_b, \mathbf{p}\rangle$ are both related to the same mass

[5] It was discovered in 2000 in the experiment DONUT (Direct observation of the nu tau) hosted by Fermilab.

[6] which is experimentally verified to be correct

[7] This is usually referred to as neutrino mixing

eigenstates, which means that one can write a time-evolving neutrino state as a linear combination of it's flavor states, e.g:

$$|v_a, \mathbf{p}\rangle_{mixed} = A|v_a, \mathbf{p}\rangle_{pure} + B|v_b, \mathbf{p}\rangle_{pure} \tag{1.13}$$

By doing a little bit of algebra, one finds that:

$$A = \mathrm{e}^{-\frac{E_i \cdot t}{\hbar}} \cos \text{ø}_{ij}{}^2 + \mathrm{e}^{-\frac{E_j \cdot t}{\hbar}} \sin \text{ø}_{ij}{}^2 \tag{1.14}$$

$$B = \cos \text{ø}_{ij} \cdot \sin \text{ø}_{ij} (\mathrm{e}^{-\frac{E_j \cdot t}{\hbar}} - \mathrm{e}^{-\frac{E_i \cdot t}{\hbar}}) \tag{1.15}$$

The probability of measuring a v_b state is then

$$|B|^2 = \sin 2\text{ø}_{ij}{}^2 \sin\left(\frac{(E_j - E_i)t}{2\hbar}\right)^2 \tag{1.16}$$

here it's very clear that no oscillation can take place unless the energy associated with the mass eigenstates are different. By employing $E_i >> m_i c^2$ one can show that:

$$E_j - E_i \approx \frac{m_j^2 c^4 - m_i^2 c^4}{2pc} \tag{1.17}$$

This effectively means that the oscillation probability depends on the square mass difference, which is one of the reasons why estimating the neutrino masses is difficult. Since the particles are assumed to be in a vacuum, we can estimate t as $t = \frac{L}{c}$, where L is the distance travelled since emission. Also, in the ultra-relativistic limit we can assume $E \approx pc$. This together with Eq. 1.17 allows us to approximate the probability of measuring the mixed neutrino in state v_b as

$$|B|^2 \approx \sin 2\text{ø}_{ij}{}^2 \sin(\frac{L}{L_0})^2 \tag{1.18}$$

where $L_0 = \frac{4E\hbar c}{(m_j^2 - m_i^2)c^4}$. This shows that the oscillation can be written as a function of the length travelled since emission.
This effect is enhanced when the neutrino travels through a medium such as the earth [8]. Since the squared difference of the neutrino masses is very small, the characteristic oscillation length L_0 is quite big, typically several hundreds of kilometers, depending on the medium in which the neutrino travels [9] [9]. A more mature notation to describe the oscillation is

$$\begin{pmatrix} v_a \\ v_b \end{pmatrix} = U \begin{pmatrix} v_i \\ v_j \end{pmatrix} \tag{1.19}$$

where

$$U = \begin{bmatrix} \cos \text{ø}_{ij} & \sin \text{ø}_{ij} \\ -\sin \text{ø}_{ij} & \cos \text{ø}_{ij} \end{bmatrix} \tag{1.20}$$

[8] This gives rise to interesting oscillation bands. More on this in chapter 3.
[9] This is why neutrino detectors that wish to be sensitive to neutrino oscillation needs to be big

If we now upgrade the the flavor state to include three neutrinos: v_e, v_μ, v_τ with mass eigenstates v_1, v_2, v_3 Eq. 1.21 become:

$$\begin{pmatrix} v_e \\ v_\mu \\ v_\tau \end{pmatrix} = \begin{bmatrix} \cos ø_{12} & \sin ø_{12} & 0 \\ -\sin ø_{12} & \cos ø_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{1.21}$$

One may notice that the neutrino mixing matrix U takes the form of a generator of infitesimal rotation, a symmetry generator from the Lorentz group. It's important to mention that U still only describe the mixing of v_1 and v_2. With the introduction of a third neutrino, we require a matrix describing the rotation of v_2, v_3 and v_1, v_3. These can be found by repeating the steps taken above for those specific cases. This introduces two new mixing angles; $ø_{23}$ and $ø_{13}$. If one does this, one should find:

$$U_{12} = \begin{bmatrix} \cos ø_{12} & \sin ø_{12} & 0 \\ -\sin ø_{12} & \cos ø_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.22}$$

$$U_{23} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos ø_{23} & \sin ø_{23} \\ 0 & -\sin ø_{23} & \cos ø_{23} \end{bmatrix} \tag{1.23}$$

$$U_{13} = \begin{bmatrix} \cos ø_{13} & 0 & \sin(ø_{13}) \cdot e^{-ia} \\ 0 & 1 & 0 \\ -\sin(ø_{23}) \cdot e^{ia} & 0 & \cos ø_{13} \end{bmatrix} \tag{1.24}$$

here e^{-ia} represents a phase, which needs to be zero if neutrino oscillations should obey CP symmetry. A current theoretical idea for the matter-antimatter asymmetry in the universe evolves around CP symmetry breaking in neutrino oscillations, so it's left - as is - in this work [10]. Another thing to point out is that the amount of phases depends on description. If one takes the neutrino to be a Dirac particle, one arrives at the above result. If one takes the neutrino to be a Majorana particle, additional phases are introduced which plays a central role in neutrinoless double beta-decays [11], which is outside the scope of this work. Finally we arrive at:

$$\begin{pmatrix} v_e \\ v_\mu \\ v_\tau \end{pmatrix} = U_{12}U_{23}U_{13} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{1.25}$$

where $U_{12}U_{23}U_{13}$ is the **PMNS**-matrix, named after Pontecorvo, Maki, Nakagawa and Sakata, who first derived it [12]. The elements of the matrix needs to be determined experimentally, and are therefore of interest to neutrino experiments. When reduced it should contain 4 parameters $ø_{12}, ø_{23}, ø_{13}$ and the phase [10].

Sterile Neutrinos

At the very frontier of neutrino research lies the search for additional flavor states of neutrinos called **sterile** neutrinos. Their possible existence is well founded from a theoretical standpoint: firstly because all known fermions have both right and left handed chirality, but only left handed chirality for the current known neutrinos have

[10] Neutrino oscillations were first definitively confirmed in 1998 by the Japanese neutrino experiment SuperKamiokande

been observed. If neutrinos mixed with additional states carrying right handed chirality, the anomaly would be explained. However, if such a state exists and still remains hidden, it must be because it's not directly participating in the fundamental interactions described by the standard model. For this reason they're theorized to interact only via gravity, making them candidates for dark matter particles. Secondly, one may make the observation that the experimentally found mass of neutrinos are quite close to but not exactly 0 eV while the remaining fermions has larger masses by many orders of magnitudes. While the mass of fermions are given via the Higgs mechanism, certain extensions of the standard model uses sterile neutrinos together with the **seesaw mechanism**, to explain the observed neutrino masses. Several such seesaw models exists, each characterized by its assumed number of right handed neutrinos in existence, as the seesaw mechanism doesn't come with a natural constraint on the number of sterile neutrinos like electroweak theory has for ordinary neutrinos [13].

Chapter 2

IceCube Neutrino Observatory

From deep under the antarctic ice, not far from Amundsen–Scott South Pole Station, a telescope looks through earth, beyond cosmic clouds and into the universe.

Research Mission

The Collaboration itself spans over 400 physicists and covers everything from glaciology to cosmology. The following is a brief condensed explanation of *some* of the research being carried out in the IceCube Collaboration, selected after it's periphery relevance to this work [1].

Cosmic Alerts

Observations in astrophysics has historically been made by capturing light from light sources in the universe and analysing them. While that is largely the current method today, other methods now exists to examine astrophysical events via medium other than light. This includes gravitational waves from experiments like LIGO and neutrino telescopes such as IceCube. Because neutrinos doesn't interact electromagnetically, they are capable of leaving the wild inferno in supernova explosions before the light, creating a timely delay that can be exploited to capture the full supernova light signal by conventional telescopes. In effect, this make the neutrinos 'early warnings' of cosmic events. This is indeed an area of contribution for the IceCube telescope, as the experiment itself is a member of SNEWS (**Supernova Early Warning System**), which is a consortium of neutrino experiments around the globe. If a number of different SNEWS members observe cosmic alert candidates within the same 10-second timespan, and passes a series of quality checks, a message to the astrophysical community is transmitted containing the information required to redirect available telescopes[15]. One example of this is the 2017 IceCube-170922A alert identification of a high energy neutrino that was eventually discovered to originate from a blazar (TXS 0506 + 056) [16]. [2]

Point Source Search

Analogues to how conventional telescopes search the universe for stars and other light emission sources, neutrino telescopes scan the sky for point emission sources to get a 'picture' of the universe. In IceCube, neutrino maps of the sky are developed, which are similar to maps such as the cosmic microwave background. The maps can be constructed by observing neutrino flux over a long period of time and

[1] You can find the official research highlights here

[2] These Neutrino alerts are also available to amateur astronomers. More information here.

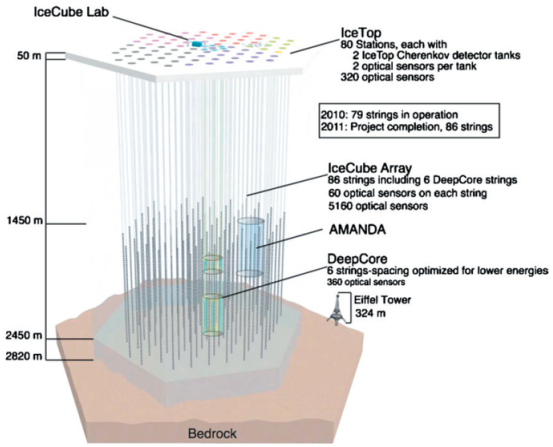


FIGURE 2.1: An illustration of the IceCube Detector array with a size comparison to the Eiffel Tower. [14]

reconstructing the angle from which the neutrino came. One emission source of particular interest is **gamma-ray bursts** (GRB), a short and very bright burst of gamma rays which is an astrophysical phenomenon not completely understood. By complimenting conventional observations of GRBs' with neutrino observations, one might better understand the emission candidates and the physics involved leading to the GRB's. Other possible sources for GRB's includes Active Galactic Nuclei (AGN) [17]. As of now, IceCube is most sensitive to the northern hemisphere because that direction is better shielded from atmospheric interference. Work has been done to extend the sky search to the southern hemisphere by including ANTARES, another neutrino experiment [18].

Dark Matter

The current conventional understanding of the matter composition of the universe shows that only approx. 15% of the universe is made of matter, leaving 85% unknown. This portion is referred to as **dark matter** and in recent decades many theories using WIMP's (**Weakly Interacting Massive Particles**) has been proposed as explanations to the full or parts of the mass deficit. Some of such proposed particles would be indirectly detectable by neutrino telescopes such as IceCube. One such theory proposes dark matter particles called **Kaluza-Klein** (KK) particles, whos annihilations should produce muon neutrinos. The proposed theory dictates that such annihilations should be present in the sun, and that an excess of neutrinos originating from the sun should be measurable. In 2009, IceCube analysed data taken over 104.3 days during 2007, and concluded that no excess was recorded [19].

Neutrino Oscillations

The mixing parameters from in Chapter 1 needs to be found experimentally. Generally, there's a variety of different methods and environments in which it's possible to observe neutrino oscillation - in IceCube the mixing parameters are estimated by observing oscillation induced patterns in the atmospheric neutrino flux [20]. IceCube is believed to be sensitive to atmospheric oscillations as low as approx. 5 GeV,

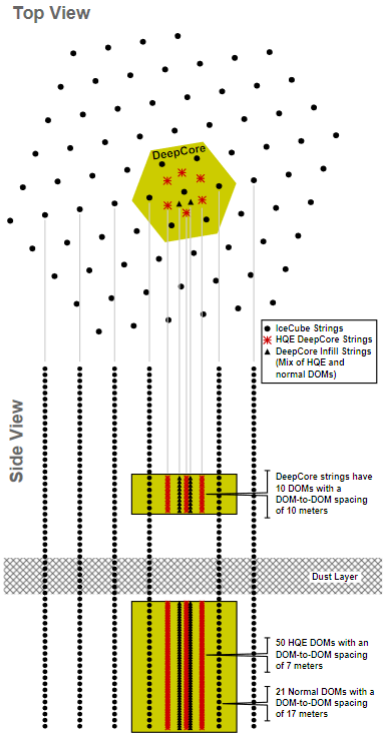


FIGURE 2.2: An illustration of the IceCube Detector DeepCore Section. Kindly borrowed from [21].

enough to produce one oscillation band in the neutrino flux. The planned upgrade of the detector is expected to increase sensitivity to higher order oscillation bands.

Current Detector Array

The current IceCube detector array consists of 86 strings arranged in a hexagonal pattern. A string is a 2820m long signal and power transmitting cable that begins at the surface of the antarctic ice and runs down to bedrock depth. Along a string, 60 digital optical modules (**DOM's**) are distributed, giving a total of 5160 DOMS. These DOMS measure **Cherenkov Radiation** using photo-multiplier tubes. 6 of these strings comprises the **DeepCore**, a section of the detector located centrally in the hexagonal pattern, at a depth of around 2100m. The DeepCore strings has a different distribution of DOMS and also contains High Quantum Effiency DOMS (**HQE DOMS**). A DeepCore string has 10 DOM's situated with a spacing of 10 meters right above the horizontal dust layer in the ice. Right under the dust layer a DeepCore string has 50 HQE DOM's with a spacing of 7 meters. DeepCore was installed and operational by May 2010 and were commissioned to increase the sensitivity of the detector by an order of magnitude, making it possible to detect neutrinos with energies as low as 10 GeV. The increase of sensitivity is due to a combination of several factors. Firstly because the density of DOMS in that area of the detector is now higher, secondly because the PMT's has a higher quantum efficiency, thirdly because the ice at this depth is very clear and lastly because of a clever scheme where the part of the detector that is not DeepCore (which is the majority of the detector) is used as a veto to filter out unwanted signal from cosmic muons [21].

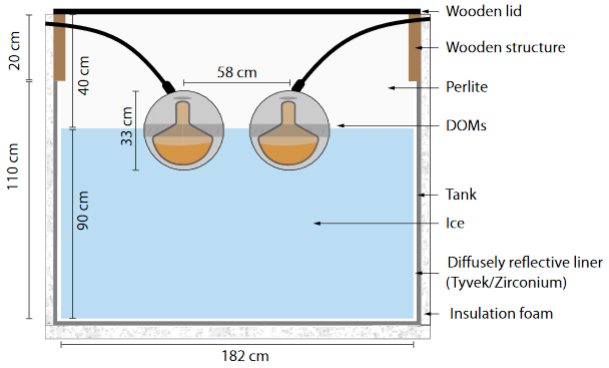


FIGURE 2.3: An illustration of an IceTop frozen water tank Cherenkov Radiation Detector. Kindly borrowed from [22]

At the top of the ice, a pair of frozen water tanks are installed in close proximity to each of the strings in IceCube, in total comprising what's referred to as **IceTop**, a surface Cherenkov Radiation detector that is both used to study air showers, here especially cosmic muons, but also as a veto against cosmic muons, which are unwanted signal in IceCube. Each tank contains two DOMs spaced 58cm apart.

In total, the IceCube array spans a volume of a cubic kilometer under the ice, making it the largest man made object by volume.

Planned detector upgrade

A further installation of 7 strings around DeepCore is planned for 2022/2023 arctic summer. The strings will in total carry around 700 detector modules, which is comprised of ordinary DOM's and two new DOM types: Multi-PMT Digital Optical Modules (**mDOM's**) and Dual Optical Modules (**D-Eggs**). The mDOM contain 24 PMT's distributed evenly around the mDOM's spherical shape. Having multiple PMT's in a single DOM gives a better sense of direction and provides a larger PMT area than that of the ordinary DOM [23]. D-Eggs contain two HQE PMT's, one facing up and one facing down, together with 12 LED's for calibration purposes.

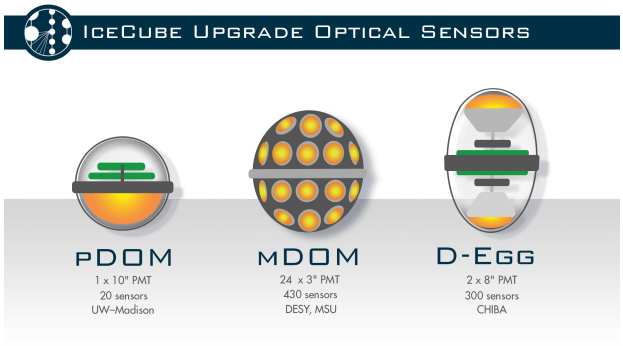


FIGURE 2.4: An illustration of the different DOM types present in the detector when the Upgrade is installed. Kindly borrowed from [24]

D-Eggs are expected to greatly increase the understanding of the hole ice - the ice in the drill holes where the strings are located [25]. This configuration is referred to as **IceCube Upgrade**, and is intended to increase the sensitivity of the detector to neutrinos with energies as low as a few GeV, and also to improve calibration capabilities. IceCube Upgrade is also a stepping stone towards a much bigger upgrade of IceCube called **IceCube Gen2**, that will increase it's volume to 8 cubic kilometers and double the amount of optical modules [26].

Chapter 3

Data From IceCube

The data from IceCube can generally be divided into two unsurprising categories: **real measurements** from the telescope and **simulated data**. Both data types are shipped to the collaboration in an internally developed file format called **.i3**. The i3 format is intended to be read using **IceTray**, an internal environment developed for analysis in the IceCube collaboration. While containing dependencies on the IceTray source code, several alternative ways of reading i3 files exists that circumvents the necessity of intimate knowledge of internal IceCube software. For reference to the reader that is unfamiliar with the i3 file format - the file structure once read resembles that of an ordered dictionary, where each principal key corresponds to either the original reconstructable particle attributes used in the simulation, or a detector readout after a given selection on DOM signal is made. Full documentation is available at [27].

Simulated Data

The simulated data originates from different **Monte Carlo simulations** (MC) implemented by the IceCube Collaboration. MC simulation is a broad umbrella term for different simulation algorithms that simulates probabilistic outcomes in systems where a deterministic approach is ill-suited. This way of simulating data is therefore ideal for particle physics as the interactions at a quantum level are probabilistic.

The inner workings of the simulations are the result of years of iterative improvement and therefore the reader is encouraged to seek these details from the internal documentation available for collaboration members, as a detailed walkthrough of these would constitute a thesis on it's own [1]. However, conceptually the simulations seek to answer the question:

Given a particle with energy E, direction of travel R and interaction vertex V, what readout will the detector produce?

To answer this question the simulation models not only the particle interactions from a theoretical stand point but also models experimental aspects that are vital to predict the detector readings. This includes accounting for the properties of the ice, as the clarity of the ice changes with the depth, and a dust-layer exists in the upper half of DeepCore, as mentioned in Chapter 2. The simulations also models the noise

[1] Technical papers can be found in [28], [29] and [30] but these algorithms are developed outside of IceCube and to understand the precise implementation in IceCube, one must see the internal pages.

from the detector - cases where DOM's register charge despite none being there. The following event generators has been used in this work:

- **MuonGun**: Simulation models atmospheric muons that makes it into the ice and it's interaction with the ice molecules [28].
- **CORSIKA**: CORSIKA stands for Cosmic Ray Simulations for Kaskade. The simulation tracks the propagation of the particle through the atmosphere and describes how it either decays or interacts with the air molecules, leading to what's referred to as an **atmospheric shower** [29].
- **GENIE**: Simulates the interaction between the neutrinos and the ice molecules [30].

An important point to make is: **To accurately represent all sources of signal for real measured data, a union of all (or some) of these simulations are required in a realistic distribution**, as each represents different sources. One of the inputs of the simulations are the spatial configuration of the detector. This information includes position of the DOMs and the type of DOMs - it acts in effect as a snapshot of the detector at the time of simulation. Therefore, there exists simulations for the current detector and the planned upgraded version. Because the planned upgrade includes new DOM types, the corresponding detector readout contains additional variables.

Event Variables

With the word **event** we associate a particle, either a muon or a neutrino, and its truth and feature variables. The truth variables represents the reconstructable attributes of the particle, such as energy and its direction of travel, whereas the feature variables correspond to the detector readout from that specific particle. In practice this means that every event has a single set of values with a fixed number of rows and columns that represents the reconstructable attributes, and that the feature variables represents the amount of DOMs that triggered and the information of each DOM, which produces a matrix with a variable amount of rows but a fixed amount of columns.

Feature Variable	Meaning	Upgrade Only	Data Type
dom_x	The x-position of the specific DOM		float
dom_y	The y-position of the specific DOM		float
dom_z	The z-position of the specific DOM		float
dom_time	The time at which the DOM triggered		float
dom_charge	The charge that the DOM measured		float
rqe	Relative Quantum efficiancy		float
pulse_width	width of pulse in ns		integer
pmt_x	x-unit vector of the PMT	✓	float
pmt_y	y-unit vector of the PMT	✓	float
pmt_z	z-unit vector of the PMT	✓	float
pmt_area	The surface area of the PMT	✓	float
pmt_type	The type of the DOM	✓	integer

TABLE 3.1: Table containing the most relevant input fields from .i3 event data files. These features are simulated values of IceCube detector readouts.

The positional variables **dom_x**, **dom_y** and **dom_z** are given relative to the detector, where as the unit vector variables **pmt_x**,**pmt_y** and **pmt_z** makes out the position of a specific PMT on the new DOM types that contain multiple PMTs such as D-Egg and mDOM. These unit vectors are given relative to the DOM, and not the detector. As we shall see later, the fact that each event has a variable amount of rows of feature data gives rise to certain technical issues that needs to be addressed. The feature data will always have the variables presented in Table 3.1 but the different selection methods will rule on which DOMs are included in the readout. This is a construction made since not all types of analysis requires all DOMs to be included, and certain selections represents noise reduction attempts.

Truth Variable	**Description**	**Data Type**
energy_log10	The logarithm in base 10 to the energy of the particle in GeV	float
position_x	The x coordinate of the interaction vertex	float
position_y	The y coordinate of the interaction vertex	float
position_z	The z coordinate of the interaction vertex	float
azimuth	The azimuth angle of the particle's interaction vertex as seen from a spherical coordinate transformation of its interaction vertex	float
zenith	The zenith angle of the particle's interaction vertex as seen from a spherical coordinate transformation of its interaction vertex	float
pid	The identity of the particle following the convention from [31]	integer
muon_tracklength	The length of the muon track	float

TABLE 3.2: Table containing the most relevant truth variables for an event for both the current detector and the upgrade. The **muon_tracklength** variable only exists in the cases where the particle is a muon.

The most relevant principal keys used in this work are:

- **MCInIcePrimary**: This field contains the truth variables depicted in Table 3.2
- **SplitInIcePulses**: This field represents an uncleaned selection of feature data depicted in Table 3.1.
- **SplitInIcePulsesSRT**: This field is the SplitInIcePulses features that has passed through a noise cleaning method based on special relativity, where causality of the signal is taken into consideration. This SRT cleaning method will not matter much for reconstruction on DeepCore data, but will play a big role in dealing with noise in the simulated IceCube Upgrade data, as we shall see.

IceCube Data Filtering Levels

The data selection used in this work is the data selection utilized and developed by the Oscillation Group within the IceCube collaboration, and the following is a conceptual condensation of their data selection, which is described in detail in [32].

Level 1 (Initial Trigger)

The initial data selection is collaboration-wide and quite unsurprising - it is simply a trigger that recognizes the readings as an event if it passes a **single multiplicity trigger** (SMT) check. The SMT labels a detector readout as an event if a multitude of DOMs register signal within a short window of time. At this stage the rate of events recorded is around 2500Hz.

Level 2 (DeepCore Filter)

This level removes any signal from level 1 that does not have more than 3 SMT's. The SRT cleaning algorithm is run on that selection, and a further causal selection is made to minimize the probability of the signal originating from a muon flying across the detector. At this level a veto mechanism as described in Chapter 2 is also applied.

Level 3 (MC to Real Data Agreement)

At this level derived features of the data that passed level 2 are calculated. These features are known to allow simple cuts to be applied to remove events where agreement between real data and simulated data are known to be poor. A list of these derived variables are available in [32] in table 7, page 28.

Level 4 (Boosted Decision Tree Classification)

This level applies **Boosted Decision Trees** (BDTs), which is a type of machine learning algorithm, to classify the data passing the level 3 selection into three categories:

- Muon Events
- Pure Noise Events
- Neutrino Candidate Events

Applying machine learning at this stage is possible because the level 3 data selection makes sure that any data passing into level 4 has good agreement between simulation and real data. The specific BDT algorithm used is *LightGBM*[2], and the model configuration used is available in [32] in table 9, page 34.

Level 5 (Removing Sneaky Muons)

Muons passing level 4 selection are either those who have been able to travel undetected through **corridors** in the detector into the DeepCore section, or given rise to signal outside the DeepCore section. At level 5 a **Containment cut** is made, effectively removing any events that are not in the DeepCore section of the detector. A further **Corridor cut** is made that limits the amount of sneaky muons. At this stage the sample is dominantly neutrino.

[2]This is an algorithm known for its speed and performance. Technical paper is available in [33]

Level 6 (High Statistic Reconstruction)

The data from level 5 is split into a high and low statistic batch, and the high statistic batch is reconstructed using **RetroReco**, a statistical model used for reconstructing the particle attributes shown in Table 3.2 using a likelihood approach that utilizes a table look-up method that takes approx. 5 - 40 seconds pr. event.

Level 7 (Final Cleaning)

BDTs are used to classify the same categories as in Level 4, but this time based on reconstruction results from Level 6. The final energy estimate is calculated after this. Event rates are available in Figure 3.1.

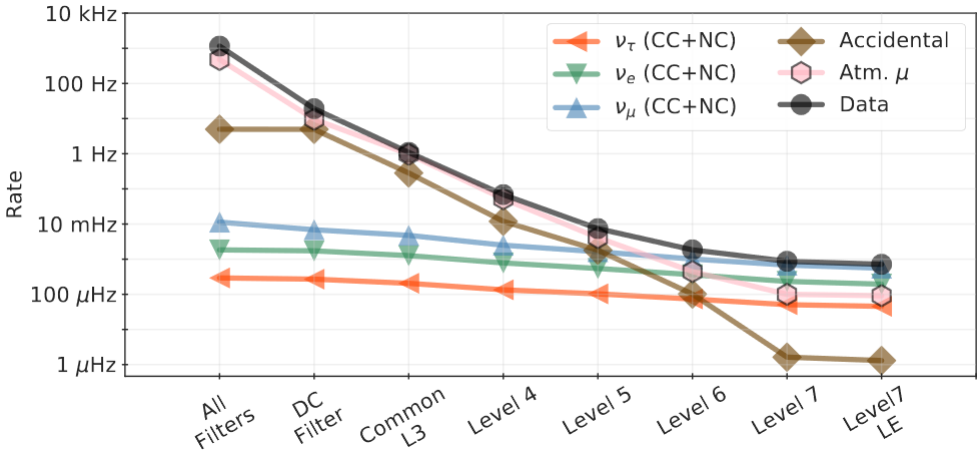


FIGURE 3.1: Event rates as a function of cleaning filters. 'All Filters' represents general Level 2 common to all collaborations and 'DC Filter' represents the Level 2 (DeepCore) filter described above. Figure is kindly borrowed from [34]

Another important point to mention is that these cleaning steps are developed for DeepCore data, and currently no well-established cleaning methods of same level of sophistication exists for the simulated Upgrade data, as we shall see.

Chapter 4

Machine Learning

The debate on Artificial Intelligence, of which Machine Learning is officially considered a sub-field, suffers like many other debates seem to do in these times, from an urge to categorize it as either 'good' or 'bad'. There are those that envision a future characterized by E.M. Foster's 1909 novella *The Machine Stops* in which he writes

"The Machine develops — but not on our lines. The Machine proceeds — but not to our goal. We only exist as the blood corpuscles that course through its arteries, and if it could work without us, it would let us die."

And then there are those that trust The Machine and it's architects to such an extent that they're willing to let it replace something as sensitive as a criminal sentencing process [35].

Those who approach the debate more objectively might see that Machine Learning is a specific way of analysing data, and that forcing such a categorization upon a tool makes as much sense as discussing the morality of a hammer. Any morality must inherently be tied to its intended application. From a utilitarian perspective, the lowest ranking application of Machine Learning (**ML**) could very well be the Chinese mass-surveillance program that utilizes ML techniques such as Facial and Speech recognition together with a wealth of private information to compute a Citizen Score, that seems to determine whether or not your children may attend higher education, or whether or not you may board a train or an air plane [36]. In the other end of the utilitarian spectrum, one might find the 2020 breakthrough by DeepMind in prediction of protein folding, a notoriously difficult problem for medical sciences, as the folding of a protein highly characterizes its medical properties. Being able to predict the folding of a protein by simply knowing it's amino-acid sequence and structure, researchers will be able to better understand disease and dramatically improve the time it takes to produce new medicine [37].

Somewhere between the two utilitarian extremes lies a ML-based classification and reconstruction of neutrinos in the IceCube detector. Determining exactly where is left as a exercise for the reader.

Supervised and Unsupervised Learning

In the broadest of terms, the usage of Machine Learning can be divided into **supervised** and **unsupervised** learning. Exotic mixtures do exist [1]. Unsupervised learning

[1] This is typically called semi-supervised learning. This is actually quite common for graph neural networks

covers the cases where a general labeled structure of the data isn't known. It can conceptually be thought of as the answer to a question of the type:

Given this box of 1000 items, if you were only to divide these into N different categories, which category would you assign each item?

While human interpretation of such categories can be quite difficult, it is common among data scientists beginning a ML analysis to use unsupervised learning to check whether or not there even exists structure in the data, which is a prerequisite for any further analysis. In this sense, it's a method usually most practical for an abstract categorization of data, and not suitable for any predictive tasks. Supervised learning covers the cases where a labeled structure in the data is known, and of such a *learning from example* scheme can be implemented. Again in broad terms, supervised learning can be divided into two further sub-categories:

- **Classification**
- **Regression**

Common among these is the learning from example scheme. They first *learn* from examples that you present, and then they grow capable of applying that experience to examples they have not yet seen.

Classification is a ML task that categorizes data according to categories that it has been taught. A typical example of this is image recognition. If a given algorithm is presented with a 1000 pictures of either a cow or a cat, and you specify which is which for every picture, then if you present it with a picture it has not seen before, it will attempt to classify it as either a cow or a cat[2].

Regression is a ML task that closely resembles the conceptual foundation of regression analysis. In the examples on which such an algorithm trains, one must provide not only the data but also the known value representing the truth. Suppose you for times $T = [0, 1, 2, 4, 5]$ measured $X = [0, 1, 4, 16, 25]$, and you wanted to apply ML regression to X given T. Effectively this would let the algorithm learn the function $f(X) = X^2$ as defined on the domain given by T. After the learning one could present the algorithm with T = 6 and it would attempt to forecast $f(6)$. This is, obviously, a completely unnecessary exercise for cases where the analytical expression for a given problem can be achieved, but it's very powerful for complex systems where the analytical expression is unknown. It's also worth noting that in cases, especially for simulations of fluid dynamical systems, where an analytical or numerical solution to the problem exists already, serious computational advantages exists with using ML to regress the problem instead of using a simulated result. This is due to the fact that evaluating an already trained regression model beyond it's training domain e.g asking for $f(6)$ is often much quicker than solving a complex system numerically [3] [39].

[2] And it does so pretty well. But if you present it with a picture that falls outside the categories on which it has trained, perhaps a picture of your girlfriend, you better hope it picks the cat category!

[3] This is actually how most 4K rendering is done in graphics cards. Newer generation GPU's from Nvidia renders the images in a lower resolution and uses AI to predict how that image would look like if it had actually rendered it in 4K, and then displays that. The method is called **Deep Learning Super Sampling** [38]

Artificial Neural Networks

In the Chapter 3, I briefly mentioned **Boosted Decision Trees**, a category of ML algorithms. There exists many different categories, too many to address in this work, and for this reason I will here only briefly explain the central idea of one such category, Neural Networks - more specifically **feed-forward multi-layered perceptron** neural networks, as it will become relevant later.

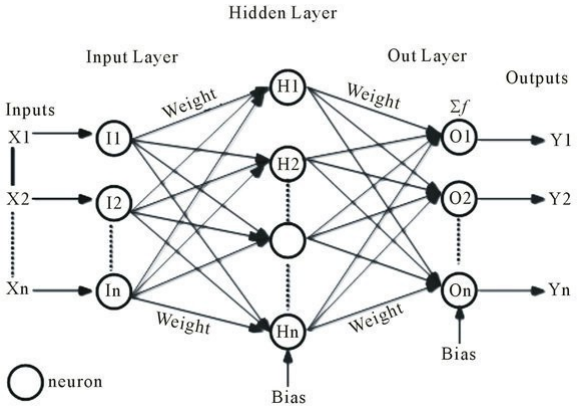


FIGURE 4.1: An illustration of a MLP with a single hidden layer. Data flows from left to right. As depicted by lines in the diagram, every neuron in one layer is connected to all neurons in the next layer. For this reason this architecture is often called **fully connected**. Kindly borrowed from [40]

Artificial Neural Networks began as a joint venture between Warren McCulloch, a neurophysiologist, and a mathematician named Walter Pitts who in 1944 published [41], in which they proposed how the neural workings of brains could be artificial modelled in computers. After decades of iterations and different neural architectures, the feed-forward multi-layered perceptron neural networks, or just simply **MLP**, is the most dominant today. As illustrated in figure 4.1, the architecture consists of neurons, the circular objects, which are arranged in an input layer, a hidden layer and an output layer. Each layer of neurons receives data and outputs, as depicted in the diagram, the results of an activation function, typically given by

$$O_L = f(b_L + w_L \cdot O_{L-1}) \tag{4.1}$$

where O_L is the output of the L'th layer, f is the *activation function*, typically the **sigmoid** activation function, w_L is the weights of the current layer and O_{L-1} is the output of the previous layer. The feed-forward mechanism simply means that the output of one layer flows into the next layer as input. Both b_L and w_L are *learnable parameters*, variables that are tweaked in order to minimize the **loss** of the output of the output layer. Minimizing the loss is essentially the only thing any supervised ML algorithm does. The process of tweaking these parameters is called **backpropagation**.

Backpropagation, Loss Function and Optimizers

Eq. 4.1 is applied iteratively throughout a network until an output is calculated. This is referred to as a *forward pass*. First time this is done, typically the learnable parameters used are some initialized values that is most likely far from the ideal parameters obtained after lots of training. But how are the variables tweaked? Consider a case where we've completed the very first forward pass in the regression of X given T, as mentioned earlier, and now have a single value from the MLP as a result. Since this is supervised learning, we have some true value to compare the output to. Since this is the first pass, T = 0 and the true value is X = 0. Suppose our network outputs $\tilde{X} = 3$. This result is far from correct, and the way we can specify this to the algorithm is to quantify ideal. The way to do this is to specify a **loss function**, a function $L : \mathbb{R}^d \longrightarrow \mathbb{R}^1$ which we then ask to keep the value of as low as possible. A common loss function chosen for regression problems is the **mean squared error** (MSE):

$$Loss = \frac{1}{n}\sum_{i=n}^{n}(X_i - \tilde{X}_i)^2 \tag{4.2}$$

where n is the number of outputs of the algorithm. In our case n = 1, so our loss is $Loss = 9$. Now that we calculated the loss, we need to perform **backpropagation**, which is to move backwards in the diagram in Figure 4.1 and update the weights and biases in each layer such that the loss gets smaller. The scheme of updating weights are called **optimizers**, and there exists many, each with their own advantages. In this particular case, let's pick a simple **Gradient Descend**. This is typically done as:

$$w_L = w_L - k \cdot \frac{\partial Loss}{\partial w_l} \tag{4.3}$$

$$b_L = b_L - k \cdot \frac{\partial Loss}{\partial b_l} \tag{4.4}$$

where k is the **learning rate**, a scalar parameter for each learnable parameter that dictates how big of a nudge the weight and bias gets on every backward pass. Since Loss depends on $\tilde{X}$, which in turn depends on *every other learnable parameter in the model*, the partial derivatives in Eq. 4.4 and Eq. 4.3 requires the chain rule, and grows to quite inhuman complexity quickly as the number of hidden layers, usually referred to as **depth** and number of parameters in each hidden layer, usually referred to as **width**, increases.

Once the weights are updated one would expect the next forward pass to carry a lower loss. A complete pass of the entire training data T is referred to as a **training epoch**, and typically one would require multiple training epochs to achieve a global minima in the loss landscape. Defining a loss function that suits the problem that the ML algorithm is being applied to is quite important. Take for instance a dumb loss function $F = X - \tilde{X}$. Because the job of the algorithm is to minimize the loss, the algorithm would simply output larger and larger $\tilde{X}$, as this would produce a smaller and smaller loss, and as the loss gets smaller, the actual goodness of the output would become worse and worse. Therefore, good loss functions tends to have a lower bound.

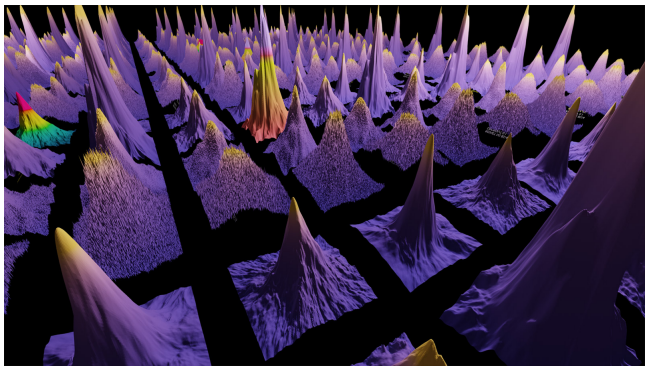


FIGURE 4.2: An illustration of loss landscapes. Each landscape represents a slightly different model architecture. Each point in a landscape represents a loss function evaluated at a specific set of weights and biases. Shows well how cumbersome ML prototyping can be!

Validation and Early Stopping

If we gave the simple algorithm above infinite training time, eventually it would reach a perfect loss on the domain in which it's asked to train. Effectively the loss in Eq. 4.3 would reach zero. The point of supervised learning is not to perform well on training data as the labels of such is already known. The goal is to perform well on a new subset of data on which the labels are unknown. If we were to present the algorithm with new labeled data, a so called **validation set**, the loss would most likely not be zero. This is due to **overfitting**, a common problem in ML, as models

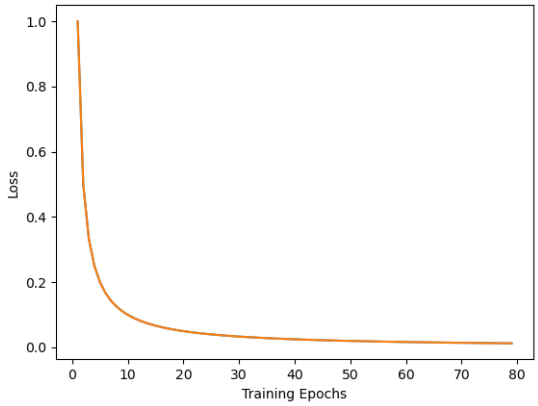


FIGURE 4.3: An illustration of Loss as a function of training epochs. Loss plots are not often as beautiful as this fictitious illustration makes it look!

often pick up variance and bias only present in the sub-sample of data they train on, and therefore expects this in the new sample. One great source of bias is obviously the initial data selection - the training data needs to be a randomized sub-sample of the total data set. Assuming this is the case, a common practice is to use a validation set actively - another sub-sample of the data set, on which a validation loss is calculated. While validation schemes can be different, the one used in this work is to evaluate the model on a validation data set after every training epoch. If the

validation loss is lower than the one previously recorded as the lowest, the specific weights of the model is saved. The idea of **early stopping** is then to stop the training once the validation loss stops improving, which is at the point where the training and validation loss begins to diverge. One can apply this logic in a so-called **k-fold validation** scheme, where the entire labeled data set is split into k parts, and where the designation of validation and training set is rotated such that one eventually have results that have trained and validated on the entire labeled data set. One would then average the validation performance to get a gauge on how well the model generalizes.

Test Set, Generalization Error and the Bias-Variance Trade off

A k-fold validation does not guarantee that performance on labelled data will be close to performance on unlabeled data. Since labelled data is typically a sub-set of all data, any error estimates made on data with labels should be considered a biased estimate on a **general** error. Mathematically, the general error J of a function f can be expressed as:

$$J(f) = \int L(f(x), \tilde{x})\rho(x, \tilde{x})dxd\tilde{x} \tag{4.5}$$

where x denotes training data, $\tilde{x}$ represents the labels and L is the user-specified loss function. $\rho(x, \tilde{x})$ is a **joint probability distribution** which describes the probability of finding a pair $(x, \tilde{x})$ in a specific range or discrete set[4]. However, typically ρ is not known in ML problems, and one can then regroup and calculate the **empirical loss**

$$J(f)_n = \frac{1}{n}\sum_{i=n}^{n} L(f(x_i), \tilde{x_i}) \tag{4.6}$$

This is a typical way of calculating loss on an epoch level, and as such $J(f)_n$ represents our validation loss from above. An obvious but important observation to make is that neither of the equations above are defined for domains in which there are no labels. This is another reason why general error estimation in ML is difficult. A practical workaround is to claim that a model is general if

$$\lim_{n\to\infty} J(f) - J(f)_n = 0 \tag{4.7}$$

A common approximation to $J(f)$ is then made by introducing another sub-sample of the labelled data called the **test set**, from which one can calculate the **generalization error** G(f)

$$G(f) = J(f)_{\tilde{n}|_{test}} - J(f)_{n|_{validation}} \tag{4.8}$$

G(f) serves as a gauge on how predictions made on unseen (not unlabeled) data differs between unseen data sets. In practice, one can only hope to reduce this error down to level of noise present in the data set [42]. Certain alternatives to the choice of approximating J(f) with an empirical loss of a test set exists. Instead one could attempt to characterize $G(f) = J(f) - J(f)_{n|_{validation}}$ by producing bounds. Hoeffding's inequality [43] is one such method, that eventually gives rise to what's known as the **Bias-Variance Trade off**, where it's seen that one can decrease the variance of the outputs of a model by increasing it's bias. In practice this often translates to that

[4] In the case where the variables are in-dependant, the joint probability distribution simply becomes the product of the marginal (unconditional) probability densities.

models of higher complexity will generally have a lower probability of generalizing well [44].

Chapter 5

Graph Neural Networks

The first steps towards Graph Neural Networks (GNNs) were taken in the 1997 paper [45] to address the challenge of incorporating any specific information about the relationship between components in data in ML. As it turns out, relational information can be the major source of information in certain problems [46]. Examples of such are social, transportation and telecommunication networks where the non-Euclidean relations in the data cannot be ignored. Most of the techniques known to work well on Euclidean data have since 1997 been included in the GNN domain - this includes auto-encoders, temporal, recursive and convolutional models [47].

A (Very) Brief introduction to Graph Theory

The 'Graph' part of Graph Neural Network has its meaning from a mathematical field called **Graph Theory**. The following definitions are from [47].

A **graph** $G = (N, E)$ [1] is a set of **nodes** N and **edges** E. A node $n_i \in N$ can be connected to another node $n_j \in N$ via an edge $e_{ij} = (n_i, n_j)$ that starts at n_j and ends at n_i[2]. The **neighbourhood** of a node n is given by $f(n) = \{k \in N | (n, k) \in E\}$. The **adjacency matrix** is a matrix A where elements $A_{ij} = 1$ if $e_{ij} \in E$ and otherwise zero. One can extend this definition to what is usually referred to as a **attributed graph**, where one associate **node features** X with every node, and **edge features** X^e with every edge. All graphs in this work is attributed graphs. A graph is a **(simple) directed graph** if every node is connected once and not with itself. A graph is called **Spatial-Temporal** if the node features X changes dynamically along a time-like parameter t.

For all intends and purposes of this work, one can get away with thinking of graphs as a series of circles and vectors. As illustrated in Figure 5.1, every circle is a node that effectively represents *some* data (formally node features), and those nodes are connected to other nodes via vectors that originates from one node and ends and another (formally edges). The connection itself is information, and together with the connection one can associate other data (formally edge features). In Graph Theory, one differentiates between methods of extracting information from graphs. Technically, the methods deviate on how the Laplacian of the graph is interpreted. The Laplacian of a graph is defined as the difference between the adjacency matrix of the graph and the **degree matrix**, a matrix containing information about the number of

[1] Some defines a graph as the set $G = (N, E, A)$, where A is the **adjacency matrix**. This work excludes the adjacency matrix in the formal definition as any information present in the adjacency matrix is already present in edges.

[2] This is also called a **directed edge**. An undirected edge is simply in the case where you have directed edges pointing in opposite directions, and is therefore not included here.

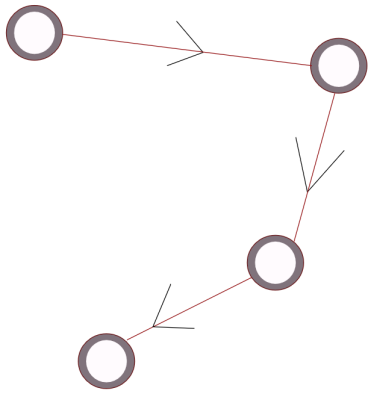


FIGURE 5.1: An illustration of a simple directed graph.

edges each node has. In essence, the **Spectral** domain characterizes the graph by an eigen-decomposition of the Laplacian as developed in **Spectral Graph Theory**, this is analogous to a Fourier decomposition in signal analysis. This breaks down the graph to subgraphs called graphlets, giving rise to models such as [48], and effectively carries out computation in a pseudo frequency space [49]. The **Spatial** method treats the Laplacian as a measure of the spatial connectivity of a graph, and does not decompose the graph via a graph Fourier transform. This is the approach used by geometric graph neural networks, such as point clouds. Details on the full mathematical rigour of topological graphs is available in [50].

A current topic of research is to better understand the connections between results obtained in either domain. [46] presents a framework from which certain results can be interpreted as equivalents between the domains.

Message-Passing Schemes

Common for all GNNs is that the input of the models are graphs. As data from the wild rarely comes pre-packed as graphs, one typically have to transform data into a graph format and specify the edges between the nodes. Once the graph exists, a GNN acts as a function on the graph, in the same sense as an ordinary artificial neural network can be seen as a function acting on its input. The GNN itself can be viewed as a series of applied operations, that eventually produces the output. The difference between a Convolutional GNN and an Attention GNN is the operations on the graph itself.

A general framework in which to interpret how these operations work on graphs is to view them as **Message-Passing Schemes**. In broad terms, every node in a graph receives a message from its neighbours that is then used to update the node features of the node receiving the messages. To see this, consider a single node a_i that receives a list of messages m_i from all or some of its neighbours n_i. The message can be defined as

$$m_i = f(n_i)W \tag{5.1}$$

where W represents a matrix of learnable weights, and f some operator-specific function on the node features of the neighbours of n_i. These messages are then aggregated according to some operator-specific way:

$$\tilde{m}_i = Aggr(m_i) \tag{5.2}$$

The node features of a_i are then updated as:

$$a_i = G(\tilde{m}_i) \tag{5.3}$$

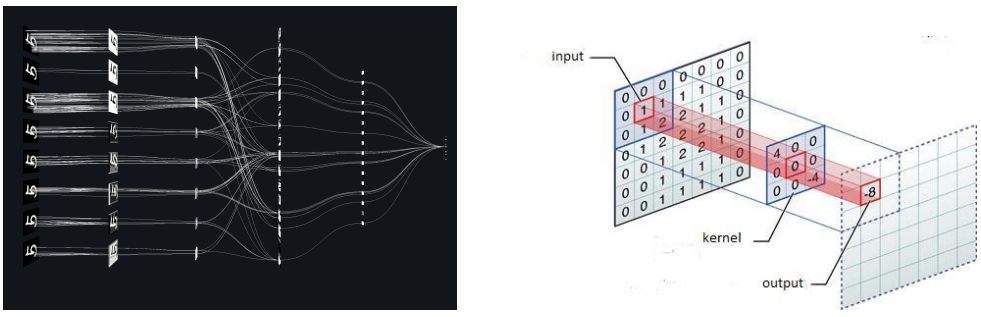
where G, again, is some operator-specific function. Note that this little example doesn't include edge features - some operators do, but the underlying scheme is the same.

Additional Learning Goals on Graphs

Certain interesting possibilities on learning goals arises when using GNNs. One such learning goal is relational prediction, or **link prediction**. Instead of training a model to predict a certain value for a certain problem, GNNs can be trained to output a graph representing a state of a social network sometime in the future, effectively predicting new edges in the graph [51]. This could represent a prediction on a new friend on your favorite social media platform, or a prediction on your next purchase on a very dominant e-commerce platform. Another exotic example is **semi-supervised clustering**, where only parts of the nodes in a graph carry labels. Message-passing schemes can be utilized to predict how labels flow from labeled nodes to unlabelled nodes, effectively creating a pseudo-classification model that [52], in principle, can mix simulated and real data in HEP.

Graph Neural Networks as a Generalization of Convolutional Neural Networks

Convolutional Neural Networks (**CNNs**) are popular tools in fields such as machine vision, which includes technologies as Facial and fingerprint Recognition. The effec-



(A) label 1 (B) label 2

FIGURE 5.2: **(A)**: An illustration of a CNN architecture used in image recognition of handwriting data. The handwritten letter 5 is given as argument to a convolutional operator that produces a feature-extracted version of that image (second column), which is then passed to an MLP that then produces a categorization of in the input but based on the feature extracted version of the input. **(B)**: An illustration of a convolution operation. Kindly borrowed from [53].

tiveness of CNNs comes from the convolution and pooling scheme that is deployed before feeding the information to an ordinary MLP. A convolution operator applies a **filter**, also known as a kernel, to the input data. The filter slides across the entire input and extracts a feature map. The specific output on the feature map is typically the sum of an element-wise matrix multiplication. The specific filter used can vary. The convolution operator is then in a sense an operation that examines a local area of an image and extracts local features. The **pooling** is effectively an aggregation of the feature maps. It carries the benefit of introducing invariance to small translation in the input data. Generally, a pooling operation divides the input into batches that then gets aggregated via a non-linear mapping. A common type is **max pooling**, where the non-linear map just outputs the maximal value of each batch [54].

There exists an underlying assumption on the way the input data is structured for CNNs. Because CNNs are developed to primarily analyze image data, the input is expected to come in the form of an image, typically a matrix, where regularity in the spacing of the pixels are taken for granted. If a CNN is being applied to a problem that doesn't have this regularity in it, the data must be altered such that this underlying assumption fits. This is not the case for GNNs. GNNs carry no inherited assumption on the structure in which the input data comes - it asks you to specify this explicitly. The structure is specified in GNNs by specifying the relational information between nodes (the edges) in the graphs. In this sense, Convolutional Graph Neural Networks are a generalization of CNNs, as one can simply arrange the structure of the input graphs in such a way that it effectively becomes image data, effectively creating a CNN. This can be done in the spatial domain by simply representing every pixel as a node with RBG values as node features. Every pixel would then be connected to its nearest pixel.

Chapter 6

Model Development

In the following I will attempt to describe the process of developing the GNN model to regress and classify on IceCube data. If you're simply interested in seeing the final model, see the last section.

The process of development has been long, as the vast majority of time spent on this work has been spent here. Including every thought is impossible, and therefore only the most relevant choices and considerations are highlighted here.

Initial Thoughts

Early on I became painfully aware that the vast majority of GNN papers were focused on classification problems in unrelated fields. Out of an already limited amount of published papers on ML-driven reconstruction in IceCube, an even smaller subset of these revolved around GNNs. In fact, all I was able to find that involved GNNs on IceCube data were [55], which is focused on classification. Widening the net to include larger HEP collaborations, such as CERN, introduced me to [56], which indicated that reconstruction using GNNs in the spatial domain might be possible in IceCube. But generally, a clear picture was emerging - namely that reconstruction and classification of particles in HEP were typically done using deep learning algorithms such as CNNs, one example of this is [57], and that GNN classification in HEP seemed to be an active area of research, with little results to be inspired from. Therefore, it was initially an open question to whether or not a GNN could even produce reasonable reconstruction results. I therefore set the initial goal of attempting to produce a reasonable regression on $energy_{log10}$ in Table 3.2 [1].

Choosing a Library

At the beginning of this work, a comparison between popular libraries such as pytorch geometric [2], deep graph learning and stellar graph, showed, at least at that time, that a larger number of implemented operators and models from papers on GNNs existed in the pytorch geometric. It was relayed through consultation with members of the Danish Institute of Computer Science (**DIKU**) that pytorch geometric offered the best computational performance on GPU - and knowing that computational speed were central to the relevance of any reconstruction made in this work,

[1] I much later realized that judging a GNN approach solely from its ability to reconstruct energy might not be a very fair criteria, as the GNNs ability to capture non-Euclidean aspects of the IceCube data might be better evaluated in reconstruction of the zenith angle in 3.2

[2] A larger list is available here

made it seem like the better choice at the time. As seen in 6.1, this is still the case for GPU-based calculations - but not for CPU!

Compute	Framework	Ave. time per epoch (ms)			Test accuracy		
		Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed
CPU	PyG	104.4	182.9	798.5	0.717	0.696	0.718
CPU	DGL	71.3	153.4	338.6	0.785	0.710	0.723
GPU	PyG	7.7	8.4	10.9	0.796	0.720	0.718
GPU	DGL	13.3	12.4	12.5	0.721	0.641	0.682

FIGURE 6.1: Computational performance benchmark from December 2020 between **pytorch geometric** (PyG) and **deep graph learning** (DGL). Results are calculated based on computation unit (either CPU or GPU) on popular datasets. Kindly borrowed from [58].

As fellow students with experience with the torch library pointed out, pytorch geometric is a library with little in-built convenience. It does not contain the modularity that libraries like Tensorflow offer, and that this increased degree of self-reliance could very well result in hours spent being stuck in trivial coding errors. However, having the confidence of a 25-year old, I decided to settle with pytorch geometric.

Preprocessing

Initially, only simulated data from the current detector setup at IceCube were available to me. Therefore this data was used in prototyping. The data were delivered in the shape of sqlite databases that contained already preprocessed events. All data were prepossessed using the

```
sklearn.preprocessing.RobustScaler(unit_variance = True)
```

which is a common preprocessing transformer from the scikit-learn library. It transforms data as

$$\tilde{x_i} = \frac{x_i - median(x)}{IQR(x)} \tag{6.1}$$

where x_i is the i'th element of the set of observables x and IQR(x) is the *inter-quartile range* of x. As it will turn out, the unit variance of data as a result of this transformation will be import in the discussion of **batch normalization**.

The sqlite databases used for the first half of this work originates from a i3-to-sqlite pipeline that **Mads Ehrhorn**, a now former master student from NBI, developed as part of his master thesis. Having access to this early on meant that this work didn't have to begin with long considerations on preproccessing steps and deciphering the not-so well documented internal IceCube software [3]. The pipeline offers a variety of convenient features - among the most practical is the ability to order data from a central archive. In such an order, one can specify the desired level of filtering

[3] Source code for the pipeline named **CubeDB** is available here

of the events, the simulation origin, and the particle type. The pipeline also incorporates the **event number**, a unique event identifier across i3-files, which is used to pull data out of the databases in a very computationally efficient way. Generally a database contain two fields: **features**, which contains the variables listed in 3.1, and **truth**, which contains the variables in 3.2. One can then pull specific events from the database **without loading the whole database into memory**. Suppose one wanted to extract event 1001 from a database - such a request could be done using:

```
import pandas as pd
import sqlite3

db_file = "mydbfile.db"
desired_event = 1001
with sqlite3.connect(db_file) as con:
   truth_query   = 'select * from truth where  event_no == %s'%desired_event
   truth         = pd.read_sql(truth_query, con)

   feature_query = 'select * from features where  event_no == %s'%desired_event
   features      = pd.read_sql(feature_query, con)
```

LISTING 1: Simple query example

which extracts the features and truth data associated with event 1001 in a pandas format. Similar requests can be made to extract events based on particle identity. As seen in 6.2, even extraction of a small number of events become efficient. Usually extraction times pr. event become very computationally inefficient when the number of events extracted is small. This is due to the fact that there is a minimum overhead of computation associated with opening files from disk and loading them into memory. While utilizing the i3-to-sqlite pipeline does add another step to the data flow, this step only have to be done once, and as evident from 6.2 one can then read the events from the databases directly during training without creating a bottleneck. Reading directly from i3-files is not a suitable alternative, as the i3-file format requires sequential reading. Alternatives to sqlite databases does exist, such as a i3-to-numpy pipeline[4], but while reading numpy arrays from disk is quicker than reading i3-files, it does not provide a comparable efficiency to sqlite databases.

Representing IceCube Data as Graphs

With the ability to designate relational information in IceCube data by representing it in terms of graphs, one naturally gets to wonder how to do this best. In the domain of GNNs, no general method exists other than the edge configuration of the graphs should depict *some* information that is not apparently embedded in the array-format in which the data comes. If one finds such a configuration, a model utilizing the connection between nodes in a computation, should perform better on an information-rich edge configuration compared to an edge configuration that is completely random. This gives rise to a method of testing the graph configurations, but requires a working model first. So in order to first build a working model, one needs to work from an ansatz.

The work presented in [56] suggested that an edge configuration of graphs based

[4]This is offered by the unofficial IceCube package i3cols, code available here

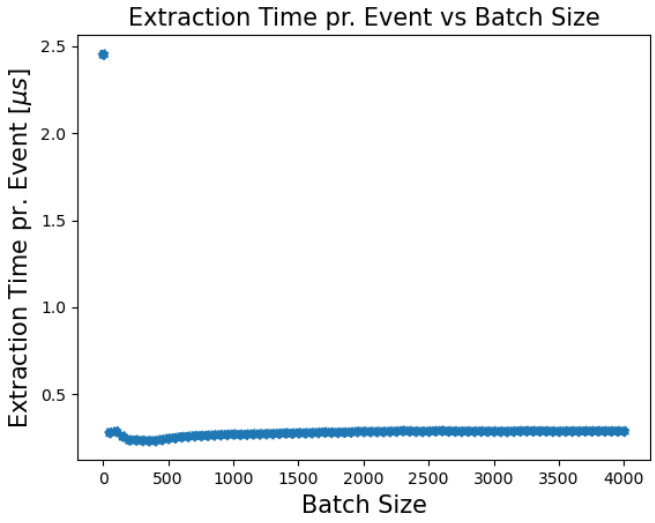


FIGURE 6.2: Extraction time pr. event in microseconds versus batch size (number of events extracted together) of a sqlite database containing level 5 DeepCore simulation data. Based on mean extraction times of 10 repetitions. These extraction times includes both extraction of feature and truth data.

on a nearest-neighbour, as calculated from a Euclidean distance, provides grounds for reasonable results and can effectively represent the irregular geometry of a particle detector. Results achieved in the master thesis of former NBI master student **Bjørn Mølvig**, where a BiDirectional GRU model is used, suggested that treating IceCube data as a pseudo time-sequence also leads to reasonable results. This was also shown in the master thesis of **Mads Ehrhorn** in his application of a **temporal convolutional neural network** (TCNN).

These observations lead to two competing edge configurations, which would eventually lead to two different GNN prototypes. To better implement the experience from previous work, and to better facilitate interpretation, **this work choose to represent each event in IceCube data as a single graph**. This representation then directly allows for the following translation between data and graph theory

- node $\longrightarrow$ Active DOM (E.g. A DOM that registered a pulse)
- node features $\longrightarrow$ Pulse data (DOM-specific measurements and related information depicted in 3.1)

In such a representation, every node in a graph represents a DOM, and the associated node features is then feature data $dom\_x, dom\_y, dom\_z, dom\_time, dom\_charge$. In the cases where a DOM is activated more than once during the same event, the DOM is represented in the graph more than once, where the time and charge is adjusted accordingly. **This means that only DOMS that activate during an event are included in the graph for that event.** The differences between the graphs in this work lies in the edge configuration, which are:

- **k-nearest neighbours**: This configuration provides an edge configuration that connect nodes based on their Euclidean distance from each other, calculated

based on the spatial information of the DOM that they represent. The number of neighbours is set as a variable.

- **time series**: This edge configuration connect nodes based on the time at which they activated, namely the variable *dom_time*. This provides a graph where the nodes are connected on a line, where the first node is the first DOM to activate during the event, and the last node is the last DOM to activate during the event. In effect, this mimics time-series data.

Another ansatz related to graph representation of the IceCube data is that **the fact that some DOMs are not activated during an event is in itself information**. In theory this could be incorporated by modeling the whole detector array in a single graph, such that all DOMs, and not just the active, gets represented by a node in a graph. Node features for inactive DOMs could be set to some constant value, but not zero, as Eq. 6.1 centers the input data around that value. The practicality of this were tested initially and were deemed too computationally expensive as memory requirements for edge specification grows exponentially with the number of nodes in a graph, effectively removing any chance of a GNN producing significantly faster reconstructions compared to current methods.

The above described choice of representation directly leads to another technical choice. If the graphs are to only include active DOMs, any graphs produced will have a variable amount of nodes. From a technical perspective this forces the question of whether or not to impose a zero-padding scheme, like CNNs would require, as a significant number of implemented graph operators requires static dimensions, not only in node features, but also in node count. If one zero-pads, one would also

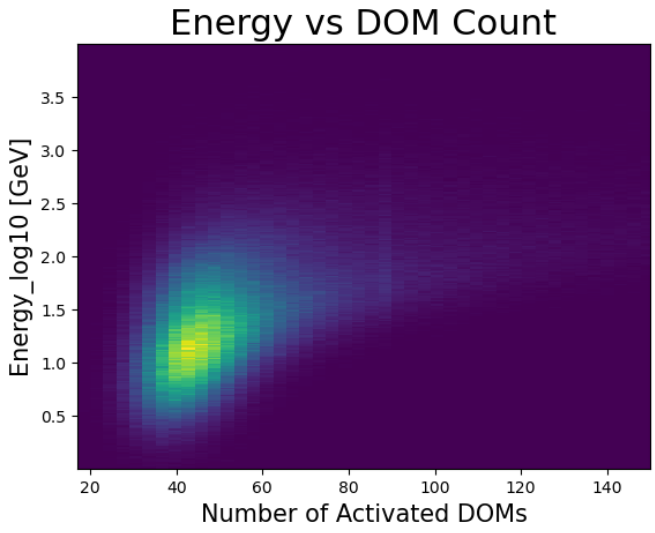
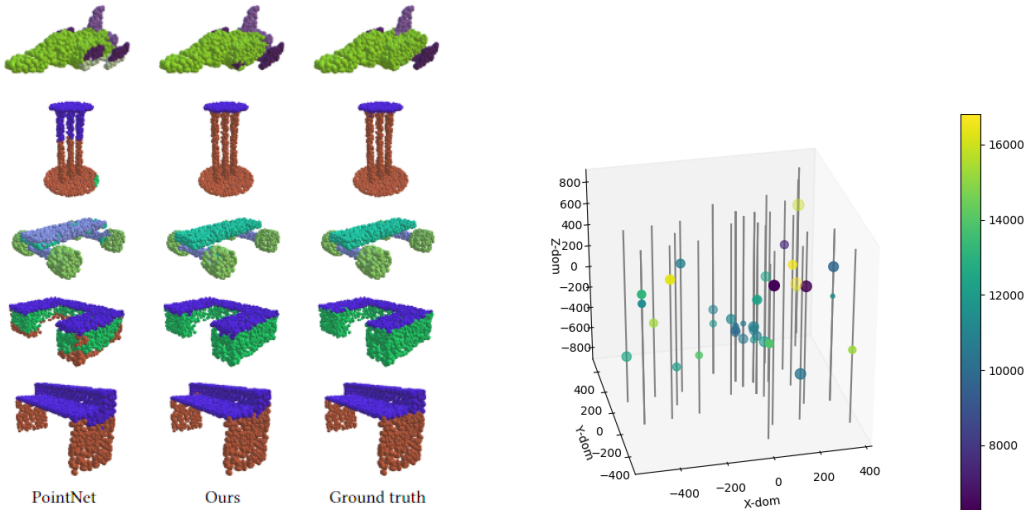


FIGURE 6.3: 2D histogram of event DOM count and energy from 500.000 muon-neutrinos at level 5 filtering. Raw pulses.

have to set an upper bound on DOM count and reject events exceeding the bound. From Figure 6.3 it's evident that a choice to cut at around 100 DOMs would only remove a tiny amount of events, but since the average amount of active DOMs in an event is around 40-45, such a cut would more than double the node count in each graph on average, without adding meaningful information to the problem.

EdgeConv

In [59], a point-cloud graph convolutional operator called **EdgeConv** is presented. It acts as the backbone of their proposed point cloud model capable of learning geometric shapes. The proposed model dynamically calculate graph configurations of the point clouds and apply the EdgeConv operator to extract local topological information. In 6.4 point clouds from the EdgeConv paper is depicted together with an



(A) (B)

FIGURE 6.4: **(A)**: Comparison of results on segmentation learning between the proposed model in [59] and an older point cloud model **PointNet**[60]. Kindly borrowed from [59]. **(B)**: A plot of active DOM positions in a muon-neutrino event. Colorbar depicts time variable dom_time.

illustration of an IceCube event. While the geometry of the EdgeConv data is more suggestive - One can arguably by intuition see the resemblance of point clouds in IceCube data. The EdgeConv operator acts on nodes as

$$\tilde{x_j} = Aggr(\{f(x_i, x_j) | x_i \in \epsilon(n_j)\}) \tag{6.2}$$

where $\epsilon(n_j)$ is the neighbourhood of the j'th node, and x_i, x_j denotes the node features of the i'th and j'th node, respectively. $Aggr()$ denotes a message aggregation scheme and $f(x_i, x_j)$ a function on node features.

In this work, the point cloud approach for the k-nearest-neighbours graphs were chosen as it seemed to be the most fit way to extract geometric data from the graphs. The EdgeConv operator were chosen specifically because it was known at the time to be the best in terms of performance and because it offers good flexibility on both aggregation schemes and the convolution function itself $f(x_i, x_j)$, leaving much to be tinkered with. In addition, the operator is insensitive to a variable amount of nodes, which removes immediate need for zero-padding. In the pytorch implementation of EdgeConv[5], the aggregation schemes $Aggr()$ can be chosen as either the sum, maximal or mean value of messages $\{f(x_i, x_j) | x_i \in \epsilon(n_j)\}$. $f(x_i, x_j)$ is left as a user-defined input. [59] discusses several choices of $f(x_i, x_j)$ and how this specific choice have significant consequences on the information learned on the point

[5] which is available here

clouds. For initial prototyping $f(x_i, x_j)$ were set to a simple MLP with LeakyReLU activation function

```
conv = torch.nn.Sequential(torch.nn.Linear(l1*2,l2),
                           torch.nn.LeakyReLU(),
                           torch.nn.Linear(l2,l3),
                           torch.nn.LeakyReLU())
```

where LeakyReLU is a piece-wise linear version of ReLU that allows for negative values, a quality of necessity as 6.1 centers input variables around 0.

Node Aggregation Methods

In the broadest of terms, a machine learning model receives an input with certain dimensions and returns an output that typically has different dimensions. If the input is an array, this would mean that somewhere along the way, the amount of columns and the amount of rows have to be adjusted *somehow*. This is also the case for GNNs, and in order to meet dimensionality in a supervised learning problem where each event is represented with its own graph, a GNN would have to adjust the number of node features (columns) and the number of nodes (rows) in order to meet the dimension of the regression target. In other words, the model needs to output a new graph where the number of nodes is equal to the number of rows in the truth array (which is 1, as we make a graph for each event), and the amount of node features is equal to the amount of truth variables.

As evident from Eq. 6.2, EdgeConv only acts on node features. Given the choice of MLP as $f(x_i, x_j)$, the EdgeConv layer has the ability to change the number of node features of each node. To change the number of nodes, one can choose between either using graph pooling layers or simple aggregation schemes like the ones available in EdgeConv. Since the pooling operators tend to only accept a 'keep-fraction' as input, and therefore doesn't let one specify the desired dimensionality of it's output, simpler aggregation schemes were chosen for the final many-to-one projection of nodes.

Evaluation Metrics and Loss Functions

In order to compare different architectures and choices, the prototypes needs a common metric for evaluation. Since the nature of the regression targets are different, different choices on loss functions are chosen. The reason for this is that a variable like *energy_log*10 is a real value that mostly[6] lies in the range $[0, 4]$, which corresponds to an energy spectrum 1 GeV to 10 TeV. For comparison, *azimuth* is a real, cyclical variable in the range $[0, 2\pi]$, where 2π and 0 represents the same point on a circle. An evaluation metric that is common between all choices of loss function is the **width of the error distribution** given as:

$$W(e) = \frac{IQR(e)}{1.349} \tag{6.3}$$

[6]The available upgrade simulation data does produce events with energy lower than 1 GeV, which produces a small number of negative values for *energy_log*10.

where e is the error, and the factor 1.349 is added such that $W(e)$ approximately correspond to one standard deviation under the assumption that the errors are Gaussian. This is an unbounded distribution that serves as a descriptor on prediction consistency with easy interpretation - the lower a width the better. This measure, however, does not capture any bias in prediction and can as such not serve as the only measure of evaluation. Below the additional measures listed.

Energy Regression

The quantity of interest for energy regression in this work is **percentage error**, a relative error measure on the form

$$e = \frac{E_{Reco} - E_{True}}{E_{True}} \tag{6.4}$$

which describes the error of the prediction E_{Reco} in percentage of the true energy E_{True}. Using this error measure as a loss function in training directly provides a series of immediate issues. Firstly, the scale of the energy is wide, meaning that the output of a GNN needs to cover several orders of magnitude, which is generally known to lead to a sink in prediction quality in ML. This has been addressed in preprocessing by taking the logarithm to the quantities, so if we were to plug in those values directly in Eq. 6.5, we'd get

$$e = \frac{log_{10}(E_{Reco}) - log_{10}(E_{True})}{log_{10}(E_{True})} \tag{6.5}$$

But this is problematic as this effectively corresponds to changing the base of the logarithm to be in the true energy [7]. To accommodate this, one can then apply the fact that $log_{10}(A) - log_{10}(B) = log_{10}(\frac{A}{B})$ to write a error measure on the form

$$e = log_{10}(E_{Reco}) - log_{10}(E_{True}) = log_{10}(\frac{E_{Reco}}{E_{True}}) \tag{6.6}$$

Eq. 6.1 is applied on the quantities, such that if the preprocessed data is plugged in directly in Eq. 6.6, we'd get

$$e = C \cdot log_{10}(E_{Reco}) + B - C \cdot log_{10}(E_{True}) + B = C \cdot log_{10}(\frac{E_{Reco}}{E_{True}}) \tag{6.7}$$

in simpler terms, this corresponds to:

$$e = output - \tilde{E}_{True} = C \cdot log_{10}(\frac{E_{Reco}}{E_{True}}) \tag{6.8}$$

where output is the output of the GNN and $\tilde{E}_{True}$ is the preprocessed truth. This isn't problem free either, as this loss function is unbounded from below, and a minimization of Eq. 6.8 would simply mean outputting large negative values, effectively producing gibberish. What makes Eq. 6.8 appealing is the property

$$|log_{10}(2E_{True}) - log_{10}(E_{True})| = |log_{10}(\frac{1}{2}E_{True}) - log_{10}(E_{True})| \tag{6.9}$$

[7] Early on this was actually tested out of curiosity, and it did not produce good results. One can also conclude this apriori by realizing that the measure is diverging on the lower end of the energy scale.

which means that the numerical size of the loss in cases where the GNN predicts twice the true energy is equal to the cases where it predicts half the true energy. What remains to be chosen is the way in which this numerical size is generated. In this work two such methods are tested: **1)** e = $Abs(output - \tilde{E}_{True})$ and **2)** $Log_{10}Cosh(output - \tilde{E}_{True})$.

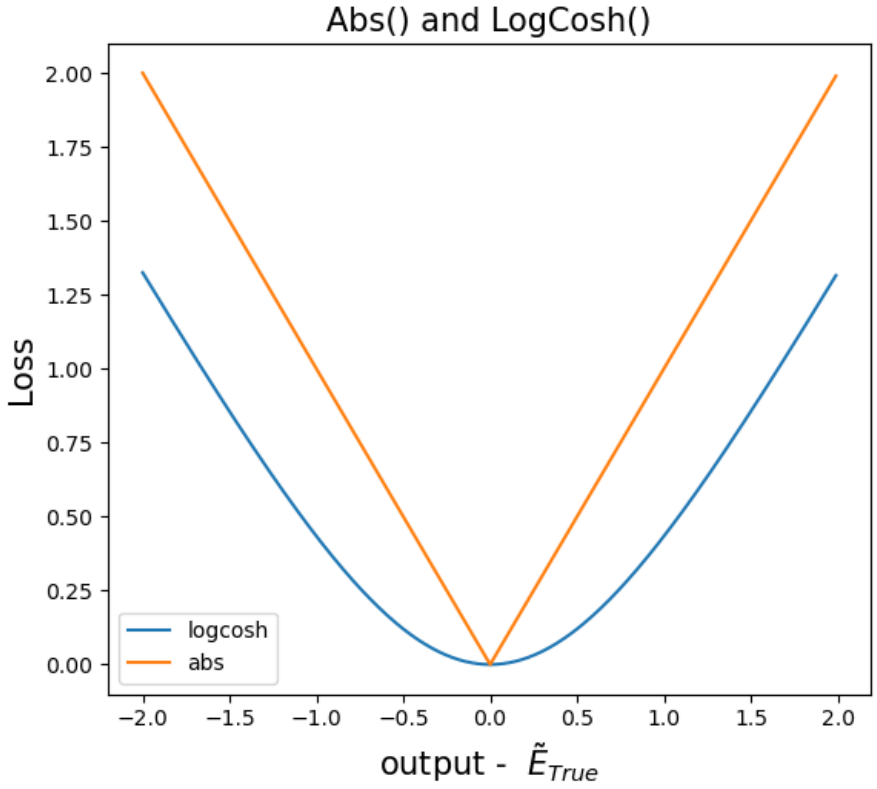


FIGURE 6.5: Plot of the two loss functions.

As evident from Figure 6.5, when the difference between output and truth is close to 0, the logcosh function offers a smoother surface for differentiation, which in theory should be more forgiving to the size of the learning rate.

Angular Regression

The *zenith* angle lies in the range $[0, \pi]$ where the bounds correspond to two different points on a line. As of such, the zenith angle can be regressed by minimizing the numerical difference between the predicted and true angle. e.g.

$$e = (output - zenith_{scaled})^2 \tag{6.10}$$

However, this isn't a viable option for *azimuth*, as a numerical difference of 2π would punish the GNN despite producing good results. This will produce *azimuth* predictions too high at the lower end of the angle range and too low at the higher end. If one thinks of the an arbitrary azimuth angle ø as a point on a circle in $\mathbb{R}^2$, one realizes that a learned regression of ø $f : \mathbb{R}^d \longrightarrow \mathbb{R}^2$, where d is the dimension of the input data, cannot be surjective. The surjectivity of the GNN can be controlled via the choice of activation function, as $tanh(x)$ is surjective, while *LeakyReLU* is

not. Alternatively, instead of predicting *azimuth* directly, one could predict sin (ø) or cos (ø) as these defines a line in **R**, like the *zenith* angle. Both methods are examined and evaluated in this work. If one chooses to regress *azimuth* directly, one could attempt a circular numerical difference such as **The Cosine Loss**

$$e = \sqrt{(1 - \cos\left(output - azimuth_{scaled}\right))^2} \tag{6.11}$$

A more indirect option is to minimize the angle between the predicted and true direction, eg:

$$e = 1 - \frac{\bar{R}_{pred} \cdot \bar{R}_{true}}{|\bar{R}_{pred}| \cdot |\bar{R}_{true}|} = 1 - \cos\left(\Delta ø\right) \tag{6.12}$$

where $\Delta ø$ is the angle between the true and predicted direction. This should also minimize the zenith and azimuth errors, and they could be recovered by a bit of algebraic effort, eg.

$$zenith_{pred} = \arccos(\frac{\bar{R}_{predz}}{r}) \tag{6.13}$$

$$azimuth_{pred} = \arccos(\frac{\bar{R}_{pred\,x}}{r \sin\left(zenith_{pred}\right)}) \tag{6.14}$$

An obvious reason for choosing Eq. 6.12 is that one regresses 5 variables at once, but it might not be the optimal choice for all variables involved. Another choice of *azimuth* regression is the **Sine-Cosine Pair Loss**, where the *azimuth* angle is represented as a pair

$$p = [\sin(azimuth), \cos(azimuth)] \tag{6.15}$$

and the GNN then outputs a prediction for each element in the pair, such that a loss on the form

$$e = \sqrt{(output_1 - \sin(azimuth))^2 + (output_2 - \cos(azimuth))^2} \tag{6.16}$$

can be calculated. The *azimuth* prediction can then elegantly be extracted by

$$azimuth_{pred} = \arctan(\frac{output_1}{output_2}) \tag{6.17}$$

A last angular method investigated in this work is a probabilistic regression based on a **von Mises-Fisher** distribution with probability density

$$p_n(\bar{x}|\bar{u}, k) = C_n(k) \exp\left(k\bar{u} \cdot \bar{x}\right) = C_n \exp\left(k \cos \Delta ø\right) \tag{6.18}$$

where $\bar{x}$ and $\bar{u}$ are n-dimensional unit vectors of predicted and true direction, respectively, and k resembles $\frac{1}{\sigma^2}$ from a normal distribution. C_n is the normalization constant written in terms of *modified bessel functions* $I_{\frac{n}{2-1}}(k)$, given by

$$C_n(k) = \frac{k^{\frac{n}{2-1}}}{(2\pi)^{\frac{n}{2}} I_{\frac{n}{2-1}}(k)} \tag{6.19}$$

which in the 3-dimensional case reduces to

$$C_3(k) = \frac{k}{2\pi \sinh(k)} \tag{6.20}$$

Eq. 6.18 defines a probability distribution on a (n-1)-sphere embedded in $\mathbb{R}^n$, and as written it's simply a probabilistic version of Eq. 6.12, but $\Delta\text{ø}$ could be any desired angle. A loss function can be created from Eq. 6.18 by using the **negative log likelihood** trick, which is

$$e = -\ln p_n(\bar{x}|\bar{u}, k) = -\ln(k) + \ln(4\pi) + k + \ln(1 - \exp -2k) - k\cos(\Delta\text{ø}) \quad (6.21)$$

Eq. 6.21 optimizes Eq. 6.18 by minimizing it's exponent. The benefit of a probabilistic regression is that if one lets the GNN output an estimate of k together with its prediction, it serves as the networks own error estimation. This is a valuable tool for later data selection and for producing valid error estimations. A clear problem with Eq. 6.21 is that the logarithmic terms are singular for k = 0 and the introduction of k in general can lead to the network paying more attention to k than minimizing the angular-related loss. [61] proposes two **regularization terms** to Eq. 6.21 to counter this behavior

$$L_1 = 0.2 \cdot k \quad (6.22)$$

$$L_2 = -0.2 \cdot \cos(\Delta\text{ø}) \quad (6.23)$$

and present results superior to an unregularized Eq. 6.21. In this work Eq. 6.21 with both regularization terms and with none, as [61] shows that this should produce the biggest comparative difference, will be tested. Also, a series of experiments are ran with the unit vectors in Eq. 6.18 changed to be on the form

$$\begin{vmatrix} \sin(\text{æ}) \\ \cos(\text{æ}) \\ 1 \end{vmatrix}$$

such that a probabilistic version of the Sine-Cosine Loss can be obtained. The last entry of 1 is added to meet the dimensionality, such that the simple normalization constant can be kept.

Most Notable Conclusions From Prototyping

The above considerations lead to a geometric prototype for energy regression that has the architecture illustrated in Figure 6.6. In this model the data is put through the EdgeConv layer, where the node features are convoluted and aggregated. The data is then interpreted by an MLP, and its output is aggregated to a graph with a single node. A final MLP interprets the node features of the single node graph and outputs a prediction directly.

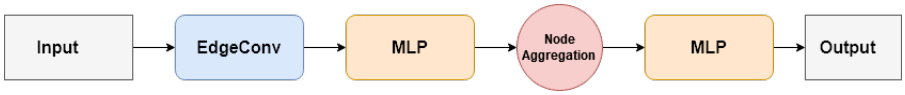


FIGURE 6.6: Graphic explainer of the architecture of the first geometric prototype.

Aggregation Schemes

In Figure 6.6, the EdgeConv layer needs to have a node aggregation scheme specified. Also, after the first MLP layer, a choice on a node aggregation scheme needs to be made to carry out the many-to-one projection of nodes. First notable round of experimentation were oriented at testing which aggregation schemes for the EdgeConv layer, and which aggregation schemes for the nodes, were best.

The data set used for testing aggregation schemes is a subset of a level 5 filtered data set originating from the GENIE simulation. Such a filtering level is chosen to minimize the chance of any poor performance being the result of noise. The data set consists of 2 million muon-neutrinos. The training sample is a randomly selected subset accounting for approx. 75% of the prototyping sample. The rest is reserved for validation. Distributions for key variables is available in Appendix A.

The EdgeConv layer supplies 5 ways of aggregating node features:

- **summation**
- **mean**
- **maximal**
- **Concatenation of the three**

The many-to-only projection of nodes can be done in 5 ways:

- **minimal**
- **maximal**
- **mean**
- **summation**
- **Concatenation of the four**

This leaves 20 possible configurations for testing. The concatenation is included as the MLP layers in Figure 6.6 should be able to learn which aggregation to rely on in certain cases if none of them are consistently better than the others. As the experiments ran, it became clear that the model utilizing the concatenation scheme for the EdgeConv aggregation were able to converge to a lower validation loss than its single aggregation counterparts. In Figure 6.7 the validation loss for the prototype using the concatenation of convolution aggregation is shown. The different runs represent choice of many-to-one node aggregations. As seen in Figure 6.7, it's difficult to conclude with absolute certainty from the validation loss that the combined node aggregation is the better performing model, as the validation loss is spiky between some epochs. The spiky-ness is an indicator of that the learning rate is not optimized perfectly, as such spikes can be produced in cases where the learning rate is too high. A second contribution to this behavior also lies in the fact that improvement made on the training set does not necessarily translate into improvement on the validation set.

In Figure 6.8 the energy error measure as defined in Eq. 6.8 is shown as a function of energy for the runs in Figure 6.7. As evident, the difference between models is

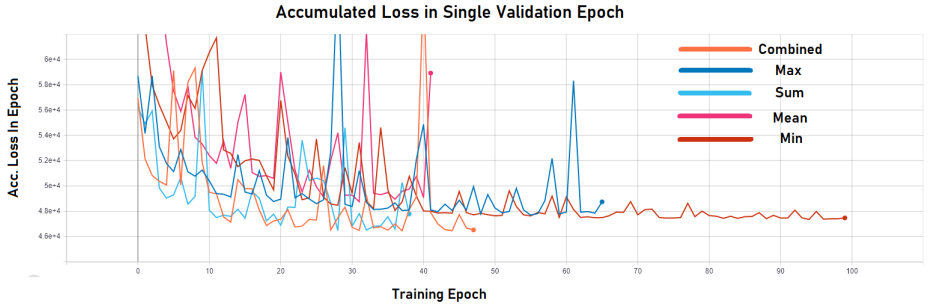


FIGURE 6.7: Comparison of validation loss for the EdgeConv Concatenation model. Different runs here represents different many-to-one node aggregation schemes. The Y axis is the accumulated validation loss within a given epoch.

small in the low to mid energy range, again making a definite conclusion difficult, as the difference in performance is so small that it is not impossible that the difference could lie in the weight initialization.

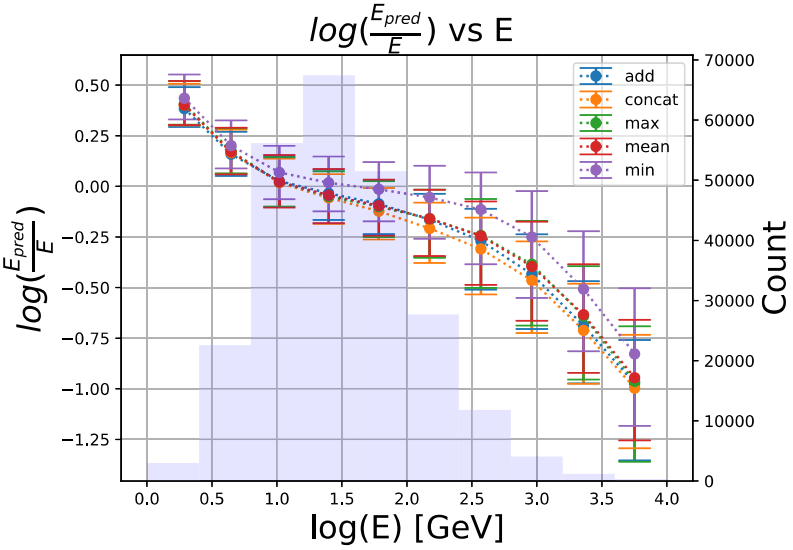


FIGURE 6.8: $log(\frac{E_{pred}}{E_{True}})$ as a function of neutrino energy. Center of error bars correspond to the median of the errors in the energy bin. Error bars span the IQR of errors. Points are plotted in the mean of the given error bin. Background histogram displays the energy distribution of the validation sample.

What can be concluded is that in the data rich area, roughly between 10 - 100 GeV, the median error sits comfortably close to 0 across all models, effectively showing that a reasonable energy reconstruction using GNNs *is* possible. In Figure 6.9 the width of the errors in Figure 6.8 is shown. Here a difference in consistency is clear. Across the energy spectrum, the concatenation model most often produce the errors with the smallest width. Based on this small analysis, the architecture was changed to Figure 6.10.

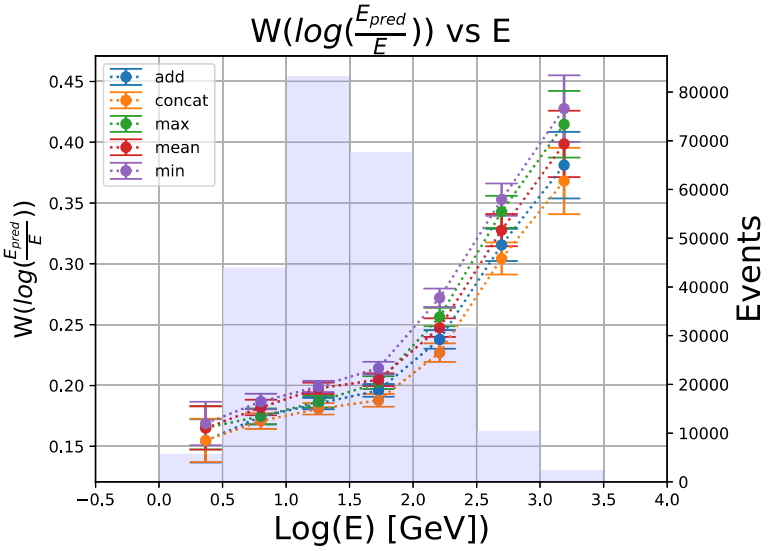


FIGURE 6.9: Width of $log(\frac{E_{pred}}{E_{True}})$ as a function of energy. Points are plotted at the location of the mean in the given error bin. Background histogram is the energy distribution of the validation sample.

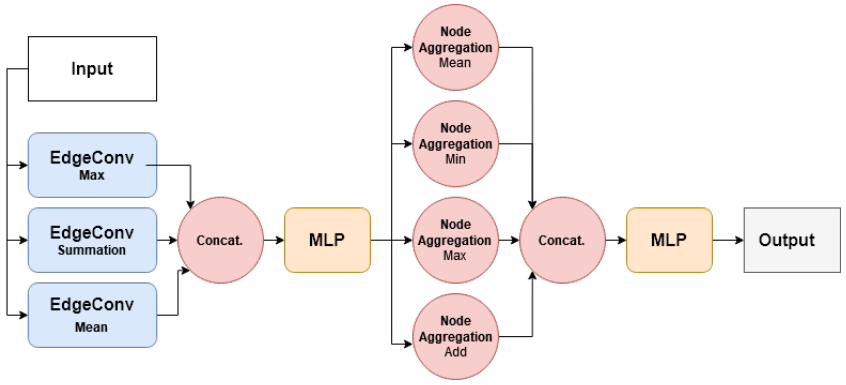


FIGURE 6.10: Graphic showing the architecture of the geometric model with full aggregation scheme.

Pooling Operators and Batch Normalization.

In Figure 6.6, one can see that the architecture is quite similar to that one would expect from a CNN. The key differences lies in the absence of any pooling and batch normalization layers, as these would typically follow after a convolutional operator. The pooling operator in CNNs are used to introduce translational invariance to the model, a property that decreases a CNNs tendency to assign importance to the location of the extracted features. In this work the following pooling operators were tested both in Figure 6.6 and Figure 6.10 architectures: **TopKPooling**[62], **ASAPooling**[63], **SAGPooling**[64] and **EdgePooling**[65]. From their respective benchmarking results, especially those presented in [65], one would expect an increase in performance by utilizing pooling operators. However, experiments carried out in this work showed that the graph pooling operators did not increase performance

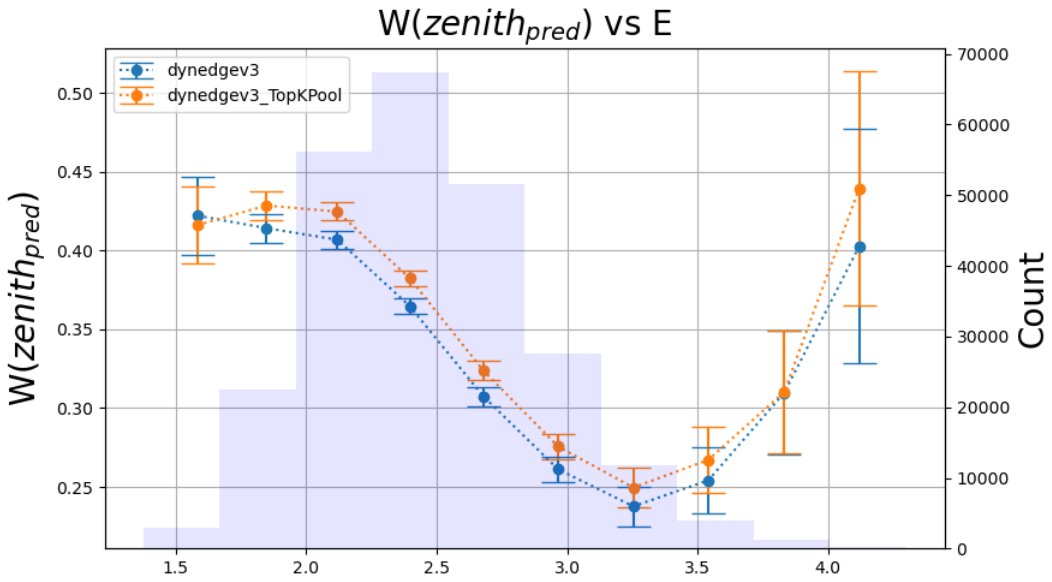


FIGURE 6.11: Width of Zenith angle prediction errors for the basic model, denoted with **dynedgev3** and the basic model + TopKPooling operator denoted **dynedgev3_TopKPool**. This particular run utilized the architecture shown in Figure 6.10 with no batch normalization.

and in some cases, as seen in Figure 6.11, the pooling operator could lead to worse prediction consistency. While these results were surprising initially, an explanation might lie in two factors. Firstly, the results presented in the above mentioned benchmarks are not point cloud problems, and as such, a direct comparison is difficult. For point clouds, Pooling nodes alters the topology of the graph by decreasing the amount of nodes and replacing edges. This could lead to sub-optimal graph representations, as evident for TopKPooling. Secondly, as mentioned in [65], pooling operators on graphs are under-developed compared to convolutional operators. For this reason pooling operators were not included in the final model.

Batch Normalization is a tool to combat the change in distribution of the output of single layers in a ML model, which can prolong the time it takes for a model to converge to a global minima in the loss landscape. This change in distribution is referred to as an *internal covariate shift* in literature[66]. Generally, a batch normalization is done as

$$\hat{x_i} = c \cdot \frac{x_i - mean(x_i)}{\sqrt{Var(x_i)} + \epsilon} + b \tag{6.24}$$

where c and b are learnable parameters, and ϵ a scalar for numeric stability. Batch normalization were tested in this work both as a direct correction to the output of the EdgeConv and node aggregation layer in Figure 6.10 separately. Tests were also carried out where the batch normalization were used to correct the output of the pooling operators. The results of the experiments were that convergence time were increased by a significant amount. The increased time for convergence is due to the increase in trainable parameters. It's plausible that Figure 6.10 simply isn't deep enough to benefit from batch normalization. While this might seem atypical, it comes with several benefits; decreased training time and restored independence

within mini-batches. For this reason batch normalization were not included in the final model. This issue was revisited much later in the process, and interestingly [67] presents a normalization free ResNet and argues that such models are superior.

Choosing Angular Loss Functions

In Figure 6.12 and Figure 6.13 the width of *zenith* and *azimuth* predictions from the angular loss function experiments are displayed. Only a subset of the discussed loss functions discussed earlier are displayed. The cosine loss (Eq. 6.11), the loss related to the angle between unit directional vectors (Eq. 6.12), and the von Mises distribution of the angle between unit directional vectors (Eq. 6.21) have been omitted from Figure 6.12 and Figure 6.13 as these all performed worse than those displayed[8]. To much excitement and surprise, the numerical difference as initially proposed as loss function for *zenith*, did not outperform the the fancier loss function alternatives proposed for *azimuth*. As evident from Figure 6.12, picking a clear winner isn't easy. While the regularized von Mises, denoted by **von Mises-az-zen**, outperforms its competitors on the higher end of the energy scale, the performance gain from regularization doesn't seem to hold in the rest of the energy scale. The Sine-Cosine Loss, denoted by **zenith-cosine-pair-tanh**, seems to be the worst performing in general.

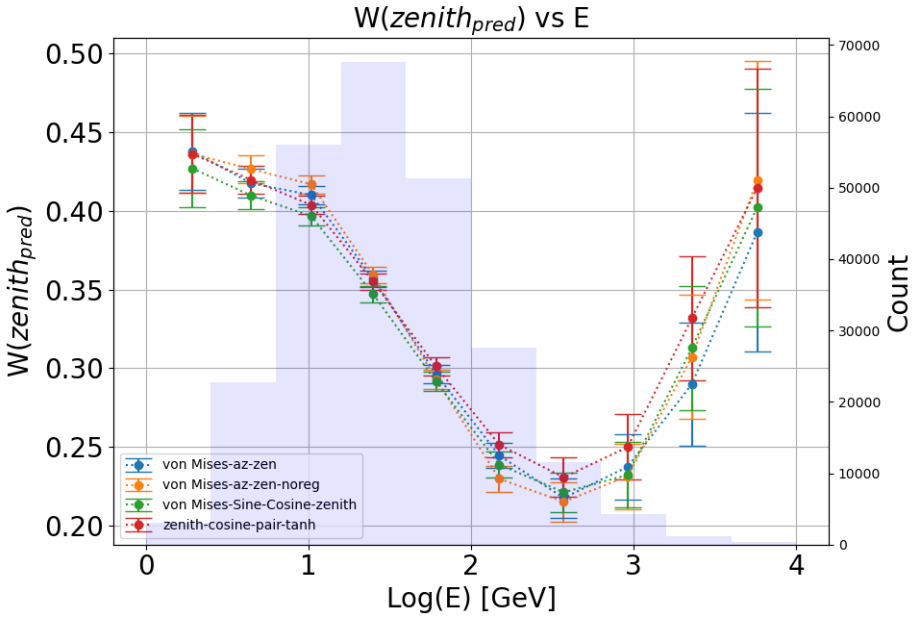


FIGURE 6.12: width of the *zenith* predictions as a function of energy. **von Mises-az-zen** denotes the probabilistic regression of both *zenith* and *azimuth* with regularization terms. **zenith-cosine-pair-tanh** denotes the *zenith* regression using the Sine-Cosine Loss with tanh as final activation function. **von Mises-az-zen-noreg** denotes the probabilistic regression of both *zenith* and *azimuth* without regularization terms. **von Mises-Sine-Cosine-Zenith** represents the probabilistic *zenith* regression using the Sine-Cosine-style unit vectors with regularization.

This leaves only probabilistic loss functions, which is good news as this gives access

[8] But not out-of-plot worse

to estimates on k without loss of performance. This work chooses Eq. 6.21 with the Sine-Cosine unit vectors as *zenith* loss function as it's width of prediction on *zenith* lies lower and outside the uncertainty on the widths of it's competitors on the lower energy bins and well within their uncertainty on the higher energy bins.

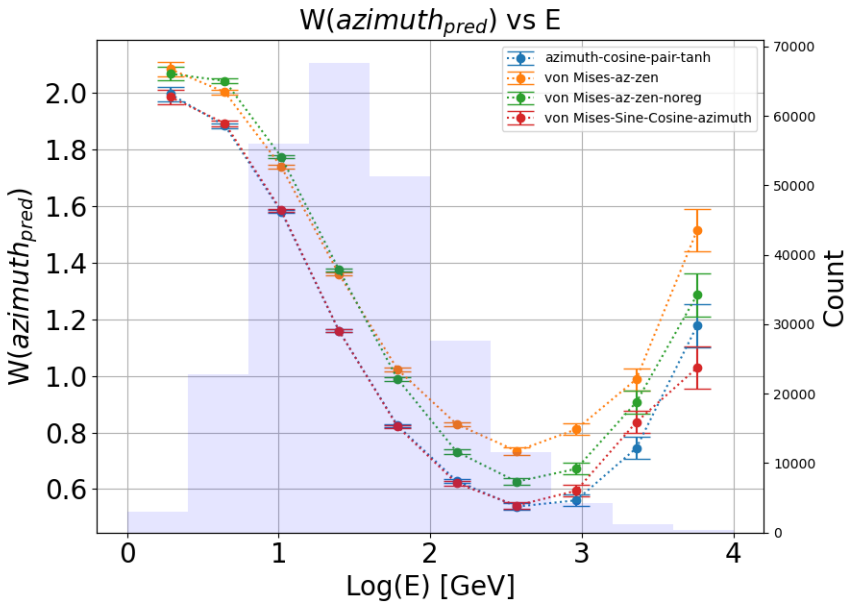


FIGURE 6.13: width of the *azimuth* predictions as a function of energy. **von Mises-az-zen** denotes the probabilistic regression of both *zenith* and *azimuth* with regularization terms. **azimuth-cosine-pair-tanh** denotes the *azimuth* regression using the Sine-Cosine Loss with tanh as final activation function. **von Mises-az-zen-noreg** denotes the probabilistic regression of both *zenith* and *azimuth* without regularization terms. **von Mises-Sine-Cosine-Azimuth** represents the probabilistic *azimuth* regression using the Sine-Cosine-style unit vectors with regularization.

In Figure 6.13 a clearer picture emerges for *azimuth* regression. Neither the regularized or unregularized von Mises loss functions, denoted by **von Mises-az-zen** and **von Mises-az-zen-noreg** respectively, outperforms the alternative loss functions on any energy bin. However, for the probabilistic and regular Sine-Cosine-pair loss functions, denoted by **von Mises-Sine-Cosine-azimuth** and **azimuth-cosine-pair-tanh** respectively, results are almost identical, except in a few energy bins on the higher end of the energy spectrum. While it's clear that widths produced by the regular Sine-Cosine-pair loss function is superior around 2.9 - 3.4, this work chooses the probabilistic von Mises Sine-Cosine-pair loss as this gives access to estimates on k without unbearable sink in performance.

Testing The Geometric Edge Configuration

As briefly mentioned in the introduction to this chapter, the initial development of a working GNN requires working from an ansatz on how a meaningful edge configuration of a graph representation of IceCube data looks like. A priori, one would expect a completely random edge configuration to add no meaningful information to the graph representation of IceCube data, and that graph operators relying on

edges, such as EdgeConv, should perform better on graphs with information rich edge configurations as compared to random edge configurations. With a working

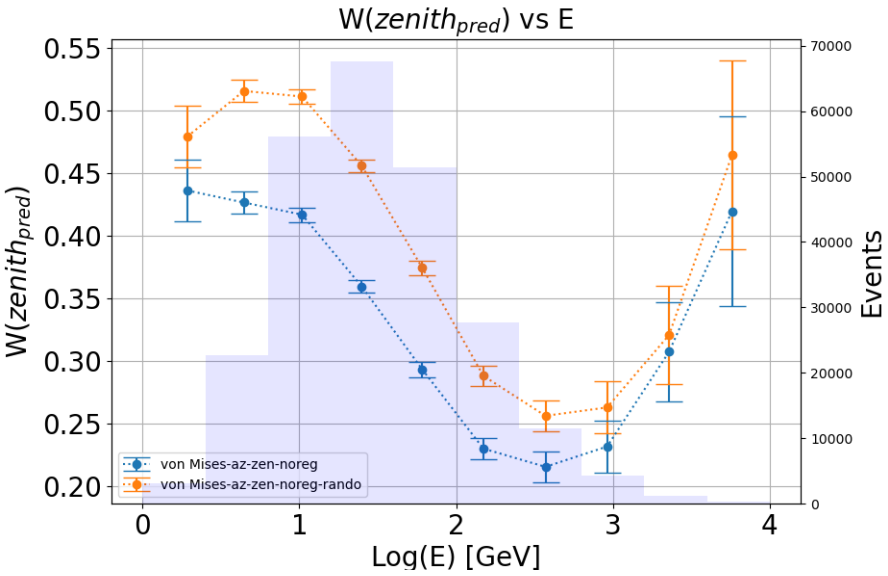


FIGURE 6.14: width of *zenith* predictions as a function of energy.**von Mises-az-zen-noreg** denotes the probabilistic regression of both *zenith* and *azimuth* without regularization terms trained and validated on the k-nearest-neighbours edge configured graphs. **von Mises-az-zen-noreg-rando** denotes the probabilistic regression of both *zenith* and *azimuth* without regularizfation terms trained and validated on the randomly configured graphs.

GNN prototype, the k-nearest-neighbours ansatz can be tested against a random configuration. Results from such an experiment is shown in Figure 6.14 in terms of *azimuth* prediction error width. The test were carried out by selecting a model, in this case the angular regression model utilizing the von Mises probabilistic loss function (Eq. 6.21) to regress both *zenith* and *azimuth*, were chosen. One copy of the model were trained and validated on graphs with k-nearest-neighbours edge configuration, while another copy of the model were trained and validated on graphs with a random edge configuration scheme. As evident from Figure 6.14, the geometric graphs does indeed lead to clear improvement in predictions as compared to a random configuration.

Learning Rate Schedules

This work experimented with a number of different available learning rate schedules. The motivation for learning rate schedules is that adjusting the global learning rate of an optimizer can speed up convergence of a model without adding addition learnable parameters to the model. One schedule tested is the **CyclicLR** schedule proposed in [68] and implemented in PyTorch, where the global learning rate oscillates between an upper and lower bound. Another scheduler, also implemented in PyTorch, is the **ReduceLROnPlateau**[9] which monitors the validation loss during training and consequently decreases the global learning rate if the validation loss

[9]No paper on this, but source code is available here

hasn't decreased in a specific amount of epochs. An additional two custom learning rate schedules have been implemented as part of this work: The **Inverse** (IS) schedule, and the **Piece-wise Linear** (PLS) schedule, both heavily inspired by the learning rate schedules used in the master thesis of **Bjørn Mølvig**. They're defined as:

$$PLS(step) = \begin{cases} step\frac{LR_{max}-LR_{init.}}{steps_{up}} + LR_{init.} & step \leq steps_{down} \\ step\frac{LR_{min}-LR_{max}}{steps_{down}} + LR_{max} - \frac{LR_{min}-LR_{max}}{steps_{down}} \cdot steps_{up} & step \geq steps_{down} \end{cases}$$

$$IS(step) = \begin{cases} step \cdot \gamma & step \geq step_{steps_{up}} \\ \frac{LR_{max}}{LR_{init.}} \frac{s}{s+step-steps_{up}} & else \end{cases}$$

where $s = steps_{down} \cdot \frac{LR_{min}}{LR_{max}-LR_{min}}$ and $\gamma = \frac{LR_{max}}{LR_{init}}^{\frac{1}{steps_{up}}}$. In Figure 6.15, the iterations be-

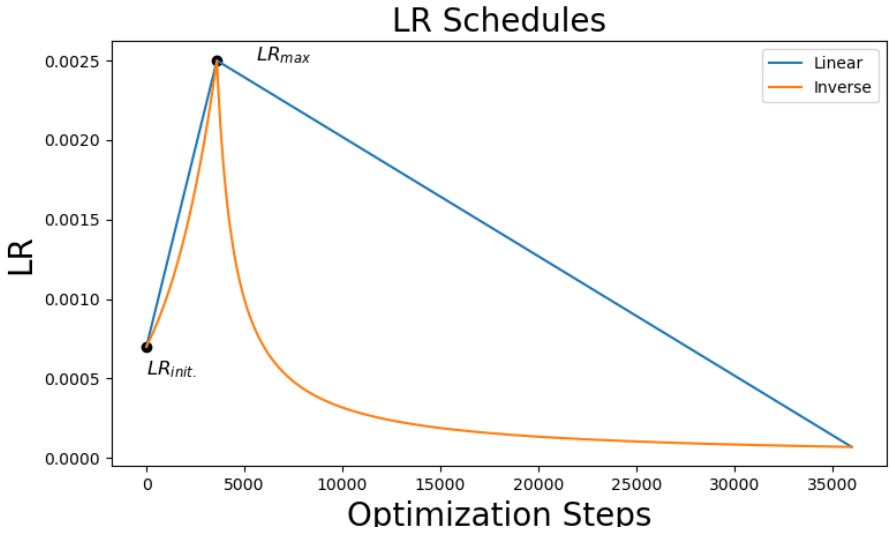


FIGURE 6.15: An example of the Linear and Inverse learning rate schedules.

tween $LR_{init.}$ and LR_{max} is the *warm-up* period, a technique proposed in [69] where the global learning rate starts low and increases to a maximal value. This trick is reported to increase generality and convergence speed for optimization algorithms such as ADAM. The bounds for the schedules are determined by completing a **learning rate scan**, a method where the change in validation loss as a function of global learning rate is recorded. LR_{max} represents the learning rate where the change in validation loss is highest. $LR_{init.}$ is set to a value comfortably under LR_{max} such that optimization steps taken under the warm-up period starts out gentle. Results from the learning rate scan is available in Table 6.1. The piece-wise linear schedule

$LR_{init.}$	LR_{max}	LR_{min}
7e-4	2.5e-3	1e-4

TABLE 6.1: Results from the learning rate scan of Figure 6.10 using ADAM optimizer with a batch size of 1024.

were included in the test to check if a steadier change in the learning rate could have an impact on performance compared to the inverse schedule, as the linear schedule 'hangs around' the learning rate with the greatest impact for longer.

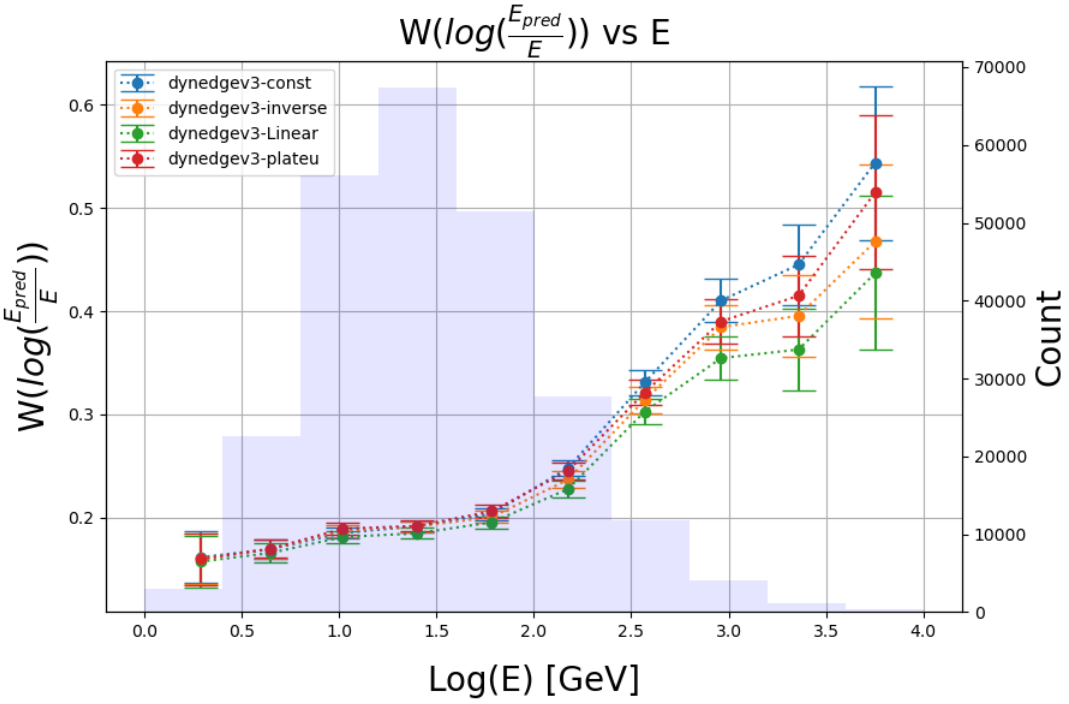


FIGURE 6.16: Width of Energy predictions of Figure 6.10 from runs with different learning rate schedules. **dynedgev3-const.** represents a model without schedule with global learning rate set to LR_{max}

When the validation loss as a function of epoch were inspected from the runs depicted in Figure 6.16, it became evident that the plateau schedule did not produce a model that converged to a global minima. However, the model without schedule (const), PLT and IS all converged to roughly the same validation loss in roughly the same amount of epochs. As seen in Figure 6.16, the widths of the error predictions between the const., PLS and IS models are very close - but generally PLS seems to produce the most consistent results. Therefore, this schedule is adopted in this work.

Weighted Ensemble

As evident in Figure 6.11 and the other plots, performance changes with the energy spectrum. A contributing factor to this is the distribution of the data. In general, it is optimal for the GNN to perform well in data rich areas. Therefore, it is less important if errors are higher in the data sparse areas of a given distribution, as such errors would be less frequent. This work attempts to negate this effect by introducing a **weighted ensemble** of models. Weights are designed on training and validation data to increase the models focus on a particular range of a variable distribution. This yields three models - one specifically for the lower range of a given variable distribution, and one for the higher range. A model with no weight is then added to represent the range between the two. The weights are calculated as:

$$w(x, x_{max})_{low} = \begin{cases} c & x \leq x_{max} \\ \frac{c}{1-x_{max}+x} & x \geq x_{max} \end{cases}$$

$$w(x, x_{min})_{high} = \begin{cases} k & x \geq x_{min} \\ \frac{k}{1+x_{min}-x} & x \leq x_{min} \end{cases}$$

where c and k are constants that are chosen such that the weights have a mean of 1.

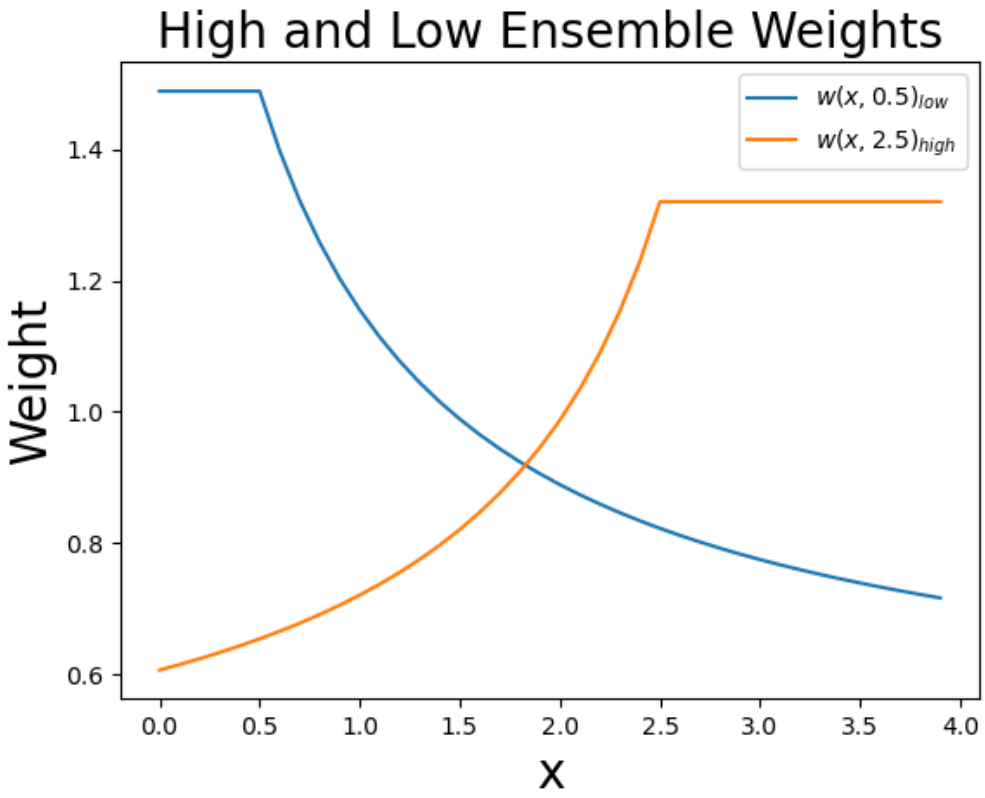


FIGURE 6.17: $w_{low}(x, x_{max} = 0.5)$ and $w_{high}(x, x_{min} = 2.5)$ illustrated for an arbitrary variable x.

These weights are then used directly in the loss function. In the case of the log-cosh for energy regression, this would equal

$$e(output, \tilde{x}, w) = w \cdot log_{10}(cosh(output - \tilde{x})) \tag{6.25}$$

This concept was tested on energy regression and the results showed that both errors and error widths lied within each others uncertainties between low, high and no weight models. This was quite surprising given this approach had improved energy regression results in the master thesis of **Bjørn Mølvig**, but since the model architectures in his and this work is quite different, such conclusions are not guaranteed to carry over here. From a practical perspective this is good news, as having to evaluate three separately weighted models and feed the input into a meta-learner to produce the final energy regression adds complexity and increases reconstruction time.

Notes on the Time-series Model

Similar steps as those described above were taken for the time series model, formally a **Gated Recurrent Unit Graph Neural Network** (GRUGNN) which eventu-

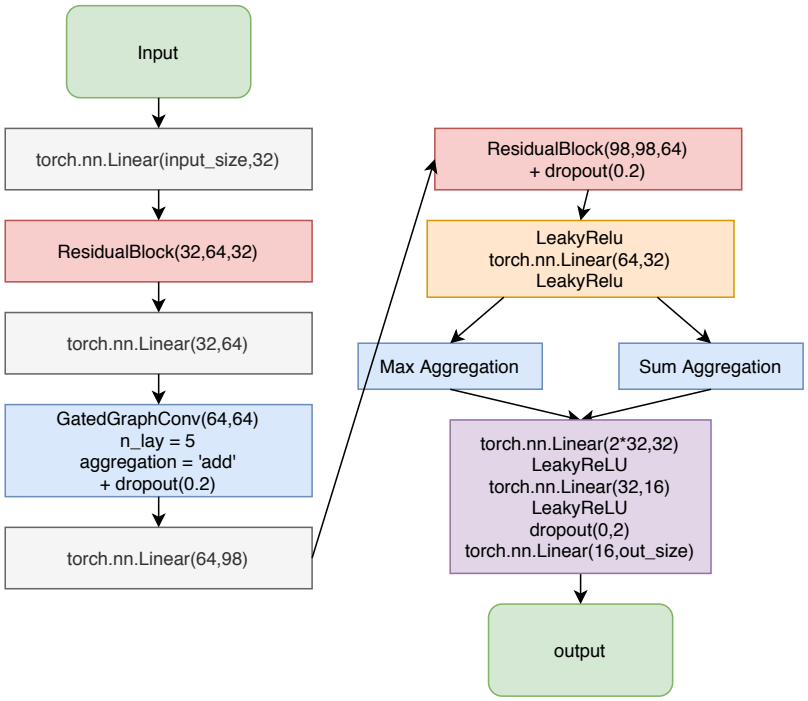


FIGURE 6.18: Illustration of the time series model. Direction of arrow depicts the flow of data.

ally took the architecture in Figure 6.18. The **ResidualBlock** is my custom graph implementation of [70], a proposed residual learning method where it's argued that deep neural networks trains better when emphasis is put on learning residual functions instead of target values directly. The **GatedGraphConv** is the PyTorch Geometric implementation of [71], a GatedRecurrentUnit operator on graphs that resembles the traditionally known operator. While this model offered comparable performance on *energy_log*10 regression, it's angular performance were inferior, as compared to the geometric model. It's not impossible that this model, with further tweaks and changes to architecture, might outperform it's geometric counter-part on energy regression, perhaps even angular regression, but due to limitations on computational resources and the impracticality of developing two models at once, it was decided about one third through the project to pick a single model - of which the geometric model were the best.

Additional Implementations

The Graph Factory

Because the input of GNNs are graphs, an additional step must be added in the data flow after the i3-to-SQLite pipeline, as the data stored in the SQLite databases are not in the shape of graphs. Early on it became apparent that if a GNN is ever to rival other ML methods of reconstruction, this additional step in the data flow must be swift. PyTorch Geometric offers a data set class, either for in-memory or direct disk reading, that originates from a different library called **torchvision** - a library for image processing. It was decided that this work would not implement the torch geometric data set classes as a method of creating and reading graphs, as this

part of the library [10] is poorly supported with documentation and examples to work from in a graph domain. Instead **Graph Factory** were implemented, a completely purpose-built SQLite-to-Graph pipeline utilizing high performance python compiling libraries such as Numba[11] and multiprocessing to build a fast and scalable pipe line to facilitate GNN prototyping and operation.

Parallel Data Loader

A custom data loader has been implemented in this work to minimize GPU utilization throttling. This is done by making sure that the computational bottleneck is the capabilities of the GPU itself and not disk read speed or CPU. This is also known as *GPU saturation*. An illustration of the data loader is shown in Figure 6.19. The

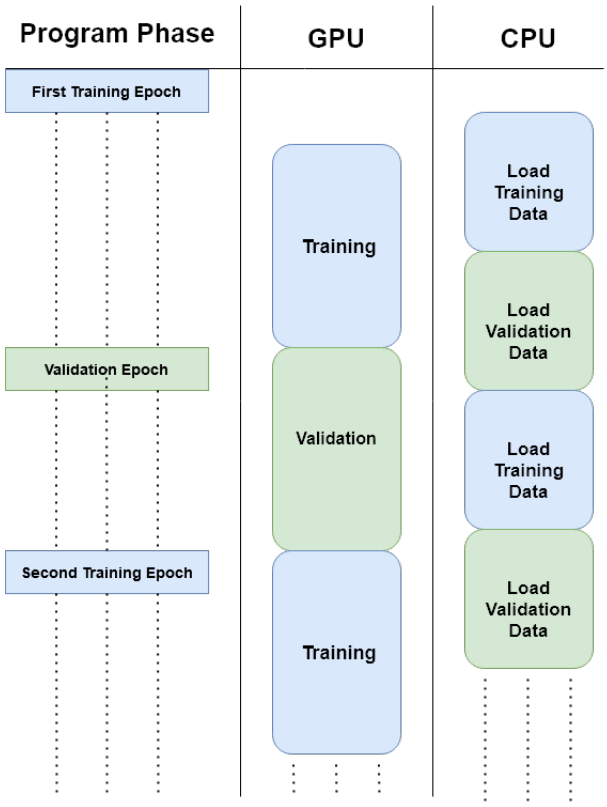


FIGURE 6.19: Graphic Illustration of the custom parallel data loader written to support training

'Program Phase' indicates the phase of the main code and the device columns 'GPU' and 'CPU' shows their tasks in the phases. The task of the CPU is to load data into memory before it is needed by the GPU, allowing the GPU to seamlessly transition between training and validation phases without throttling. The CPU fills the graphs into a Queue from multiple processes which is then passed to the GPU for consumption. This is known as a Producer-Consumer pattern and is quite memory efficient as graphs consumed by the GPU is automatically removed from memory.

[10] The data set examples are available here
[11] Additional information on Numba is available here

Chapter 7

The Final Model: dynedge

With a confirmed working model and a sufficiently explored landscape of LR-schedules and aggregation methods, it was now possible to further investigate if performance could be gained by making drastic changes to the chosen architecture as depicted in Figure 6.10. By fast forwarding to the conclusion of said investigation we arrive at Figure 7.1. This is final model chosen in this work, and the results shown in the following chapters originates from models with this architecture. As seen in the left

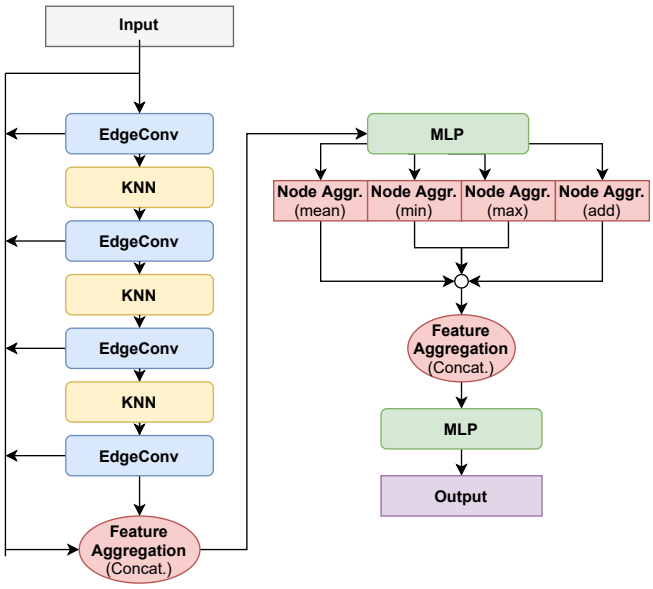


FIGURE 7.1: Final architecture of the geometric model.

side of Figure 7.1, the final architecture of the geometric model allows for the input of one convolutional operator to flow into the next while saving the state of each convolution. While this might seem dull, there's a point with this that can be easily interpreted: By feeding the output of a convolutional operator into the next **the next convolutional operator is applied on the neighbourhood defined by the previous operator**. Since one can interpret the convolutional operator itself as a translation of node features, this effectively corresponds to giving the model the ability to freely move nodes within the graph 4 times before these results are interpreted by the MLP-sequence on the right hand side of Figure 7.1. To get a gauge on how the model works on a technical level, let's go through a single forward pass on the fictive neutrino event shown in Figure 7.2.

In Figure 7.2 we have a neutrino event with 6 individual pulses recorded. For simplicity, let's pretend that for the i'th pulse we have measurements $x_i = [dom\_x, dom\_y, dom\_time]$ which make up our node features. We then construct the edges between the nodes

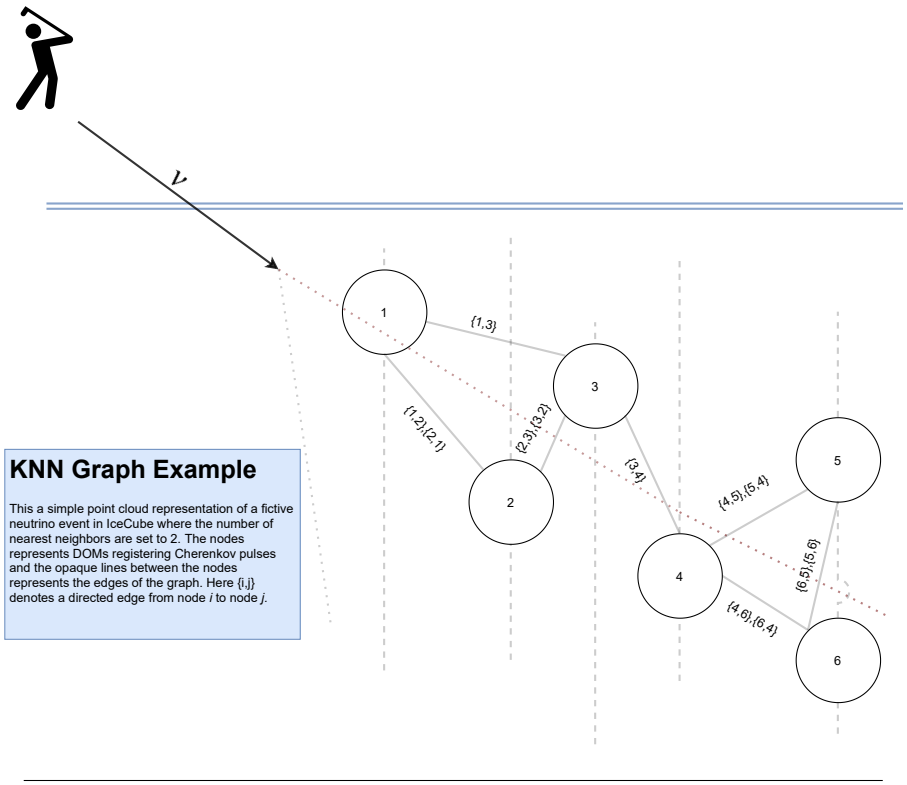


FIGURE 7.2: Example of a geometric graph where knn = 2.

according to *dom_x* and *dom_y* such that the i'th node is connected to it's 2 nearest neighbours. On the technical side, this means that the input for *dynedge* becomes an (n_nodes **x** n_features) = ($6x3$) array along with the edges shown in Figure 7.2. When the input flows into the first EdgeConv block, the convolution operation changes the node features of all nodes in the graph by considering the specified neighbourhood. Let's consider an update of node 1 (n_1) in Figure 7.2:

$$\tilde{x}_1 = f_{block1}\left([x_1, x_1 - x_2] + [x_1, x_1 - x_3]\right)$$

where $\tilde{x}_1$ represents the updated node features of n_1 and where $f()$ is a learned function specified by us. Through testing, this was chosen to be

```
f_block1 = torch.nn.Sequential(torch.nn.Linear(l1*2,l2),
                               torch.nn.LeakyReLU(),
                               torch.nn.Linear(l2,l3),
                               torch.nn.LeakyReLU())
```

which takes the form of a typical encoder-decoder but where l1 != l3, so dimensions change after the input flows through the first EdgeConv Block. Notice that

$$[x_1, x_1 - x_2] = [dom\_x, dom\_y, dom\_time, \Delta dom\_x, \Delta dom\_y, \Delta dom\_time]$$

is a (1,2*n_features)-dimensional matrix, and that $[x_1, x_1 - x_2] + [x_1, x_1 - x_3]$ therefore follows vector addition rules. In this manner, every node features are updated by the first EdgeConv block, and since the dimensions change, the number of features associated with each node has changed accordingly. Also, since the values within each feature now have changed, one can think of this as if the node has moved, and the initial edges designating nearest neighbours are no longer valid.

Now the output of the first EdgeConv block is saved, and a copy flows into the KNN-block, which simply calculates the new edges. But now that the node features are no longer physical, which do we choose to calculate the nearest neighbours with? It turns out this doesn't matter much, and the columns used (e.g. the columns that formerly contained positional information) are reused in this model, as the model adjusts for this choice. Based on this new edge configuration the second EdgeConv is applied and the above is repeated, such that the output of each EdgeConv block is saved, and a copy of it defines the edges for the next EdgeConv block which takes the copy as input.

When a total of 4 EdgeConv blocks have been applied to the data, each state along with the inital input is concatenated into a single array and passed through the MLP-interpreter sequence in the right hand side of Figure 7.1. The Pytorch model class is available in Appendix G.

Hyperparameter optimzation

The hyperparameter optimization was carried out by completing a full scan of the parameters below on an approx. 800k event sub-sample of the prototyping dataset with test-train split of 80%-20%.

Parameter	**Variation**	**Result**
k	[1,2,3,4,5,6,7,8,9,10,11,12,13,14]	8
f	[Single,Double,Triple]	Double
C	[1,2,3,4,5,6,7]	3
Optimizer	[ADAM, ADAM + LR (ADAM Rec)]	ADAM Rec

Here **k** denotes the k-nearest-neighbours used when computing edges, **f** denotes the user-specified mapping used by the convolutional operator, where *Single*,*Double*,*Triple* represents either a single, double or triple layered MLP. **C** represents the factor multiplied to the existing dimensional structure of the model. For details on the dimensions, see definition of l1,..,l7 at 4th line of Appendix G.

Chapter 8

Pre-Upgrade MC Results

In this section I display the MC reconstruction and classification results from the geometric model, *dynedge*.

Evaluation Data Details

The performance plots in this chapter is produced by models that has trained and predicted on data originating from i3-files stored at **Cobalt** in the following directories

```
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/120000"
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/140000"
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/160000"
"/data/ana/LE/oscNext/pass2/muongun/level7/130000"
```

As evident from the paths, these are events passing to level7 in oscNext event filtering. Level 7 events are chosen for evaluation in this work as these exclusively contain reconstructions from RetroReco, which will be included for reference. The directories contain a combined ≈8.1 million events. The pulses used to create the graphs for the model originates from the **SplitInIcePulsesSRT** i3-key. In an attempt to keep things brief, only performance plots from a models that has trained on all neutrino types is presented in this section. A prediction width comparison between PID selection is added in the following sections. The distributions of key variables are located in appendix.

Note: The version of RetroReco which was run on these events was a version that trades uncertainty quality for speed, which means that the uncertainty comparison shown in this section is quite unfair. If the version with better uncertainties had been accessible I would've included that instead.

Runtime Information

[1]

Model	Training Time (h)	Sample Size (mio)	Learnable Parameters
dynedge energy_log10	8	8.1	775539
dynedge angle	8	8.1	775539
dynedge classification	0.5	0.4	775539

[1] GPU used were NVIDIA RTX3090

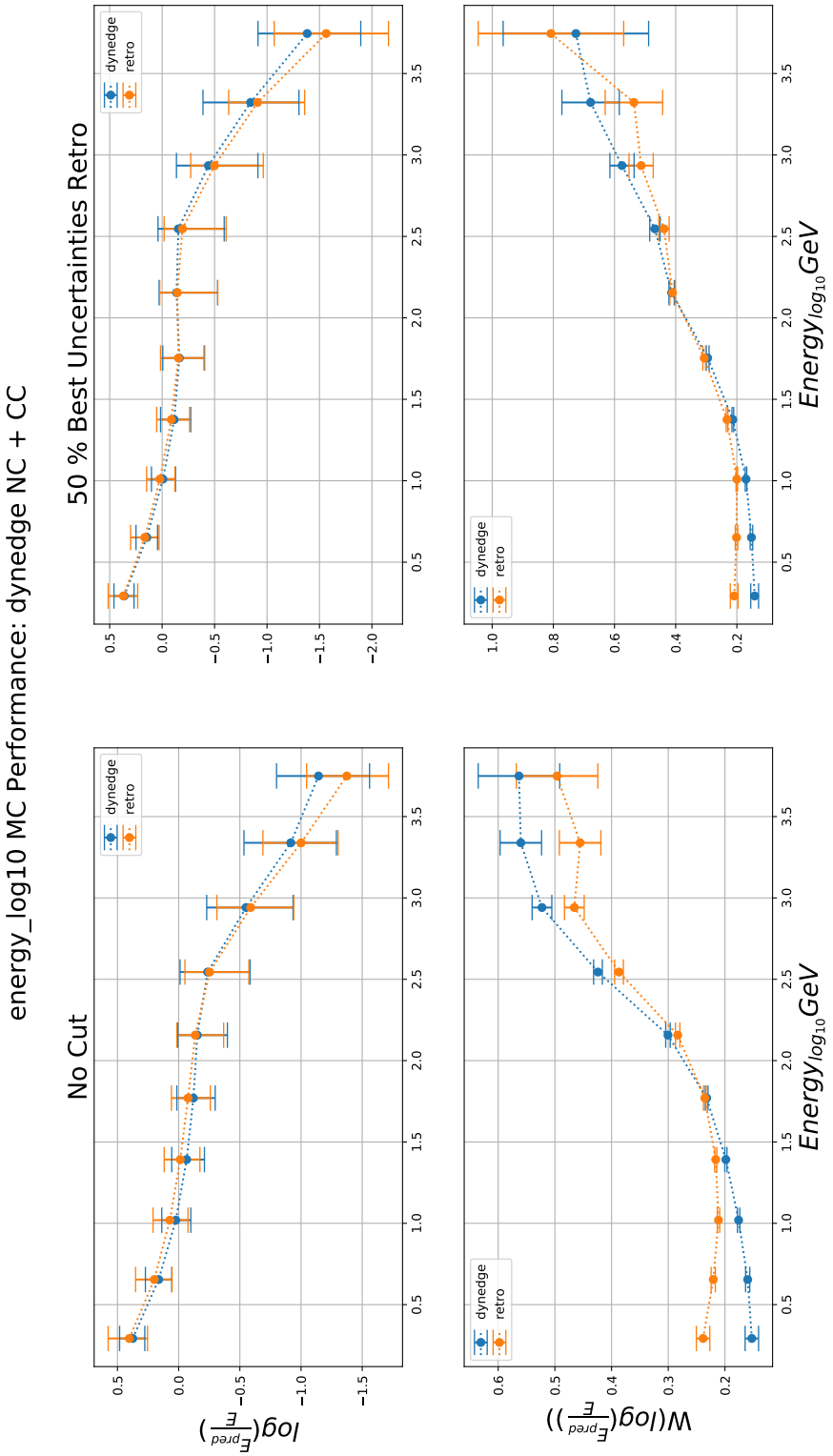
Regression Results

There will be two types of figures shown for each of the three key regression targets: **energy_log10**, **zenith**, **azimuth** respectively.

1) The Performance Plot
The first figure is a performance figure containing 4 subplots. The **upper rows contain the bias plot** - e.g. the angular difference for angles and the percentage-error for energy. The **second rows contain the error width plot**. The columns (left to right) represents a cut made in uncertainty, such that the first column shows data without cut and the second column shows the 50% most certain regressions. The uncertainties from which the cuts are made comes from either **RetroReco** or the model developed in this work, **dynedge**. The column title will indicate from which model the uncertainty cut originates, as there will be a separate figure in each case. This is included because any operational usage of regressions is likely to include cuts in uncertainties.

2) Prediction Distributions The second figure contains 6 sub-plots of distributions and 2D histograms of predictions. The first rows show distribution of predictions from RetroReco and dynedge. The second rows show a 2D histogram of RetroReco's predictions vs the true values. The third rows show dynedge's predictions vs the true values.

Figure title indicates interaction type. Also, the performance plots are available in a version where performance is measured as a function of number of pulses in each event instead of energy - these are located in Appendix A.18, A.19, A.17.


energy_log10 MC Performance: dynedge NC + CC
No Cut
50 % Best Uncertainties Retro
dynedge
retro
$log(\frac{E_{pred}}{E})$
$W(log(\frac{E_{pred}}{E}))$
$Energy_{log_{10}}GeV$

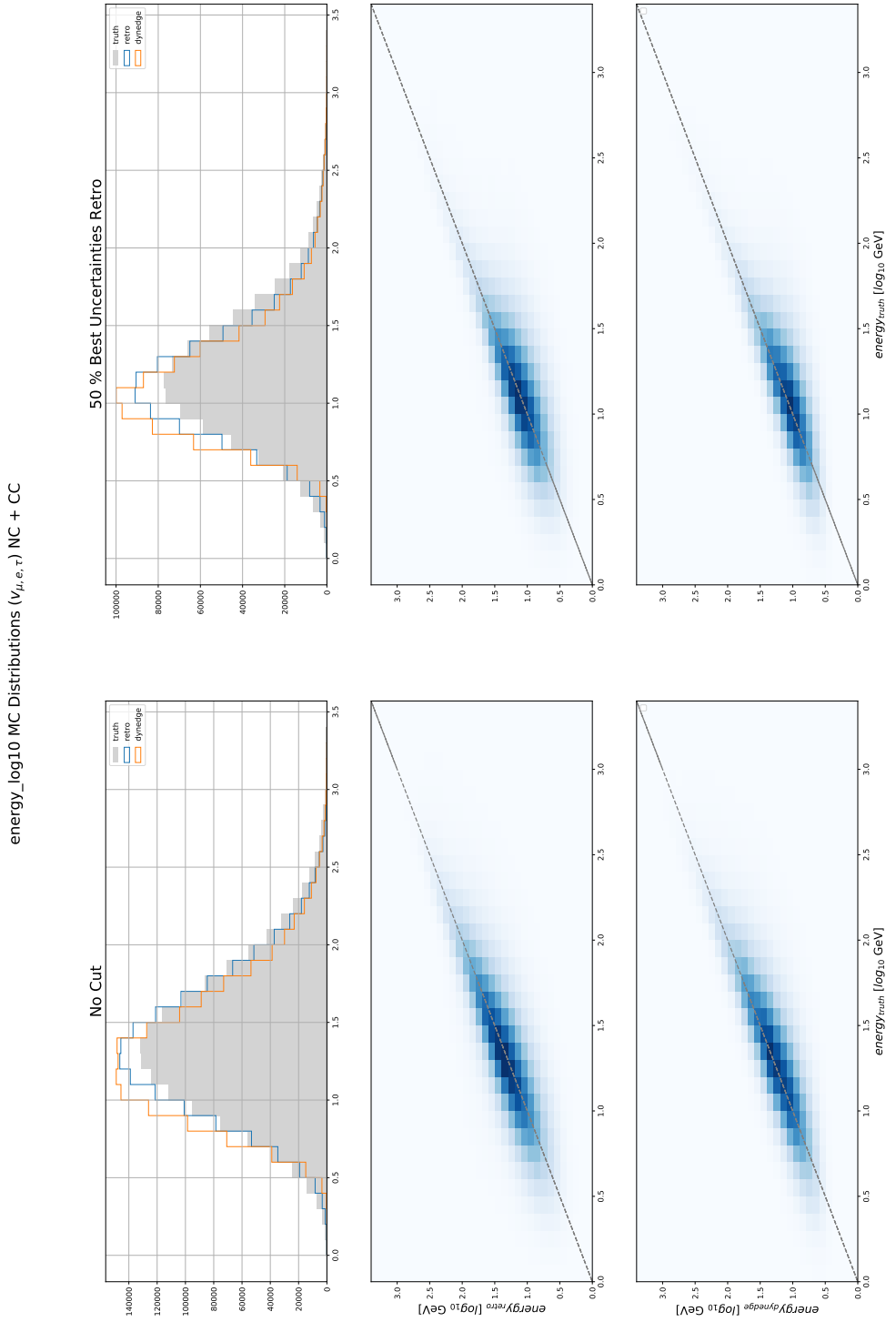

energy_log10 MC Distributions ($\nu_{\mu,e,\tau}$) NC + CC
No Cut
50 % Best Uncertainties Retro
truth
retro
dyneage
energy$_{truth}$ [log$_{10}$ GeV]
energy$_{retro}$ [log$_{10}$ GeV]
energy$_{dynedge}$ [log$_{10}$ GeV]

FIGURE 8.1

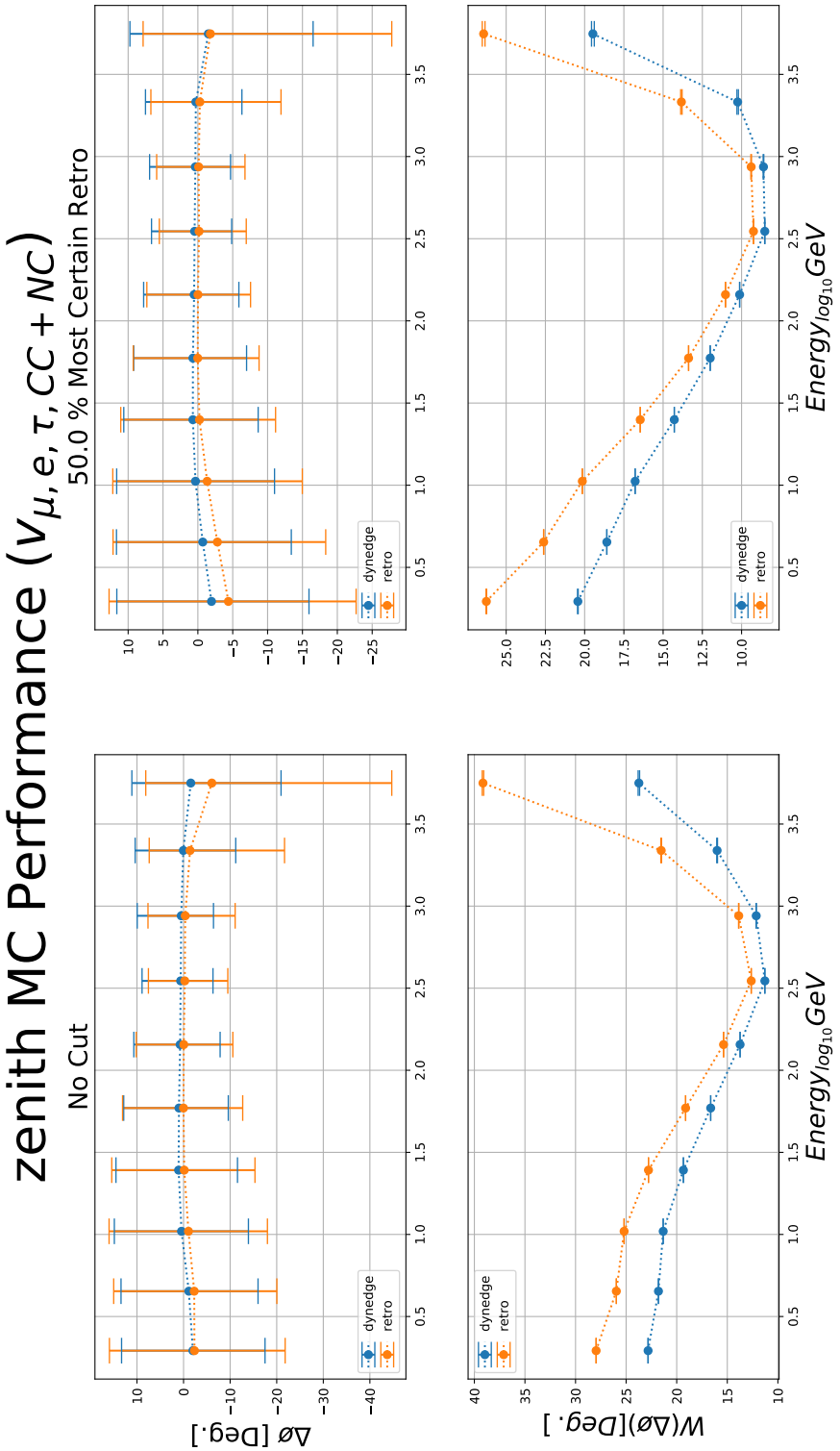

zenith MC Performance ($\nu_\mu, e, \tau, CC + NC$)
No Cut
50.0 % Most Certain Retro
Δø [Deg.]
W(Δø)[Deg.]
Energy$_{log_{10}}$GeV
dynedge
retro

FIGURE 8.2

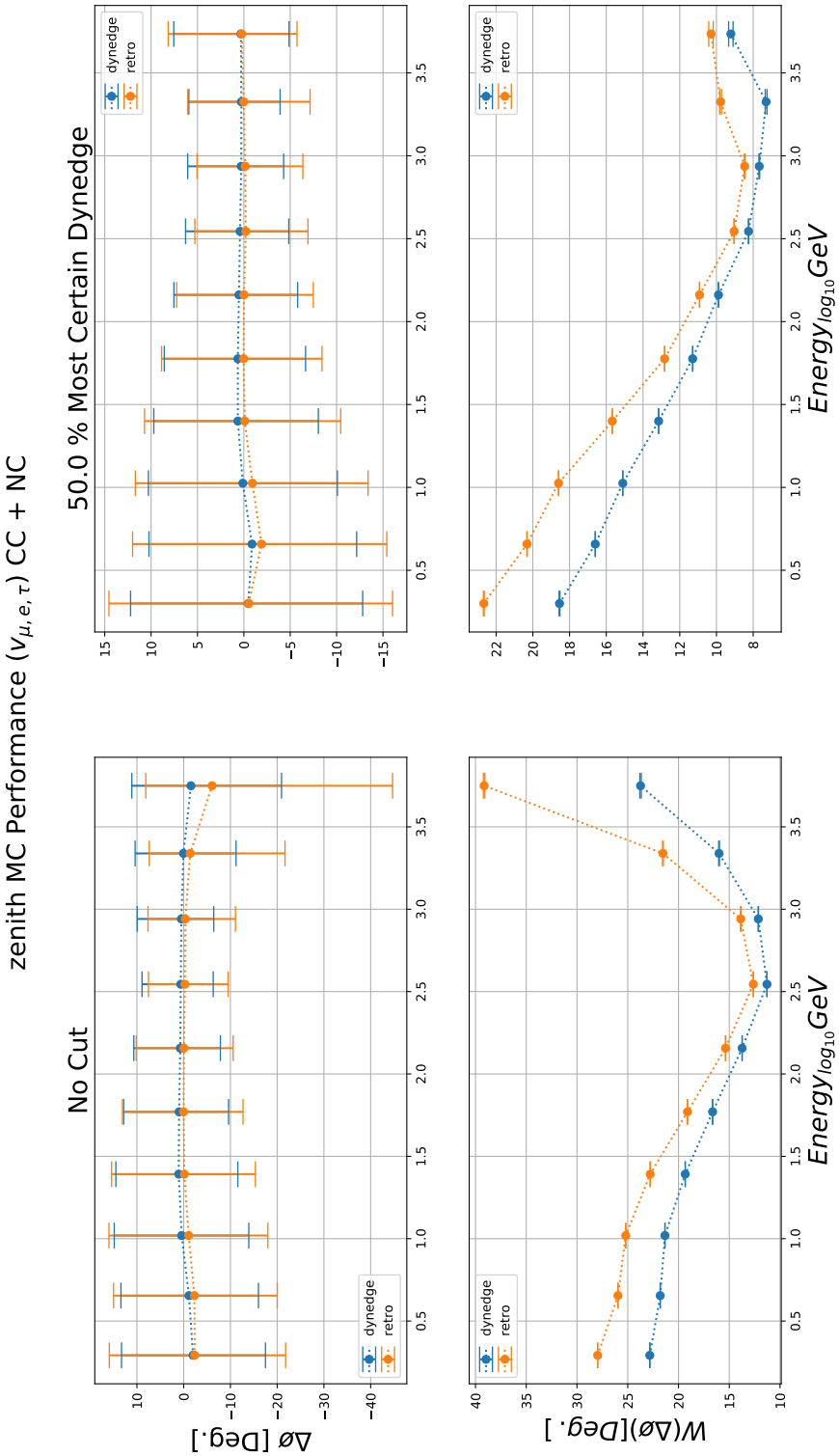

zenith MC Performance ($\nu_{\mu,e,\tau}$) CC + NC
No Cut
50.0 % Most Certain Dynedge
Δø [Deg.]
W(Δø)[Deg.]
Energy$_{log_{10}}$GeV
dynedge
retro

FIGURE 8.3

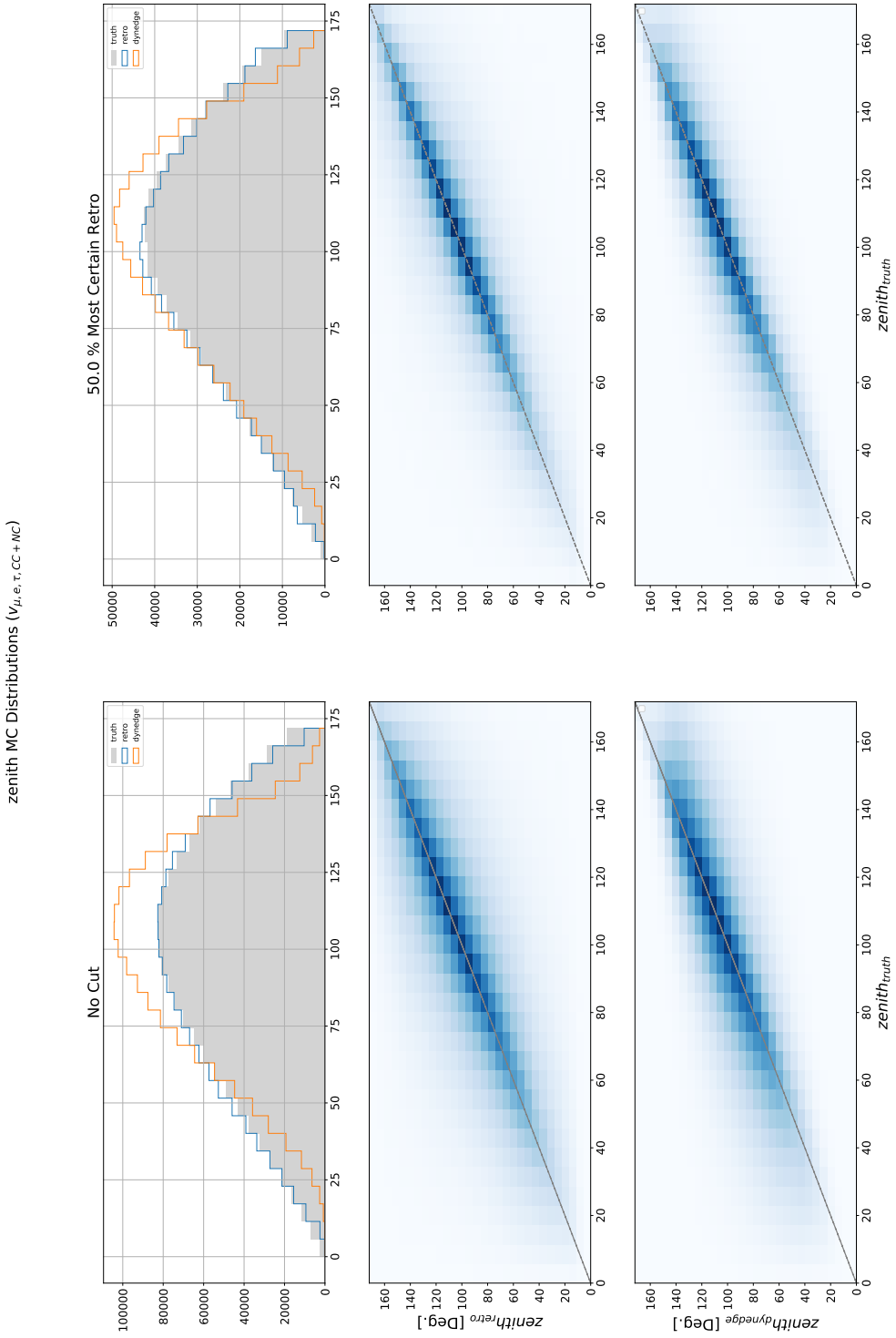

zenith MC Distributions ($\nu_{\mu,e,\tau,CC+NC}$)
No Cut
50.0 % Most Certain Retro
truth
retro
dynedge
$zenith_{truth}$
$zenith_{retro}$ [Deg.]
$zenith_{dynedge}$ [Deg.]

FIGURE 8.4

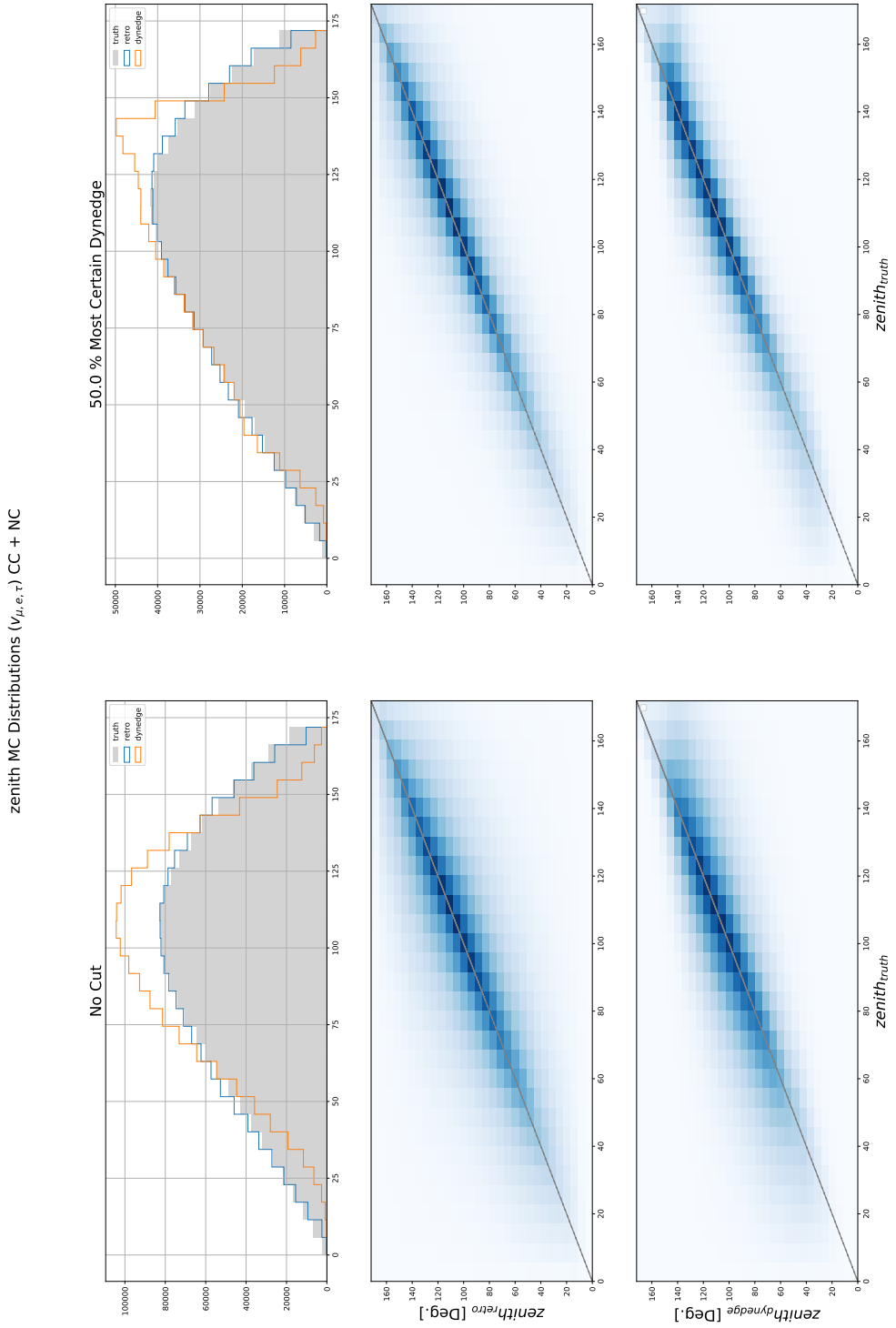

zenith MC Distributions ($\nu_{\mu,e,\tau}$) CC + NC
No Cut
50.0 % Most Certain Dynedge
truth
retro
dynedge
$zenith_{truth}$
$zenith_{retro}$ [Deg.]
$zenith_{dynedge}$ [Deg.]

FIGURE 8.5

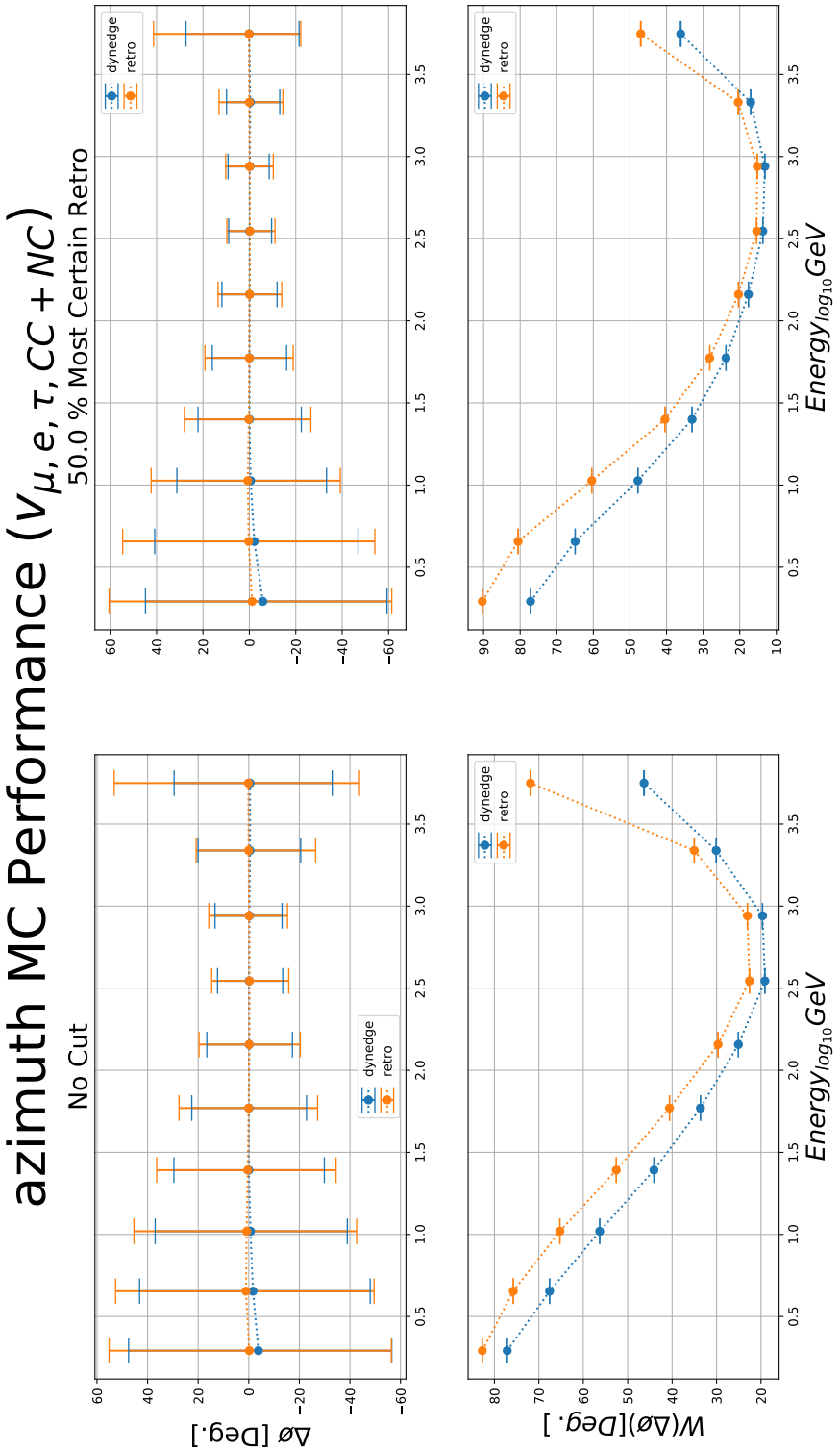

azimuth MC Performance ($\nu_\mu, e, \tau, CC + NC$)
No Cut
50.0 % Most Certain Retro
dynedge
retro
Δø [Deg.]
W(Δø)[Deg.]
$Energy_{log_{10}}GeV$

FIGURE 8.6

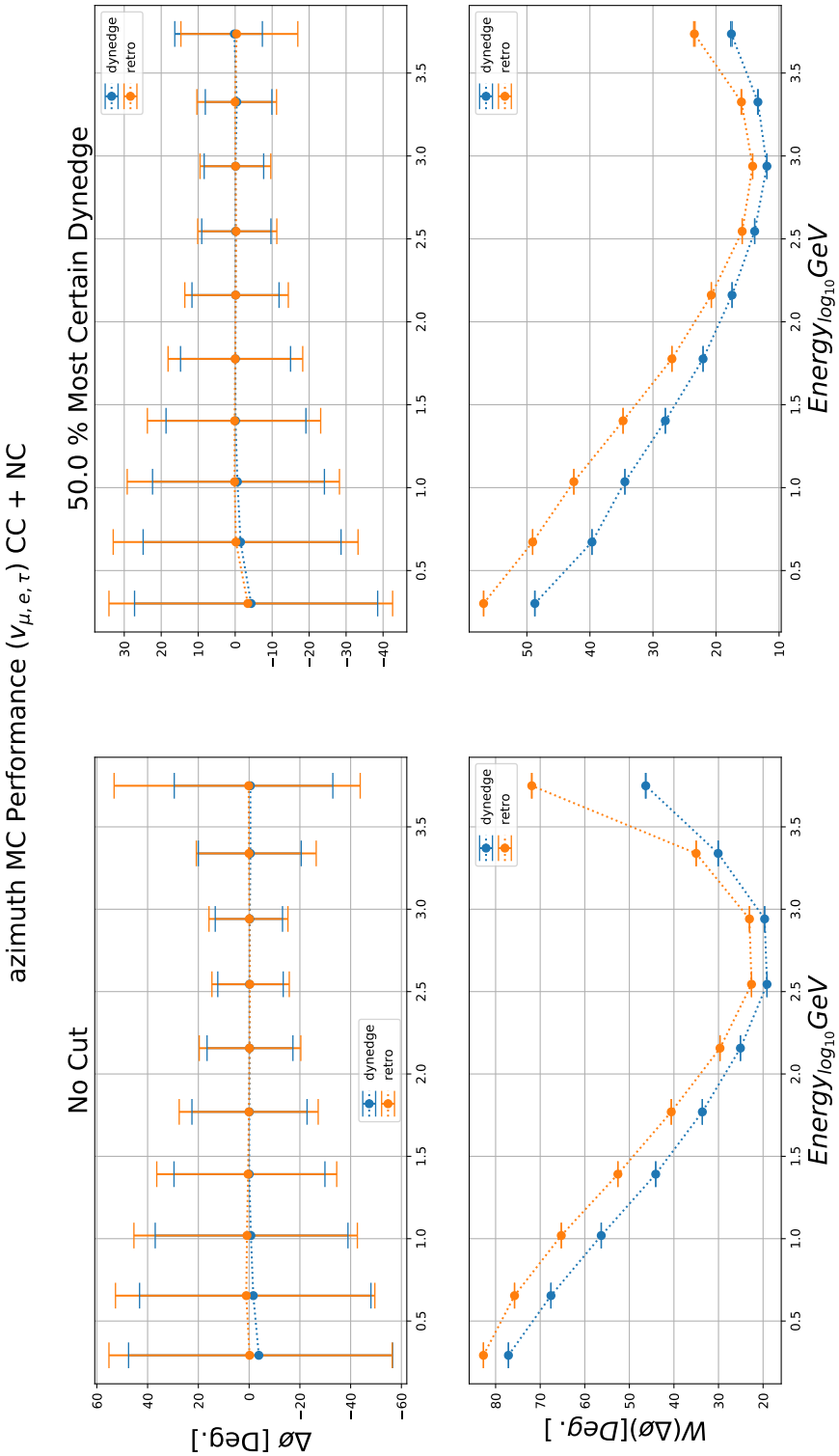

azimuth MC Performance ($\nu_{\mu,e,\tau}$) CC + NC
No Cut
50.0 % Most Certain Dynedge
Δø [Deg.]
W(Δø)[Deg.]
Energy$_{log_{10}}$GeV
dynedge
retro

FIGURE 8.7

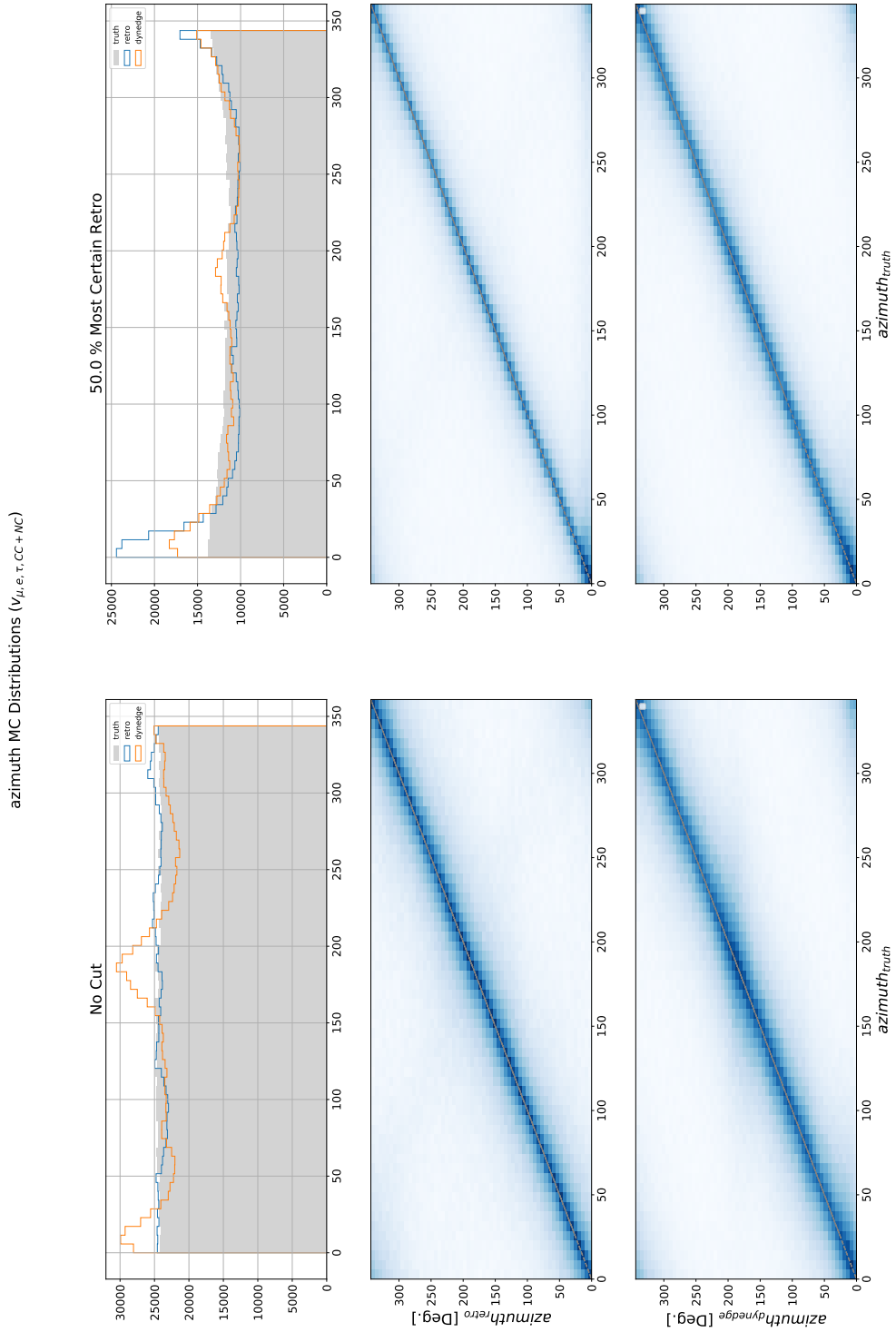

azimuth MC Distributions ($\nu_{\mu,e,\tau,CC+NC}$)
No Cut
50.0 % Most Certain Retro
truth
retro
dynedge
azimuth_truth
azimuth_retro [Deg.]
azimuth_dynedge [Deg.]

FIGURE 8.8

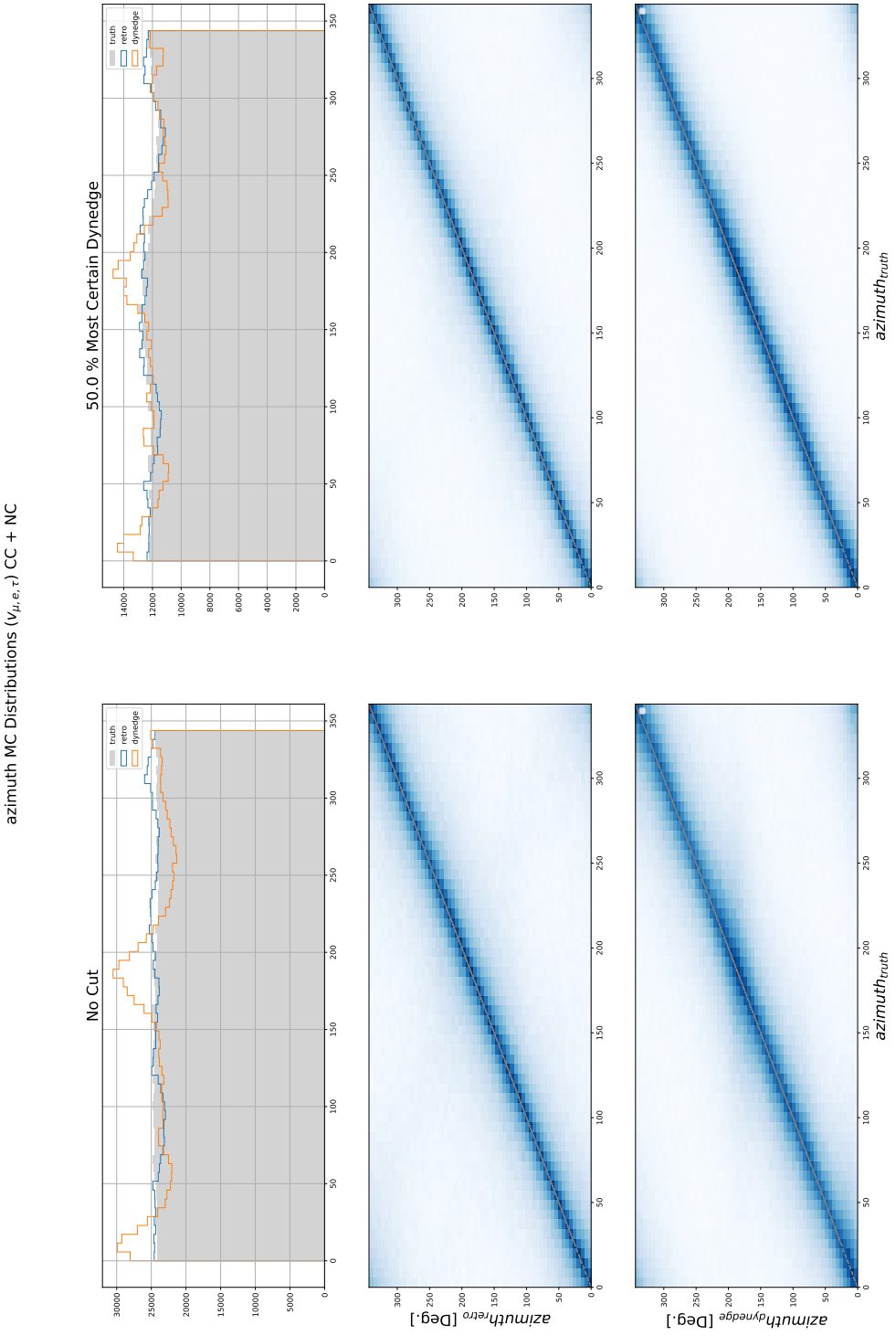

azimuth MC Distributions ($\nu_{\mu,e,\tau}$) CC + NC
No Cut
50.0 % Most Certain Dynedge
truth
retro
dynedge
$azimuth_{truth}$
$azimuth_{retro}$ [Deg.]
$azimuth_{dynedge}$ [Deg.]

FIGURE 8.9

Relative Improvement in Error Width

To better summarize the performance across all variables as compared to RetroReco, it was decided to create a relative improvement plot depicting

$$\text{relative improvement} = 1 - \frac{\text{width}_{\text{dynedge}}}{\text{width}_{\text{retro}}}$$

which is shown in Figure 8.10 for both the full regression and for the 50% most certain.

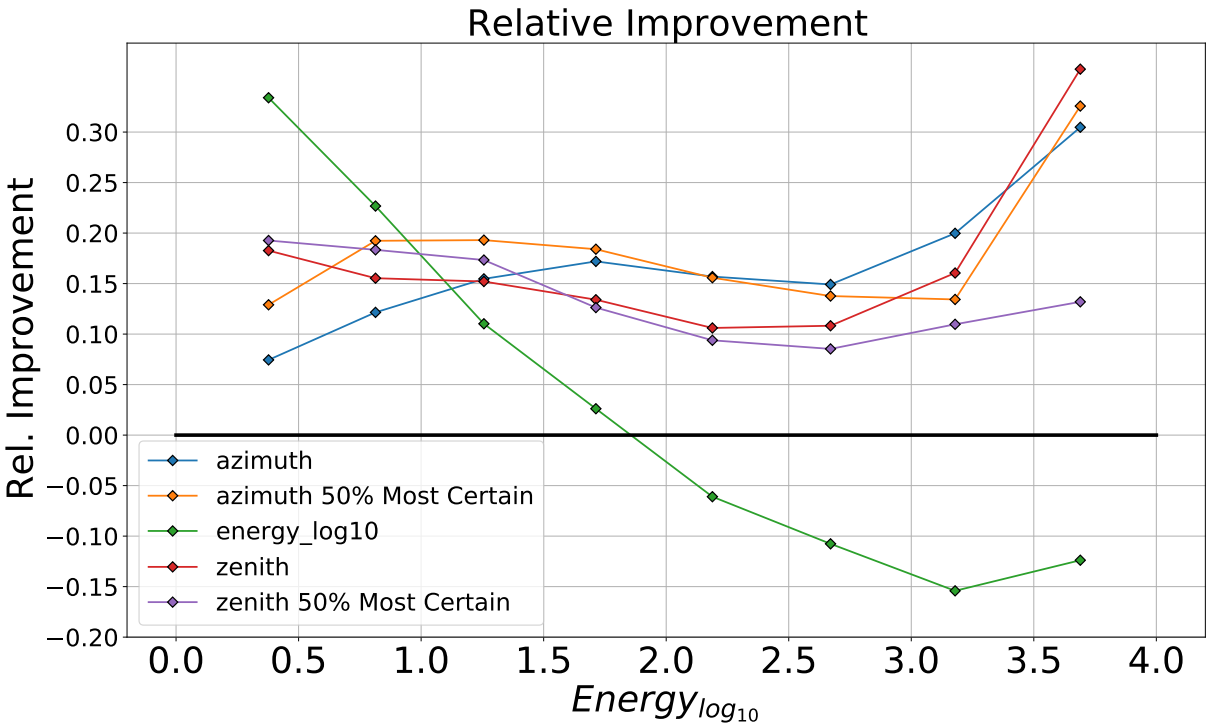


FIGURE 8.10: Relative improvement plot for error width in regressions of *azimuth*, *zenith*, *energy_log*10 for both cut and pre-cut regressions. 50% most certain is from uncertainties produced by *dynedge*.

Bear in mind that the most interesting part of the energy range is between [0,1.5] $\log_{10}$ GeV as this is where neutrino oscillation occurs. Starting with *azimuth* and *zenith*, one sees that there's an improvement on the entire energy range which is expected given the their performance plots in Figures 8.3 and 8.7. Interestingly, one can also see that while the *zenith* performance increases in the oscillation range after the cut in uncertainty, it actually decreases after 1.5 $\log_{10}$ GeV. On average, *dynedge* offers a 11.7%, 22.4% and 16.3% improvement for regressions of *azimuth*, *energy_log*10 and *zenith*, respectively, in the oscillation relevant energy range.

Classification Results

The $\nu - \mu$ classification model is merely a version of the final geometric architecture depicted in Figure 7.1 where the loss function is changed to

```
torch.nn.CrossEntropyLoss(reduction = 'mean')
```

which combines a logarithmic softmax with a negative log likelihood loss, a quite common choice of loss function for classification tasks[2]. This choice of loss function leads to a model that outputs a pseudo-probability for both PID classes. In the development of the classification model, no major deviations to the architecture of the final geometric model were explored due to time constraints, and because of the identical structure, none of the hyper-parameters, besides learning rate, were subject to further optimization tests. It is therefore plausible that increased performance of results shown in this section can be achieved by carrying out such tests, much of which resembles what's explained in the previous chapter.

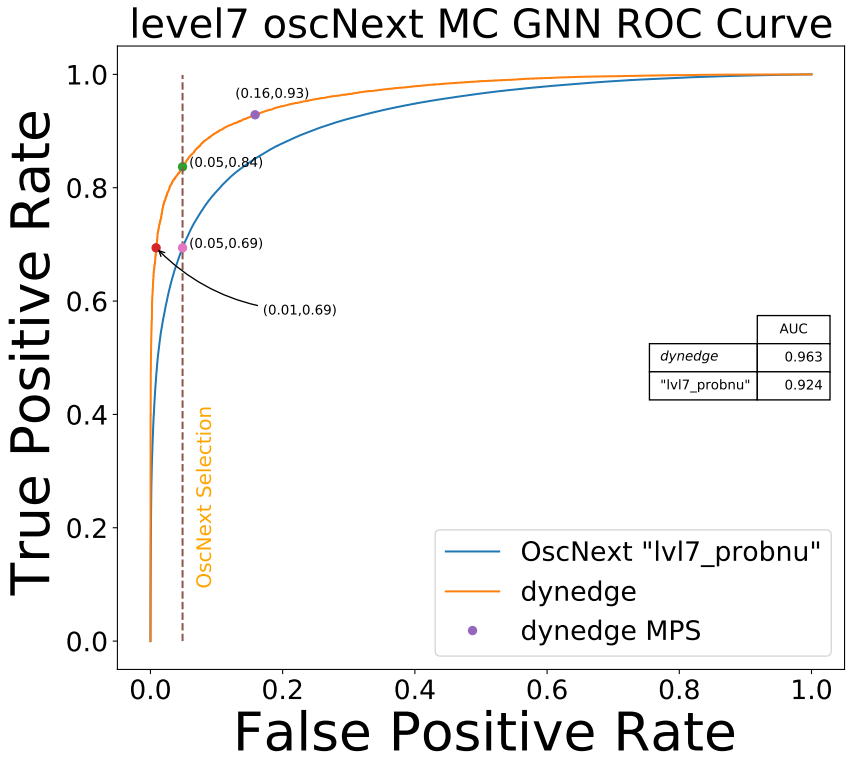


FIGURE 8.11: ROC Curve for the unbiased $\nu - \mu$ classification model. **"lvl7_probnu"** represents the oscNext level 7 muon/neutrino classifier.**dynedge MPS** corresponds to the PID selection where the event PID is concluded on the basis of the maximal estimated PID from *dynedge*.

It was initially pondered upon whether or not the training set for the classifier should biased. Because there isn't an even distribution in PID classes in real data, it was thought that creating a training set with a realistic PID distribution would lead to optimal results. However, it was found that the classifier worked best when the training set had an even distribution of classes, e.g an even amount of neutrino and muon samples. Therefore, the results shown in Figure 8.11 is from the unbiased model[3]. As evident, the *dynedge* classification offers either a 15% increase in true positive rate at the same false positive rate as the current oscNext classifier, or one fifth the false positive rate at the same true positive rate.

[2]Documentation can be found here

[3]This effect could also be because the chosen distribution for the biased model is sub-optimal. This is discussed further in Chapter 10

dynedge's Uncertainties

By the **central limit theorem**, the pull

$$g = \frac{\text{Prediction} - \text{True}}{\sigma} \tag{8.1}$$

should follow a unit Gaussian distribution. To test the validity of the uncertainties produced by the probabilistic regressions of *azimuth* and *zenith*, a set of comparative pull-plots is presented for *zenith* in this section. Similar figures for *azimuth* can be found in appendix. in Figures 8.12 and A.16, the pull associated with RetroReco's reconstructions are shown together with a unit Gaussian distribution for reference. **Please recall that this version of RetroReco produces inferior uncertainties as compared to other versions**.

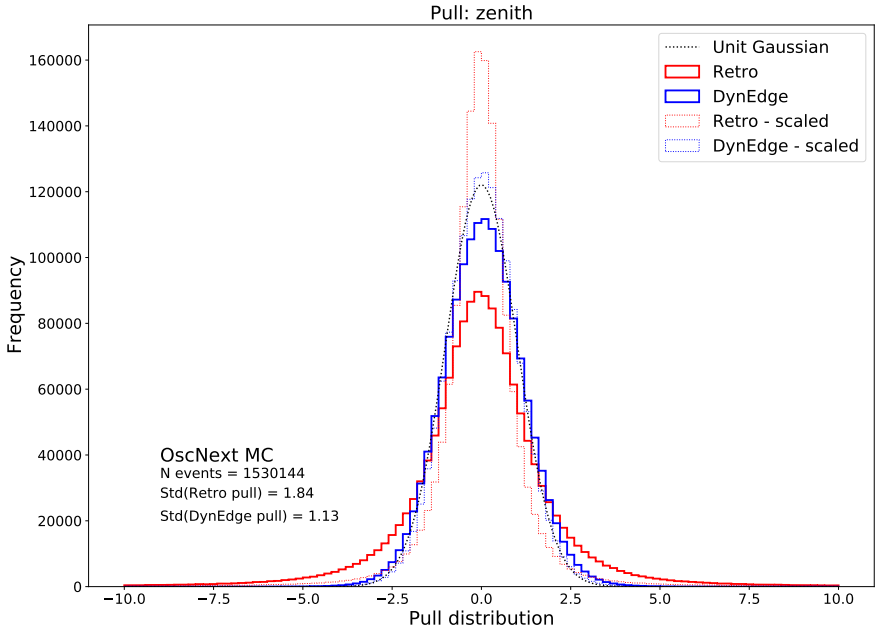


FIGURE 8.12: Pull plot for *zenith* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ - model.

Just by simply comparing the standard deviation of the *dynedge* pull distributions between *azimuth* and *zenith*, it's evident that the general quality of the error estimation is correlated with the reconstruction performance. This is no surprise as the model is asked to produce realistic uncertainties *together* with optimizing the relative angle between the sine-cosine vector representation of the angles.

However, it's quite clear that the pulls made on the basis of reconstructions produced by *dynedge* fits a unit Gaussian distribution better than this version of RetroReco. However, a more rigid test kindly proposed and first carried out by the supervisor of this work, **Troels C. Petersen**, is to check how the standard deviation of the pull changes as a function of the σ-percentiles. This is shown for *zenith* in

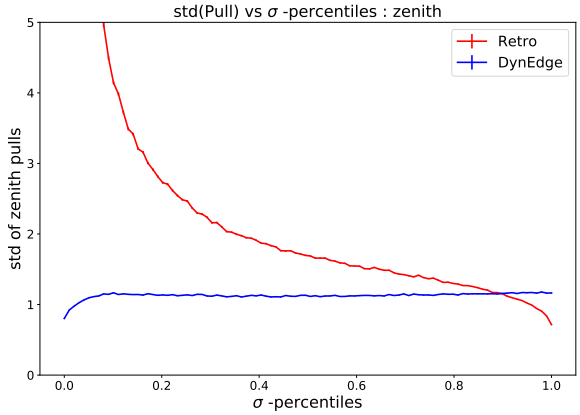


FIGURE 8.13: Running pull plot for *zenith* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model. Notice that only data satisfying $\sigma < 5$ has been displayed in this figure for both RetroReco and *dynedge*.

Figure 8.13. As evident, the standard deviation of the pull for the *zenith* regression is reasonable constant at around 1 - which is approximately the same value as seen in Figure 8.12. This suggests that the quality of the uncertainty estimation from *dynedge* is consistent. This is good news as this means that *dynedge* doesn't only offer many orders of magnitude in computational speedup, but also reconstructions with reasonable error estimates. In analytical work, this will be valuable as the reconstruction performance can be 'tuned' by simply selecting events with better uncertainties.

PID-dependency in Performance

Because a dependence on PID in reconstruction performance was noticed during development, it was decided to display this difference in performance for the MC results. To keep things brief, the difference in *zenith* reconstruction performance is shown in this section only. Similar plots for *energy_log*10 and *azimuth* is available in appendix.

The performance plot in Figure 8.14 aims to shed light on two things. Firstly, how

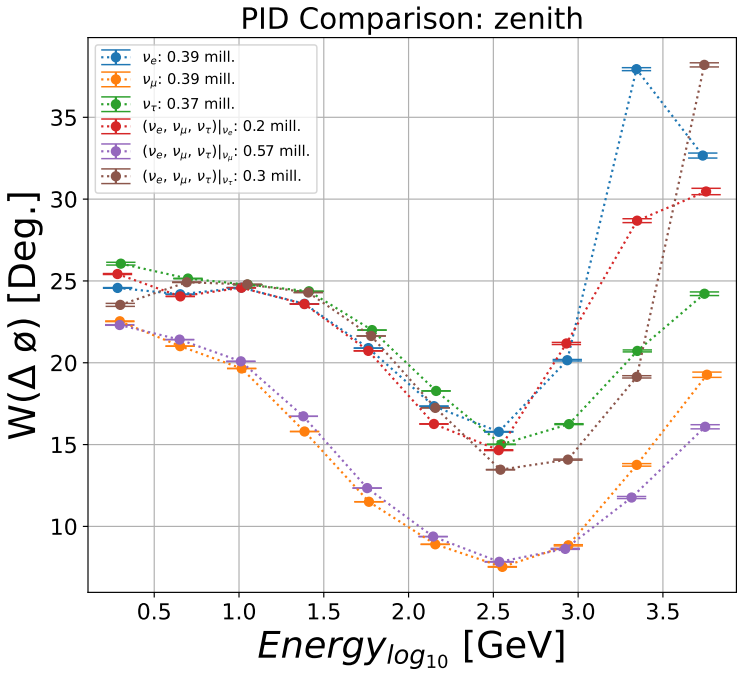


FIGURE 8.14: Performance plot showing PID dependence in regression performance for *zenith*. $(\nu_e, \nu_\mu, \nu_\tau)|_{\nu_\alpha}$ denotes a model trained on all neutrino types but evaluated only in α. Sample size in legend. Data contains both CC and NC events, at ratios of approx. 90% and 10%, respectively.

big of a difference can one expect to see in reconstruction performance for individual neutrino types? And secondly, what is the relative difference between a model that has trained to regress all neutrino types as compared to a model that has trained on a single type? As evident from Figure 8.14, muon neutrino events produces much better regression results as compared to it's tau and electron cousins. Also, while there's a slight difference for muon neutrinos, it seems like there's little performance gain (if any) to be expected in general from restricting a training sample to a single neutrino type, as compared to a model that trains on all types.

In an operational sense, it wouldn't make much sense to increase the complexity of the reconstruction process by having a regression model for each neutrino type when the difference is this small. If you find your way to the appendix and look at the same plot but for *azimuth* (Figure A.13) you'll see the same behavior as shown in this section. However, for *energy_log*10, it's difficult to see PID dependence in regression performance.

MC Neutrino Oscillation

By using oscillation weights attributed to the oscNext events, it's possible to produce MC neutrino oscillation plots by considering the cosine of the reconstructed *zenith* angle. The weights are contained in physics frames in

```
frame['I3MCWeightDict']["weight"]
```

With reconstructions that seems to rival those of RetroReco, which has been the backbone of neutrino oscillation analysis at IceCube, it was thought to include an oscillation plot in this section of the work to showcase what the increased reconstruction performance from utilizing *dynedge* means for neutrino oscillations.

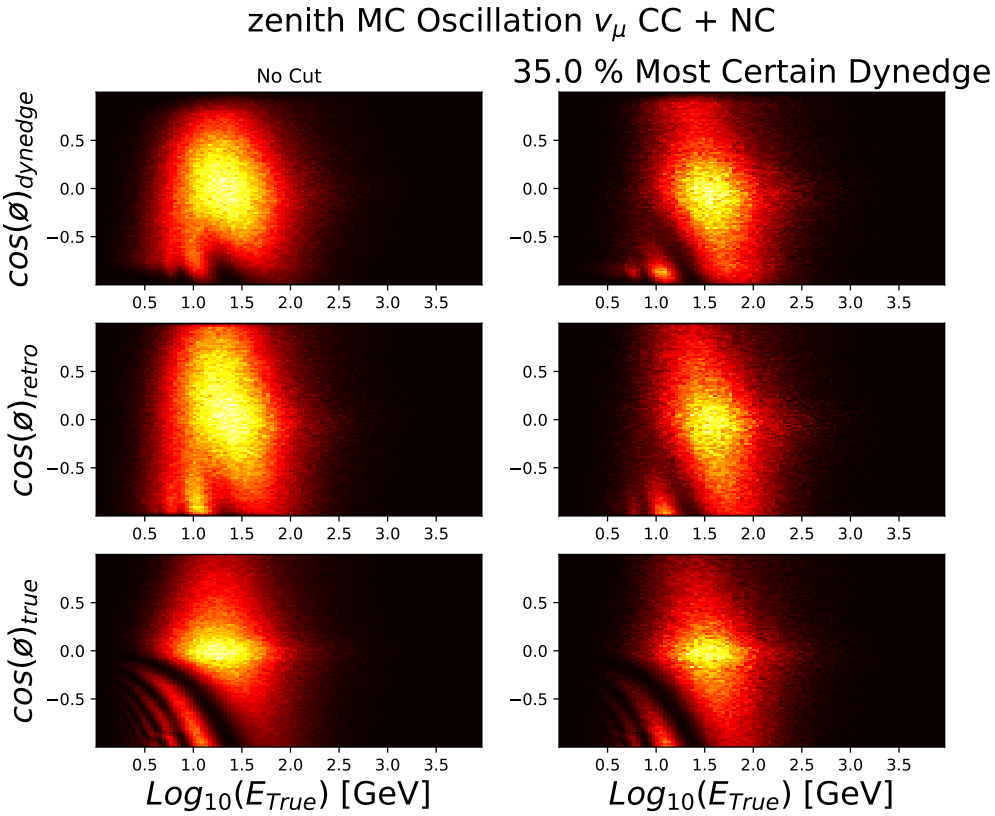


FIGURE 8.15

The results are displayed in Figure 8.15. The oscillation plot shows $\cos(zenith)$ from RetroReco, *dynedge* and from truth. Results are shown with and without cut in the uncertainties produced by *dynedge*. The *zenith* reconstruction from *dynedge* originates from the $(\nu_e, \nu_\mu, \nu_\tau)\,|_{\nu_\mu}$-model.

Chapter 9

Application on Upgrade Data

In the following I'll describe the application of the GNN on upgrade data and the challenges I met in doing so.

Waking Up to Noise

The only adaptations needed to make the GNN work on upgrade data is to expand the dimensions of the node features to include the upgrade-only feature data shown in Table 3.1. In theory. After reviewing the first reconstruction, it became painfully apparent that strange behavior from the GNN began showing. As seen in Figure

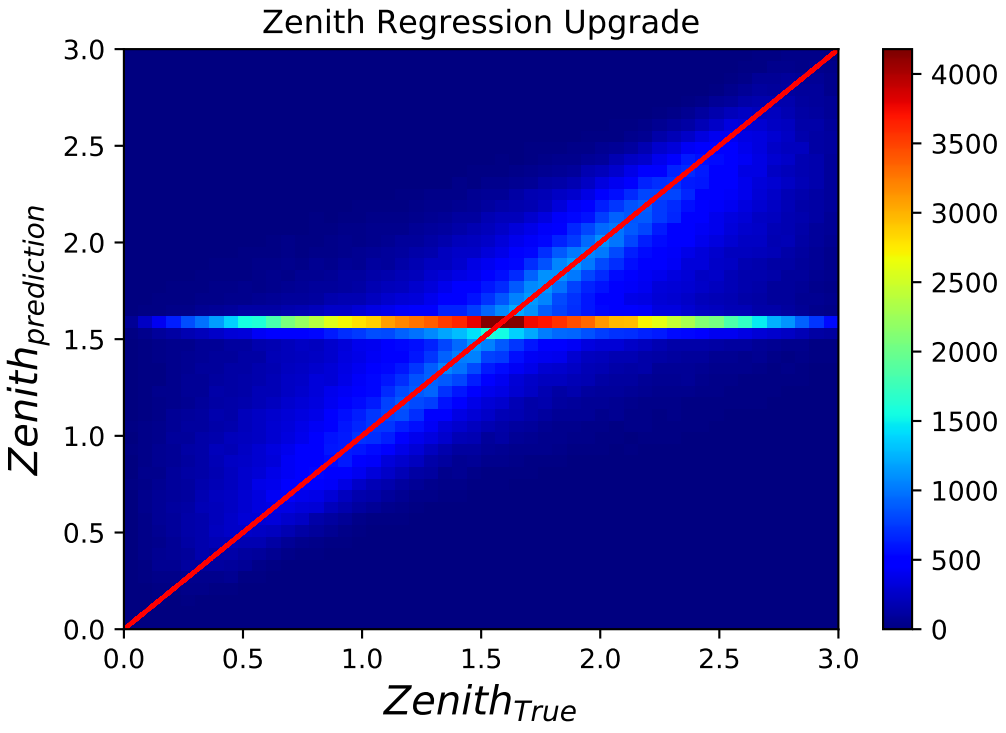


FIGURE 9.1: First *zenith* reconstruction on Upgrade data. Y-axis denotes *zenith* reconstruction and x-axis true *zenith* target values. Red line represents ideal alignment. Units in Rad.

9.1, the result were *utterly* useless. While such a result can make one briefly doubt that the sun will ever rise again, a review of code ruled out the behavior to be due to mistakes. An investigation into the data showed that an impressive amount of

activity among the new mDOMs and the dEggs for events where virtually no ordinary DOMs were active. As seen in Figure 6.3, there's a correlation between the number of active DOMs and the energy associated with an event - and it was then thought that the issue was due to pure-noise events in the upgrade data. Initially a scheme to identify the pure-noise events revolved around a k-fold training and validation scheme where reconstructed events lying in the 'bad-band' of the *zenith* reconstruction, as seen in in Figure 9.1, were tagged and removed from the data set. A set of 'clean' events, e.g. those that didn't inhabit the bad-band, were then trained and predicted upon. The result of this naive approach were better but did not fix the problem. Viewing this approach in retrospect one should add that it is **dangerous** to simply remove events that leads to poor performance for a GNN as such a scheme would make even the worst of algorithms look good. The fact that it didn't lead to massive improvements on the reconstruction performance is a strong indicator of how troublesome the data is. Also, it indirectly shows that the issue at hand is noisy mDOM and dEgg activations within meaningful events, that isn't filtered out during the SRT-cleaning.

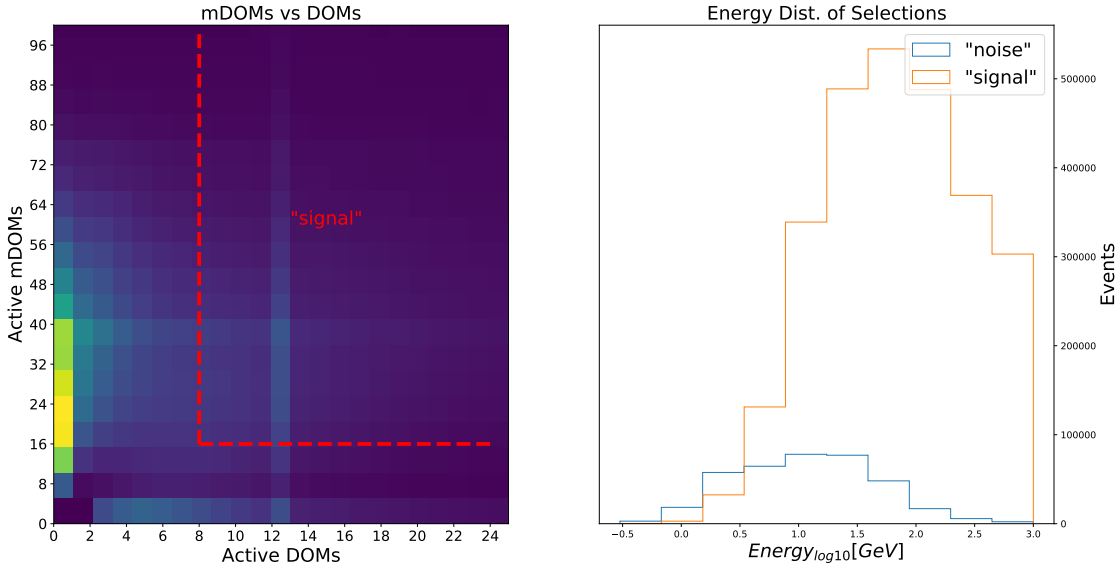


FIGURE 9.2: To the left: 2D histogram of mDOMs vs. ordinary DOMs. On the right: distribution of energy for **"noise"**, which represents events with less than 8 active DOMs and 16 mDOMS and **"signal"** which represents events with more than described cut. Data is SRT cleaned.

As seen left in figure in Figure 9.2, the vast amount of mDOMs activate in events where less than three DOMs register signal. This behavior is expected as the reason for installing the new DOM types, as mentioned in 2, is for the detector to be able to register events that the current detector can't and to increase signal for those that it can. But the sheer amount of activation suggests that the vast majority of mDOM activations are noise. This was the topic of an IceCube technical call on upgrade reconstruction on 11th November 2020, where the above results were shared. A temporary selection on data were proposed by TUM PhD student **Martin Ha Minh**:

- $n\_mDOMs \geq 16$

- $n\_DOMs \geq 8$

If one plots the energy distribution of the events on either side of the cut, as depicted in right figure of Figure 9.2, it is evident that the **"noise"** events are real neutrinos covering the entire energy spectrum. As such, one is presented with the choice of 'solving' the reconstruction issue by simply making cuts in the data based on activations or to attempt to address the underlying problem, namely by identifying noise activation as there currently exists no MC labeling for this. As a first step, the GNN was run on either side of the proposed cut to evaluate its quality.

FIGURE 9.3: 2D histogram of true vs predicted *zenith* on 'good side' of the proposed data cut. Units in degrees.

As seen in Figure 9.3, the cut removes the 'bad band' of horizontal nonsense that was present in the uncleaned *zenith* regression. As expected, the run on 'the bad side' of the cut resulted in a horizontal line, and as such seems to very effectively distinguish between reconstructable and un-reconstructable events. However, when comparing reconstruction performance with DeepCore, as shown in Figure 9.4, it's evident that the upgrade regressions are inferior to DeepCore results in part of the energy scale that is relevant for oscillations - which is problematic as increased performance in this part of the energy spectrum is one of the arguments for installing the upgrade. A likely explanation for this effect is that enough noise is still present in data selection to influence results. Therefore an investigation into possible pulse cleaning methods were launched.

Pulse Cleaning Attempts

It's been attempted in this work to address pulse cleaning of current upgrade data by using **local coincidence** - which proved to be poor as there's simply too much noise present in the data. It has also been attempted to use a 'semi-supervised' cleaning block in tandem to the ordinary regression to filter out noise based on three ideas:

1): Produce a score $[0, 1]$ for each node in a graph and multiply the node features with

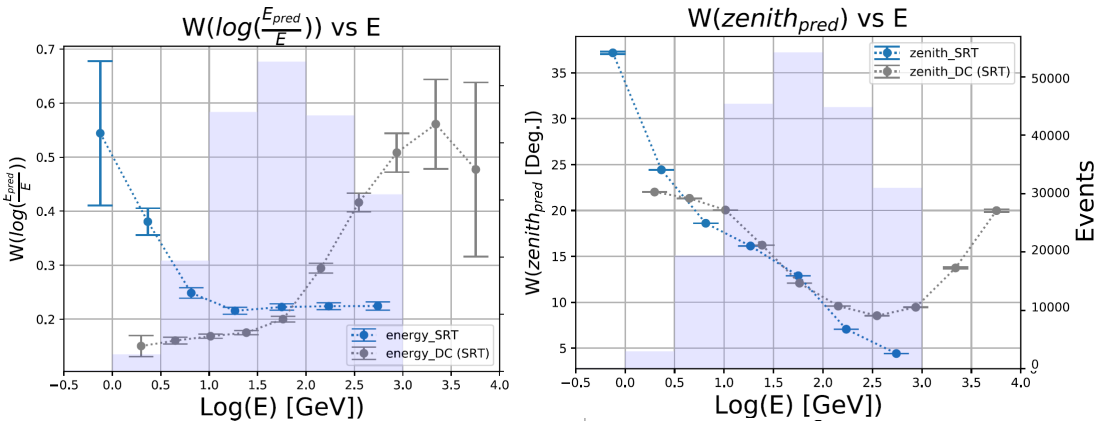


FIGURE 9.4: *energy_log*10 and *zenith* reconstruction performance plots on 'the good side' of the proposed cut. DeepCore reconstruction performance shown in grey for comparison. Note that the DeepCore results originates from different events and as such only serves as a rough comparison. Upgrade model trained only on $dom\_x, dom\_y, dom\_z, dom\_charge, dom\_time$ to be comparable to DeepCore.

this score before passing it through the ordinary model architecture. This would effectively allow the GNN to dampen contributions from nodes that didn't optimize the loss function. This failed.
2): Remove nodes entirely from the graph before passing it on to the model by using the score described in **1)**. This would effectively allow the GNN to remove any nodes that didn't contribute to the optimization of the loss function. This failed as the GNN never learned to keep a realistic amount of nodes - as it at best kept only around 18% of available nodes - a far too low amount to be realistic.
3): Consult industry-standard and current research in point cloud cleaning. Quite a bit of literature and promising methods exists, such as [72], but requires both clean and noised data for training, which isn't possible to produce since noise activations isn't labeled in icecube data at the moment. Most methods seem to revolve around cleaning bias in node features rather than removing nodes. For these reasons this approach was not taken.

With an amounting number of failed attempts, it was decided to return to what was known to work: The proposed cut. What the cut from **Martin Ha Minh** showed was that the general noise profile of the data can effectively be cut into two parts - but what's missing is an exact explanation for why this cut works well. What's known is that the noise comes in the form of false DOM activations, and that the vast majority of the total activations come from mDOM's simply because they carry more pmt's. For this reason **mDOM activation diagrams** were made on individual events on both sides of the proposed cut in the hope that suspicious activation patterns could be spotted manually. After manually reviewing a painful amount of events on both sides of the cut, a general pattern seemed to emerge. An example of suspicious mDOM activity is shown in Figure 9.5. The Y-axis of Figure 9.5 is in essence a histogram of individual mDOM activations in the given event. Since the X-axis denotes time, the dots will appear in a horizontal pattern if the same mDOM activates several times during the event. Since the mDOMs carry multiple pmts, all of which may activate during the event for valid reasons, each pmt is assigned a unique color

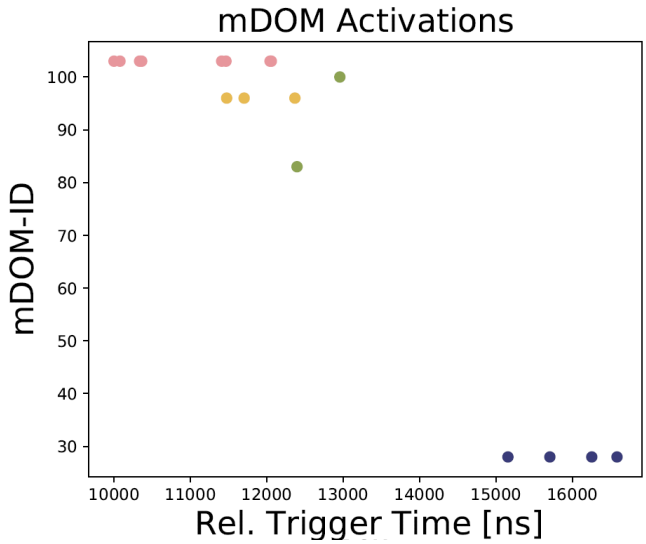


FIGURE 9.5: An example of an mDOM activation diagram for a single event on 'the bad side' of the proposed cut containing a suspicious activation pattern. Y-axis denotes unique mDOMs in the given event. X-axis shows the trigger time. Color-coding indicates which pmt. Data is SRT cleaned.

such that individual pmts can be identified using the color coding. In the case of the event displayed in Figure 9.5, which contains a total of 16 mDOM activations, it can be directly seen that those activations originate from only 5 unique pmts and that the majority originates from the pink and blue pmts. While it's possible for multiple same-pmt hits to appear from signal, it seems highly unlikely that this would be the case for the blue and pink pmt. One possible explanation for this activation pattern might lie in the fact that the SRT cleaning is originally developed for cleaning ordinary DOMs and not multiple-pmt DOMs, and that this specific implementation of SRT was a temporary solution to the new upgrade data. On a technical level, it seem plausible that the SRT cleaning doesn't check (or maybe not well enough) whether or not the causality is checked on the basis of same-pmt activations. For an initial study it was decided that pmts activating more than three times during a single event would be tagged and labeled as **"noise"**. With this definition, a quick density calculation showed that on 'the good side' of the proposed cut, 29% of the DOM activations within an event originates from **"noise"** and 40% on 'the bad side' of the proposed cut. While there is a difference of 11% in **"noise"** between the cuts, it seems unlikely that this should be whole explanation behind why the 'bad-side' seems unreconstructable and the 'good-side' isn't. One could also argue that this adhoc choice of 'three or more' pmt activations might be too loose. Either way, as an initial study to test if these activation patterns are genuine or noise related, a **first occurrence** filter (FO) were applied to SRT cleaned upgrade data such that any single pmt activating more than three times would be replaced with it's first activation. Such a filter would reduce the 16 activations in Figure 9.5 down to 7, since only the first pink and blue pmt activations would be kept. Average activations pr. event on 'the good side' of the proposed cut is available in Table 9.1, with comparison to SRT and raw activation rates.

As seen in the performance plots in Figure 9.6, the FO filtering leads to quite big

Cleaning Method	avg. mDOM	avg. dEgg	avg. DOM
SRT + FO Filter	36.94	13.02	18.16
SRT	58.34	13.77	20.90
No SRT (baseline)	75.65	15.74	48.81

TABLE 9.1: Average pulses (active DOMS) in each event on 'the good side' of the cut by cleaning method. 'No SRT' indicates no pulse cleaning, SRT + FO Filter represents the method developed and examined in this work.

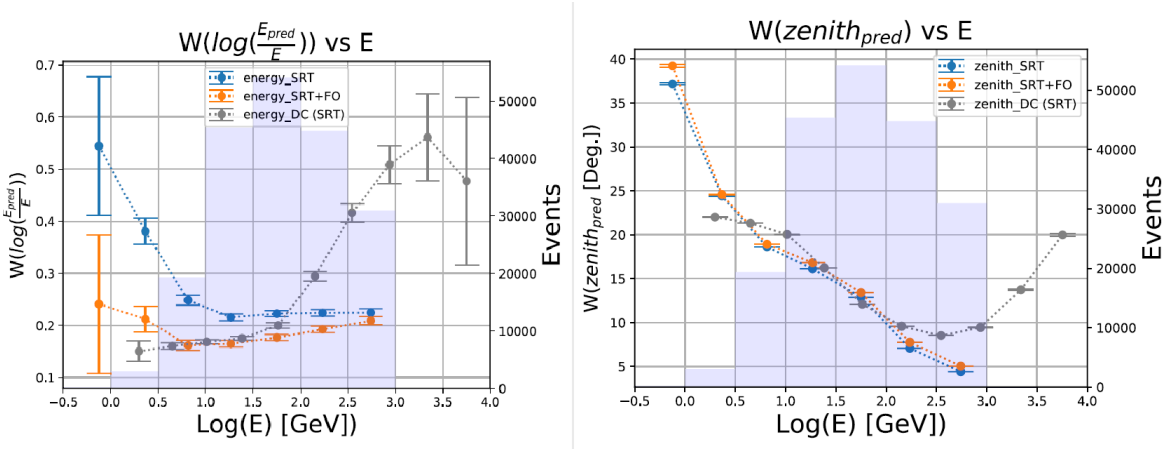


FIGURE 9.6: *energy_log*10 and *zenith* reconstruction performance plots on 'the good side' of the proposed cut. DeepCore reconstruction performance shown in grey for comparison. $SRT + FO$ denotes model trained on pulse cleaned data using the FO-filtering after SRT. Note that the DeepCore results originates from different events and as such only serves as a rough comparison. Upgrade model trained only on $dom\_x, dom\_y, dom\_z, dom\_charge, dom\_time$ to be comparable to DeepCore.

improvements in the prediction width of *energy_log*10 and makes the performance comparable to DC results in the high-statistics area of the oscillation region. However, when inspecting the prediction width on *zenith* regression, a small decrease in performance is visible. While this might seem counter-intuitive, an explanation might lie in the fact that there's a correlation between number of activated DOMs and energy, while such a correlation doesn't exists for angular results. With this in mind, one could argue that the big increase in performance on *energy_log*10 regression as a result of the FO-Filter might simply be because the filtering itself strengthens the correlation between energy and DOM count. In essence, it might not matter so much which DOM is kept but rather that the number of DOMs is consistent. For *zenith* however, the geometric picture represented by the graphs is sensitive to false activations, and as such keeping the wrong activation would decrease performance. With this interpretation, one could argue that perhaps keeping the 'first occurrence' of a pmt if it activates more than three times is on average wrong, as a more complex cleaning scheme is needed.

Because noise labeling is currently underway, it was decided to not pursue this problem further.

Chapter 10

Application on Real Measurements From IceCube

Evaluation Data Details

The MC data used in this chapter is contained in i3 files located at cobalt in the following directories:

```
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/120000"
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/140000"
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/160000"
"/data/ana/LE/oscNext/pass2/muongun/level7/130000"
```

The real measurements used is the full **IC86.11** data sample totaling approx. 64.000 events, and is located at

```
"/data/ana/LE/oscNext/pass2/data/level7/IC86.11"
```

The rates used to distribute PID's to mimic what's expected at level 7 oscNext filtering is contained in Table 10.1

PID	**lvl7 rate** (10^{-6} Hz)
μ	40
ν_μ	700
ν_e	200
ν_τ	60

TABLE 10.1: Rates at level 7 oscNext filtering obtained by manually reading 3.1

Classification between real data and MC data

Initially, when the first real data samples had to be put through the pre-processing pipeline, a question emerged: Should one fit and transform a new **sklearn.preprocessing.RobustScaler** to the real measurements to ensure unit variance and mean zero of the input, or should one simply transform the real measurements using the **sklearn.preprocessing.RobustScaler** already fitted to the training data? To answer this question a test were constructed where the IC86.11 measurements would be blended with an equal amount of MC neutrino events from the 8.1 million lvl7 sample used for regression. The MC sample were constructed such that the relative rates of neutrino types matched what one would expect at level7 filtering. One experiment had a new scaler fitted to the IC86.11 measurements and another experiment simply used the scaler fitted to the

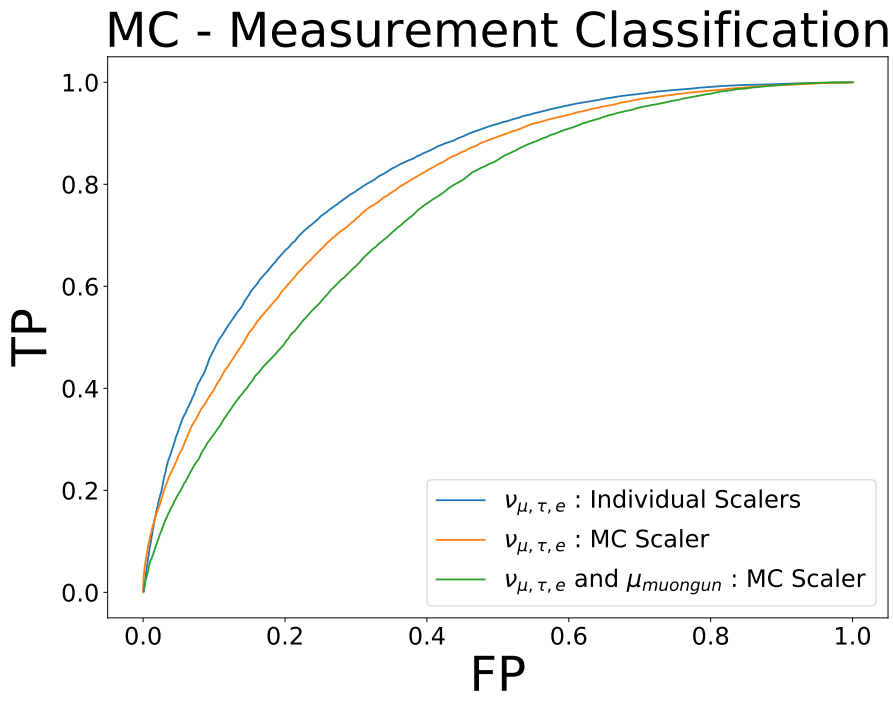


FIGURE 10.1: ROC curve for MC - Real measurement classification. $\nu_{\mu,\tau,e}$ *Individual Scalers* indicates the run where the IC86.11 sample were transformed using a scaler that was fitted to IC86.11. $\nu_{\mu,\tau,e}$ *MC Scaler* represents the run where the IC86.11 sample were transformed using the same scaler that were used to transform MC data. $\nu_{\mu,\tau,e}$ *and* $\mu_{muongun}$ *Individual Scalers* represents the experiment where the IC86.11 sample were transformed with the scaler that was used to transform MC data and with muons added to the MC events by including muongun data. **FP** represents False Positive and **TP** reprents True Positive. Axes in percent.

MC events to transform the IC86.11 sample. The idea was then this: The best method of scaling would then be the scaling leading to the worst mc / real measurements classification ROC-curve, since a bad ROC curve indicates difficulty in distinguishing between the MC and real data.

As seen in Figure 10.1, using the **sklearn.preprocessing.RobustScaler** that is fitted to the MC data to transform the IC86.11 sample produces the worst performing classification. For this reason the MC scaler was chosen as the transformer applied on real data. However, as depicted by the orange curve in Figure 10.1, simply modelling level 7 data with neutrinos only doesn't produce a great fit to real data. To get a gauge on the agreement effect by including muons, muons from the **muongun** simulations were included in a separate run, as depicted by the green curve in Figure 10.1. As evident, this greatly decreases the classification model's ability to distinguish between MC and real measurements, despite muons making up only around 4% of events at level 7. It should be noted that Figure 10.1 cannot be considered a rigid test of mc-data agreement as the deviation between the green curve and the (ideal) straight line indicating complete inability to distinguish mc and real data, has multiple contributing factors besides the quality of simulation:

1) Perhaps most obviously - the rates in Table 10.1 is produced by visual inspection of Figure 3.1 and can therefore only be considered approximations. This crude method of producing the PID distributions in the MC data for this classification task is likely the greatest contributing factor to the difference between the the green curve and the ideal straight line. In theory though, one could use this method to tweak the PID distributions in the MC set until a straight line is observed and obtain thereby a reasonably quantified measure of the mc-data agreement.
2) The rates depicted in Figure 3.1 is produced by applying the oscNext filtering to MC data and it's therefore likely that there exists small deviations in the rates observed in MC data and in real data. These possible deviations is essentially a measure of the MC and real data agreement.
3) The MC data used in this classification does not contain pure noise events.

Muon / Neutrino Classification Results

The unbiased classifier from Chapter 8 was evaluated on the IC86.11 sample.

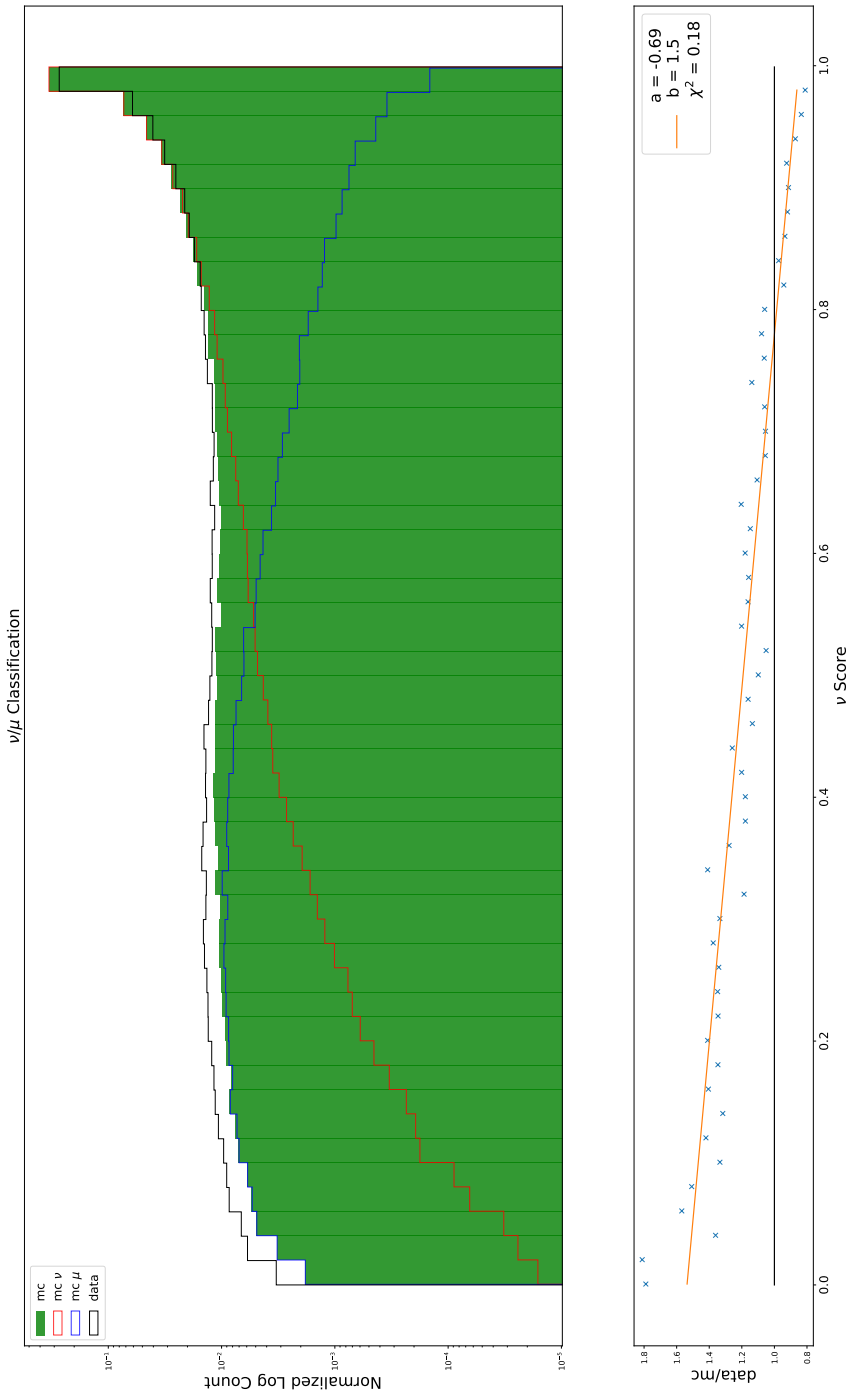
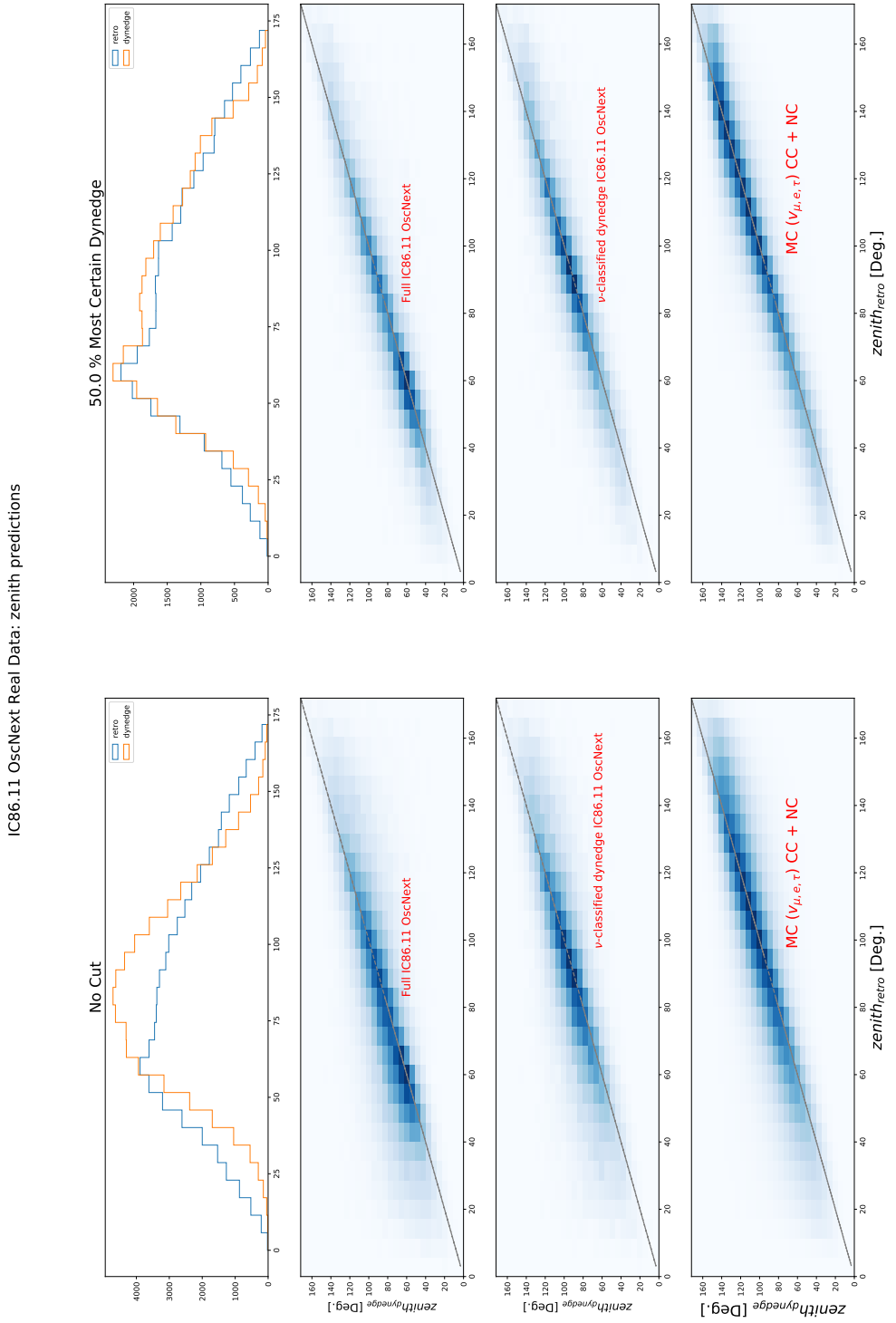


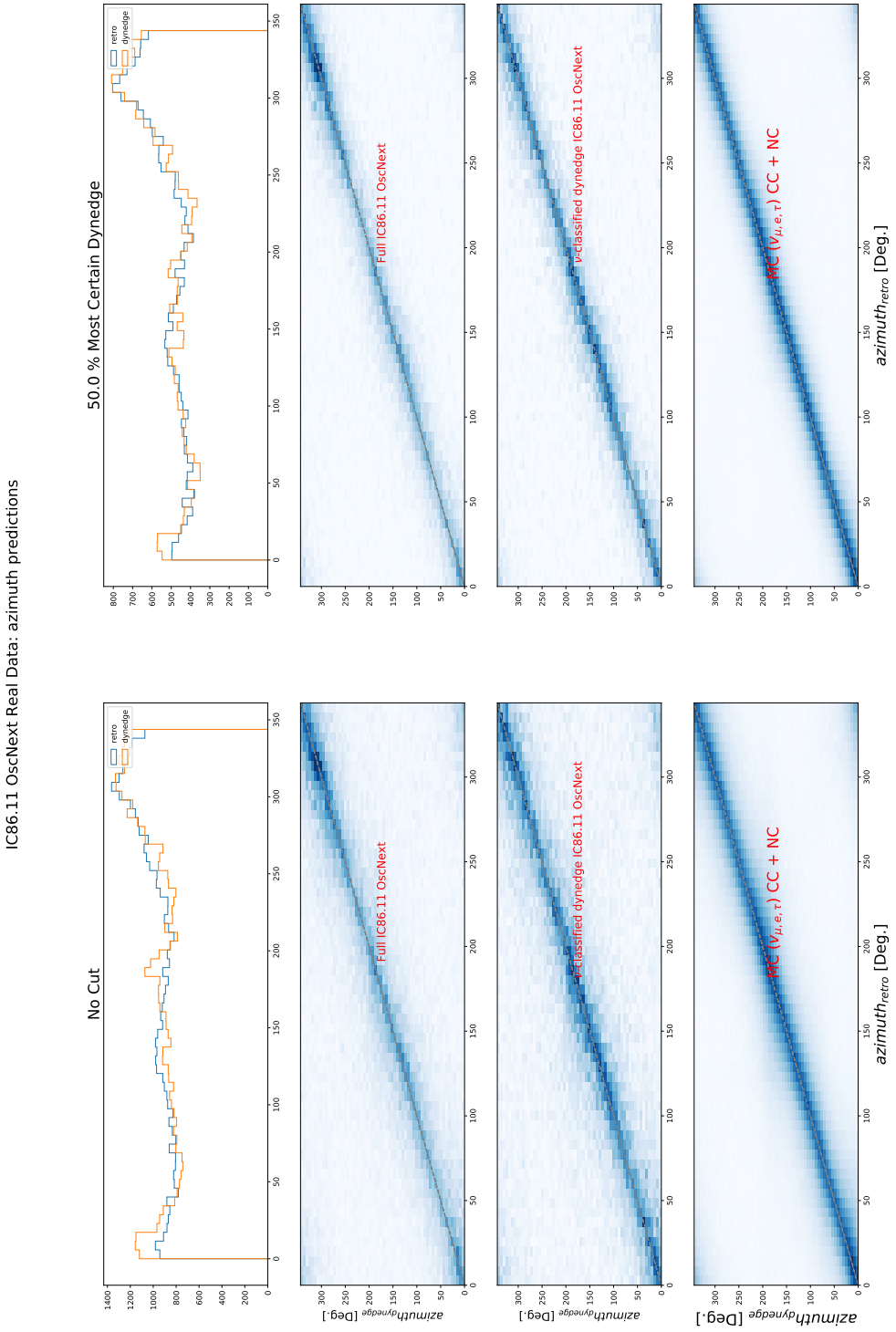
FIGURE 10.2: Distribution of ν score from the unbiased ν/μ classification model. Black represents IC86.11 measurements. PID of MC data represented by blue and red.

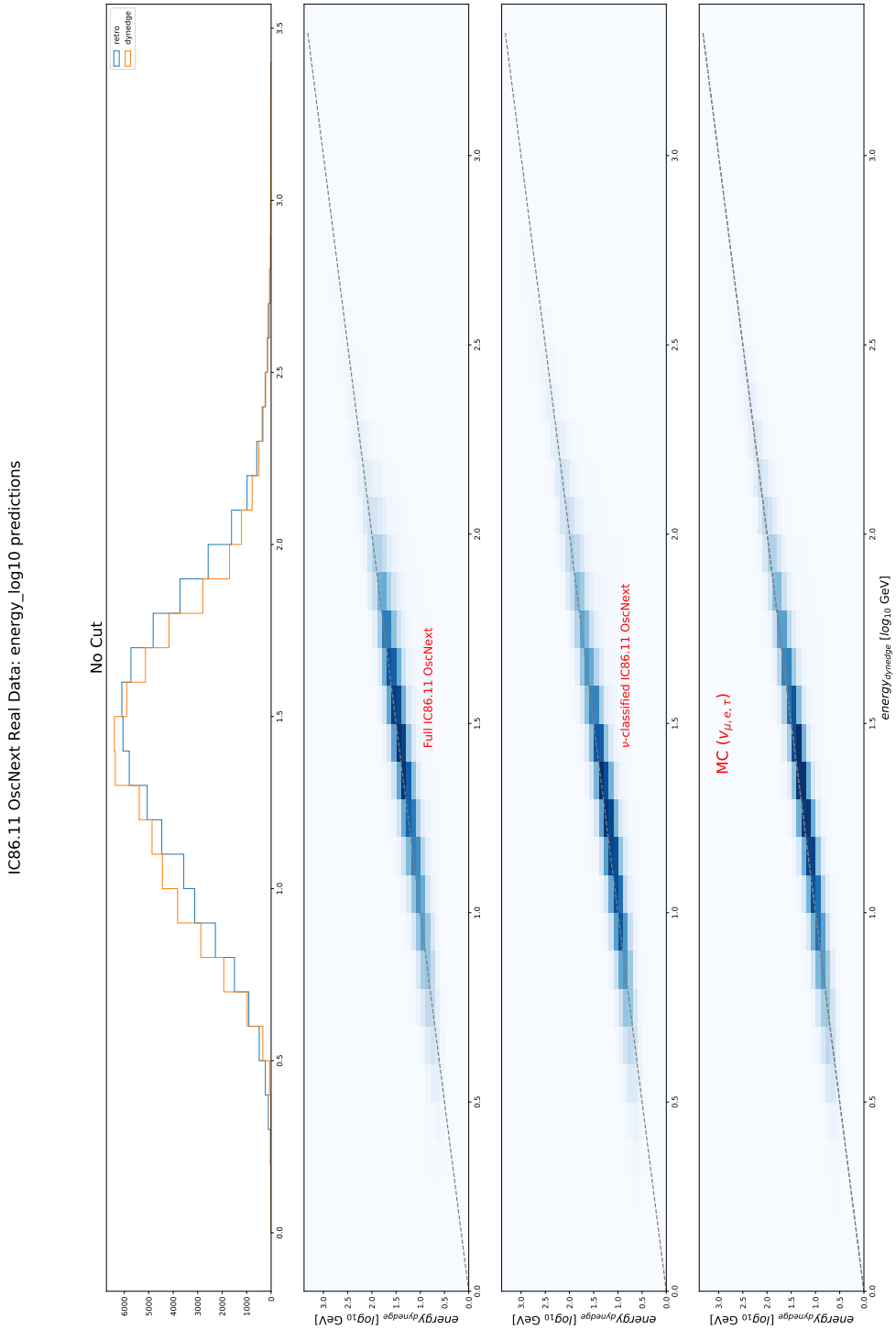
As visible from Figure 10.2, the distribution in ν score is quite similar between MC and real data. This gives good reason to think that comparable results to the quite effective classification observed in MC can be expected in real data, and suggests that a simple transformation of ν score could produce a near-perfect agreement.

Regression Results

This section contains regression results for *azimuth*, *zenith* and *energy_log*10 from *dynedge* and RetroReco for comparison. First row in the figures contain the prediction distributions. Second row contains a 2D-histogram where predictions from *dynedge* and RetroReco in real data are plotted against each other. Third row is the same as the second, with the exception that only predictions on events labeled as neutrinos by the *dynedge* neutrino/muon classifier is shown. 4th row is the same as the second and third, with the exception that the predictions are made on MC data.


IC86.11 OscNext Real Data: zenith predictions
No Cut
50.0 % Most Certain Dynedge
retro
dynedge
Full IC86.11 OscNext
ν-classified dynedge IC86.11 OscNext
MC ($\nu_{\mu,e,\tau}$) CC + NC
$zenith_{dynedge}$ [Deg.]
$zenith_{retro}$ [Deg.]


IC86.11 OscNext Real Data: azimuth predictions
No Cut
50.0 % Most Certain Dynedge
retro
dynedge
Full IC86.11 OscNext
ν-classified dynedge IC86.11 OscNext
MC ($\nu_{\mu,e,\tau}$) CC + NC
$azimuth_{dynedge}$ [Deg.]
$azimuth_{retro}$ [Deg.]


IC86.11 OscNext Real Data: energy_log10 predictions
No Cut
retro
dynedge
Full IC86.11 OscNext
ν-classified IC86.11 OscNext
MC ($\nu_{\mu,e,\tau}$)
energy$_{dynedge}$ [log$_{10}$ GeV]
energy$_{dynedge}$ [log$_{10}$ GeV]

Chapter 11

Conclusion & Outlook

Conclusions

The initial question that was shared in one of the very first pages of this work was

Can a GNN be used to reconstruct low-energy neutrino events in IceCube, and if so, how does it compare to RetroReco?

It is clear from this work that GNN's can be used to reconstruct low energy events in IceCube. In comparison to RetroReco, it is seen in Chapter 8 that *dynedge* offers an increase in performance in the metrics considered in this work in the oscillation relevant energy range of 11.7%, 22.4% and 16.3% for regression targets *azimuth*, *energy_log*10 and *zenith*, respectively. From the oscillation comparison in Figure 8.15 it's seen that the increased performance in *zenith* leads to visible improvements in the MC oscillation plot. Also, it is seen in Figure 8.11, that a quite significant boost in event classification can be expected at level 7 if the current final oscNext classifier is replaced with *dynedge*. Effectively one can either increase the neutrino count with approx. 15% or keep the current neutrino count but with one fifth the false positives (In MC, at least). The pre-fit results for classification shown in Figure 10.2 suggests that the classification behavior in MC and in real data is similar and a quite simple (nearly linear) fit would make the distributions to match. In addition, this work records reconstruction speeds at 15.000 events pr. second as compared to 5 - 40 seconds pr. event for RetroReco, which opens the possibility of cosmic alerts from low energy events. While the regressions for *azimuth* and *zenith* offers uncertainties that from Figures 8.12 and A.16 seems superior to those produced by this version of RetroReco, the kink in the 50% most certain *zenith* predictions visible in Figure 8.5 and indirectly visible in Figure 8.10 suggests that further work in post-processing of these uncertainties could lead to further improvements in cases where cuts in uncertainties are considered.

Outlook

If time had permitted, I would've liked to have addressed the following areas:

1): Exploring (and fixing) the cause that makes error estimations poor at approx. 140 Deg. in zenith angle. This effect was first recorded very late in the process as it first showed when sample size was increased from 2 million to the 8.1 million used to produce the final results. Superficial examinations of this have showed that the kink seems to originate from events where $\frac{\sigma_z}{pull} > 1$.

2): Revisit the 'realistic' mc-data composition. Here especially, use a more rigid method of obtaining the rates for noise, neutrinos and muons. Also; experiment with using CORSIKA as the main muon generator. At level7, it was explained to me (and this is perhaps evident from the classificaiton score plot) that muongun is an OK muon generator. However, at lower cleaning levels this might not be the case.

3): Check if the FO filter method can strengthen the correlation between dom count and energy in non-upgrade data.

4): Examine other methods of including / dealing with multiple DOM activations during an event in non-upgrade data. (Currently the DOM is just included multiple times, but this might impact the KNN calculation negatively)

5): Explore further a 'pulse-cleaning' pooling layer that allows the model to drop nodes it doesn't like. If this works robustly, the model *could* be applied already at trigger-level. Warp speed!

6): Apply the model in high-energy regimes and use the moon-analysis to estimate performance. If it works well there, which it *should*, one could train and apply the model to the entire energy range.

7): Try to do an oscillation analysis in real data. This is probably the ultimate test!

List of Figures

1.1 **(A)**: **(B)**: . . . 3
1.2 Fundamental neutrino verticies . . . 5
1.3 An Electron . . . 6

5.2 **(A)**: An illustration of a CNN architecture used in image recognition of handwriting data. The handwritten letter 5 is given as argument to a convolutional operator that produces a feature-extracted version of that image (second column), which is then passed to an MLP that then produces a categorization of in the input but based on the feature extracted version of the input. **(B)**: An illustration of a convolution operation. Kindly borrowed from [53]. . . . 35

6.4 **(A)**: Comparison of results on segmentation learning between the proposed model in [59] and an older point cloud model **PointNet**[60]. Kindly borrowed from [59]. **(B)**: A plot of active DOM positions in a muon-neutrino event. Colorbar depicts time variable dom_time. . . . 42

8.1 . . . 68
8.2 . . . 69
8.3 . . . 70
8.4 . . . 71
8.5 . . . 72
8.6 . . . 73
8.7 . . . 74
8.8 . . . 75
8.9 . . . 76
8.10 Relative improvement plot for error width in regressions of *azimuth*, *zenith*, *energy_log*10 for both cut and pre-cut regressions. 50% most certain is from uncertainties produced by *dynedge*. . . . 77
8.11 ROC Curve for the unbiased $\nu - \mu$ classification model. **"lvl7_probnu"** represents the oscNext level 7 muon/neutrino classifier.**dynedge MPS** corresponds to the PID selection where the event PID is concluded on the basis of the maximal estimated PID from *dynedge*. . . . 78
8.12 Pull plot for *zenith* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model. . . . 79
8.13 Running pull plot for *zenith* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model. Notice that only data satisfying $\sigma < 5$ has been displayed in this figure for both RetroReco and *dynedge*. . . . 80
8.14 Performance plot showing PID dependence in regression performance for *zenith*. $(\nu_e, \nu_\mu, \nu_\tau)\,|_{\nu_\alpha}$ denotes a model trained on all neutrino types but evaluated only in α. Sample size in legend. Data contains both CC and NC events, at ratios of approx. 90% and 10%, respectively. . . . 81
8.15 . . . 82

A.1 Energy distribution for the main dataset . . . 103
A.2 Zenith distribution for the main dataset . . . 103
A.3 Azimuth distribution for the main dataset . . . 103
A.4 Position x distribution for the main dataset . . . 104
A.5 Position y distribution for the main dataset . . . 104
A.6 position z distribution for the main dataset . . . 104
A.7 Energy distribution for the prototype dataset . . . 105
A.8 Zenith distribution for the prototype dataset . . . 105
A.9 Azimuth distribution for the prototype dataset . . . 105
A.10 Position x distribution for the prototype dataset . . . 106
A.11 Position y distribution for the prototype dataset . . . 106
A.12 position z distribution for the prototype dataset . . . 107
A.13 . . . 107
A.14 . . . 108
A.15 Pull plot for *azimuth* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model. . . . 108
A.16 Running pull plot for *azimuth* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model. Only data satisfying $\sigma < 5$ for both RetroReco and *dynedge* is displayed in this figure. . . . 109
A.17 Performance plot for *energy_log*10 regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model as a function of number of pulses in each neutrino event. . . . 110
A.18 Performance plot for *zenith* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model as a function of number of pulses in each neutrino event. . . . 111
A.19 Performance plot for *azimuth* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model as a function of number of pulses in each neutrino event. . . . 112

List of Tables

3.1 Table containing the most relevant input fields from .i3 event data files. These features are simulated values of IceCube detector read-outs. 20
3.2 Table containing the most relevant truth variables for an event for both the current detector and the upgrade. The **muon_tracklength** variable only exists in the cases where the particle is a muon. 21

9.1 Average pulses (active DOMS) in each event on 'the good side' of the cut by cleaning method. 'No SRT' indicates no pulse cleaning, SRT + FO Filter represents the method developed and examined in this work. 88

10.1 Rates at level 7 oscNext filtering obtained by manually reading 3.1 . . 89

Appendix A

Prototyping Dataset Distributions

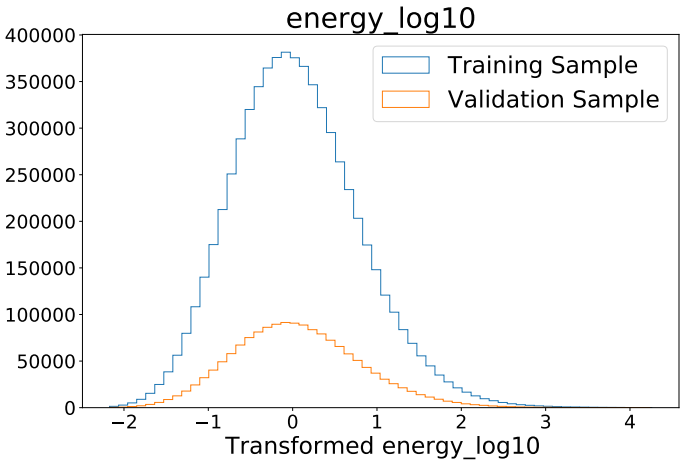


FIGURE A.1: Energy distribution for the main dataset

FIGURE A.2: Zenith distribution for the main dataset

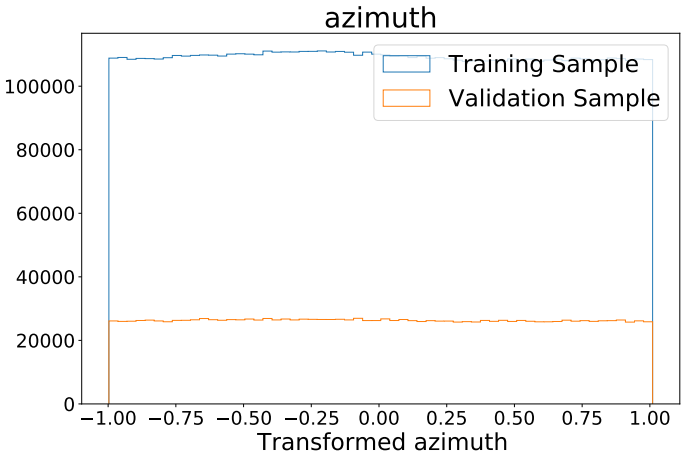


FIGURE A.3: Azimuth distribution for the main dataset

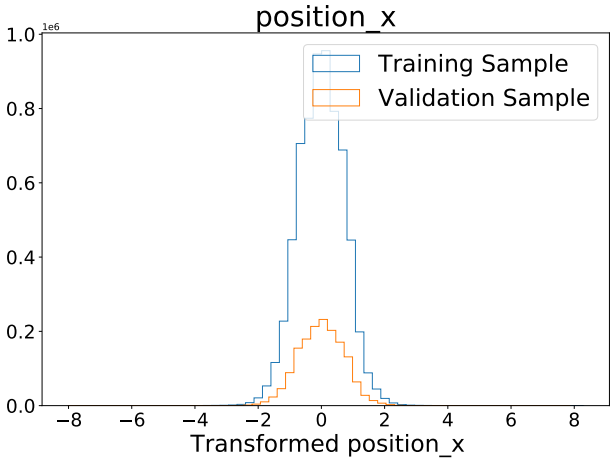


FIGURE A.4: Position x distribution for the main dataset

FIGURE A.5: Position y distribution for the main dataset

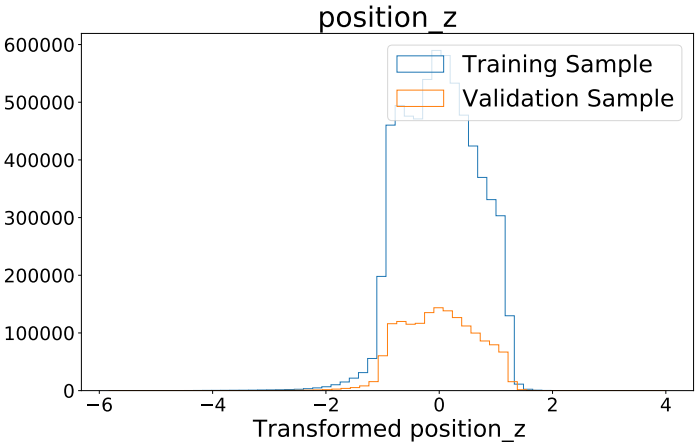


FIGURE A.6: position z distribution for the main dataset

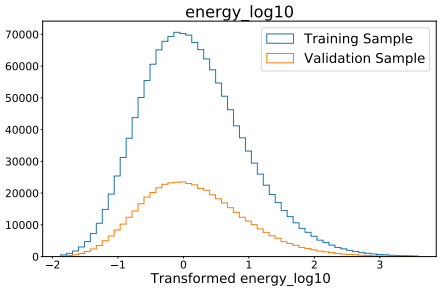


FIGURE A.7: Energy distribution for the prototype dataset

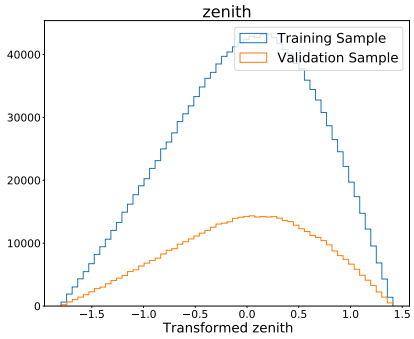


FIGURE A.8: Zenith distribution for the prototype dataset

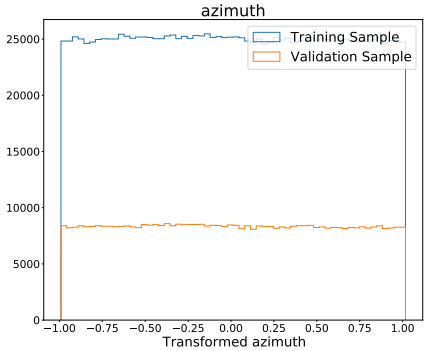


FIGURE A.9: Azimuth distribution for the prototype dataset

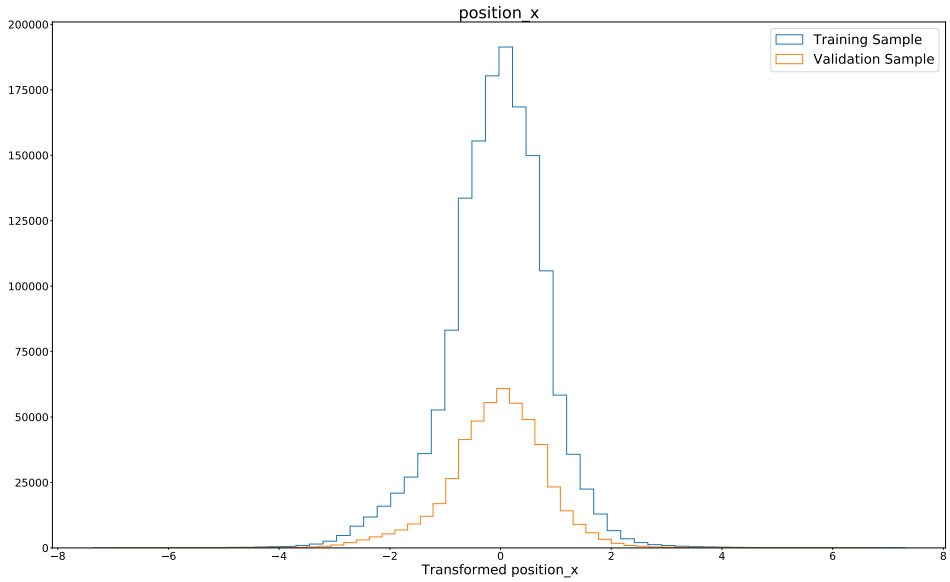


FIGURE A.10: Position x distribution for the prototype dataset

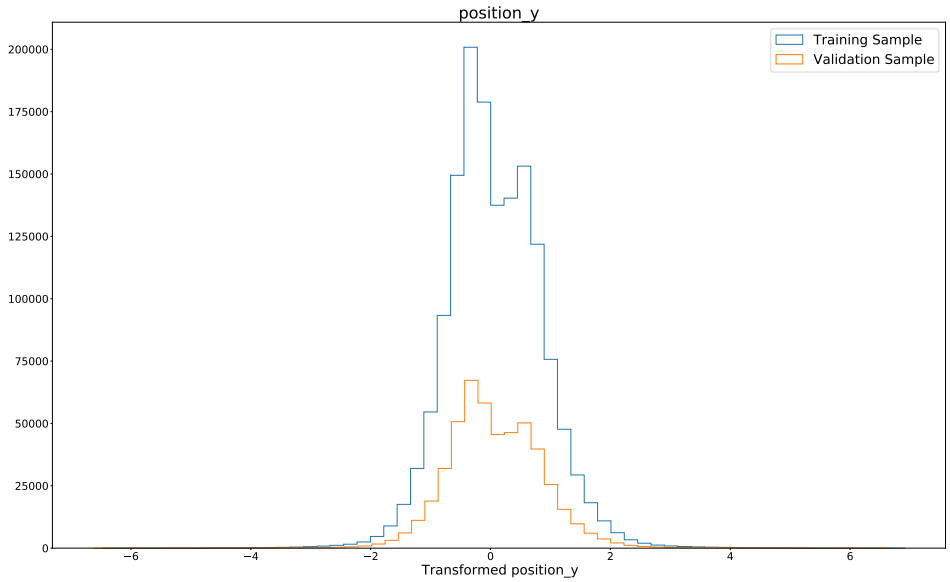


FIGURE A.11: Position y distribution for the prototype dataset

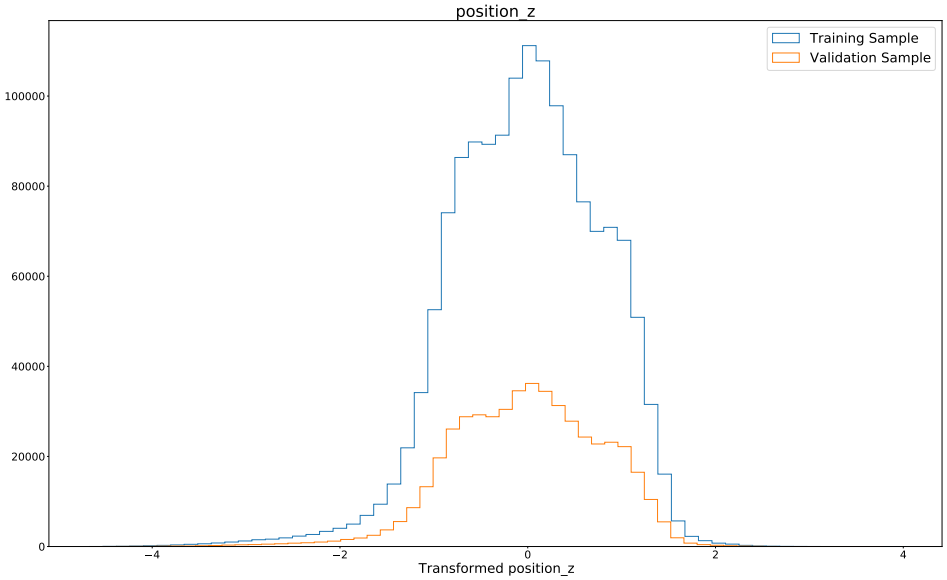


FIGURE A.12: position z distribution for the prototype dataset

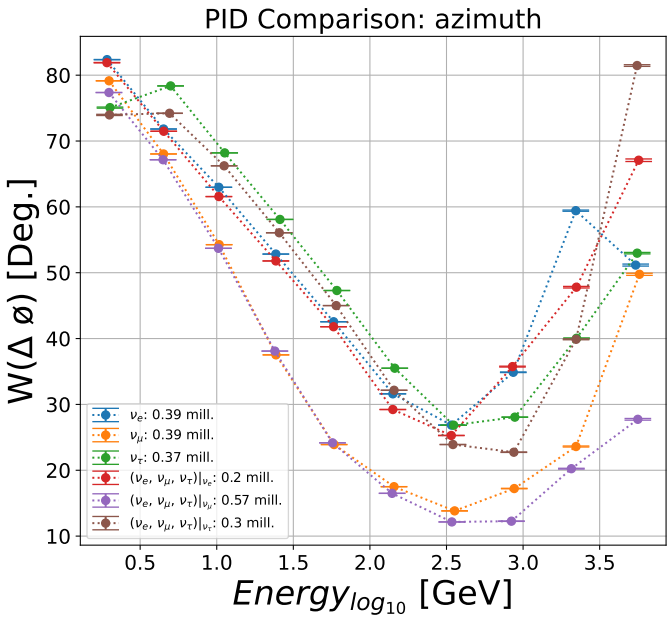


FIGURE A.13

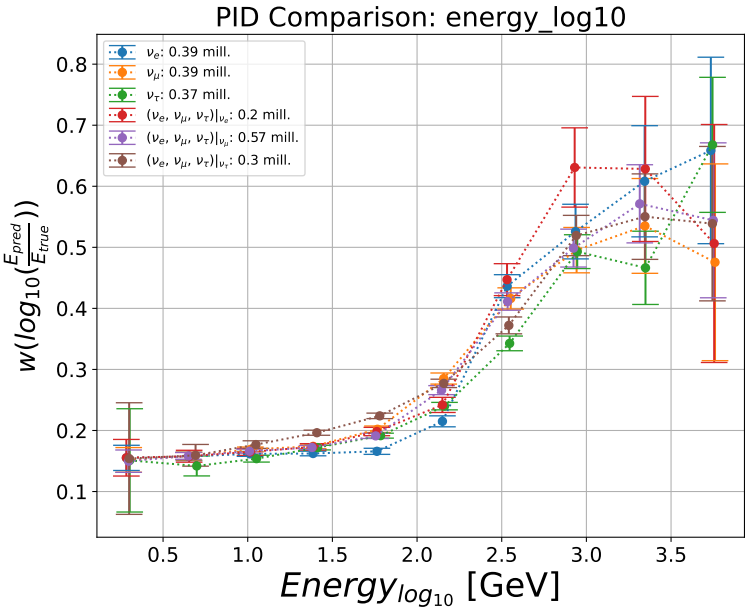


FIGURE A.14

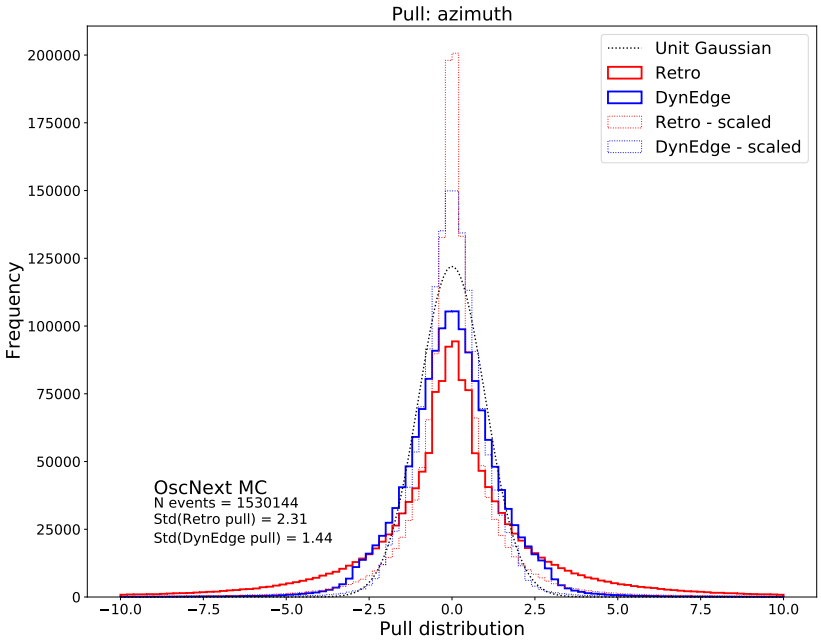


FIGURE A.15: Pull plot for *azimuth* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model.

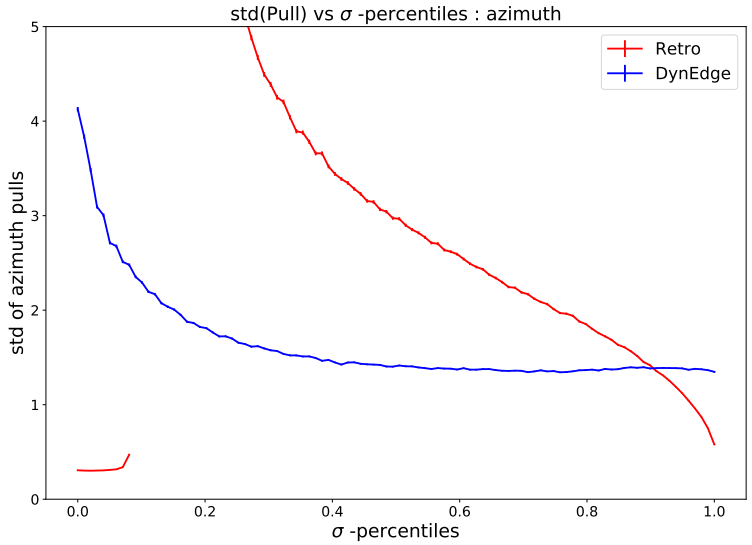


FIGURE A.16: Running pull plot for *azimuth* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model. Only data satisfying $\sigma < 5$ for both RetroReco and *dynedge* is displayed in this figure.

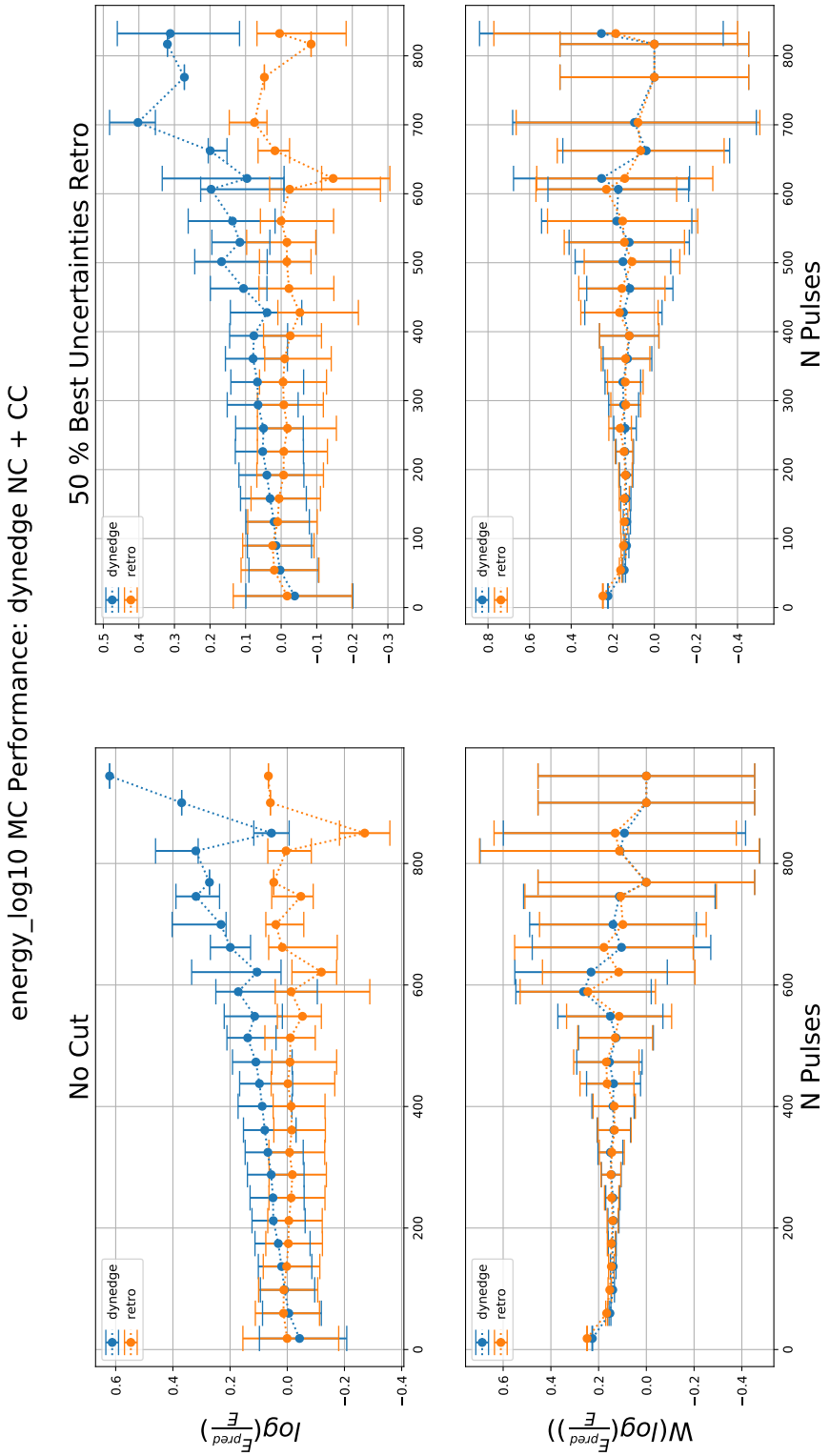


FIGURE A.17: Performance plot for *energy_log*10 regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model as a function of number of pulses in each neutrino event.

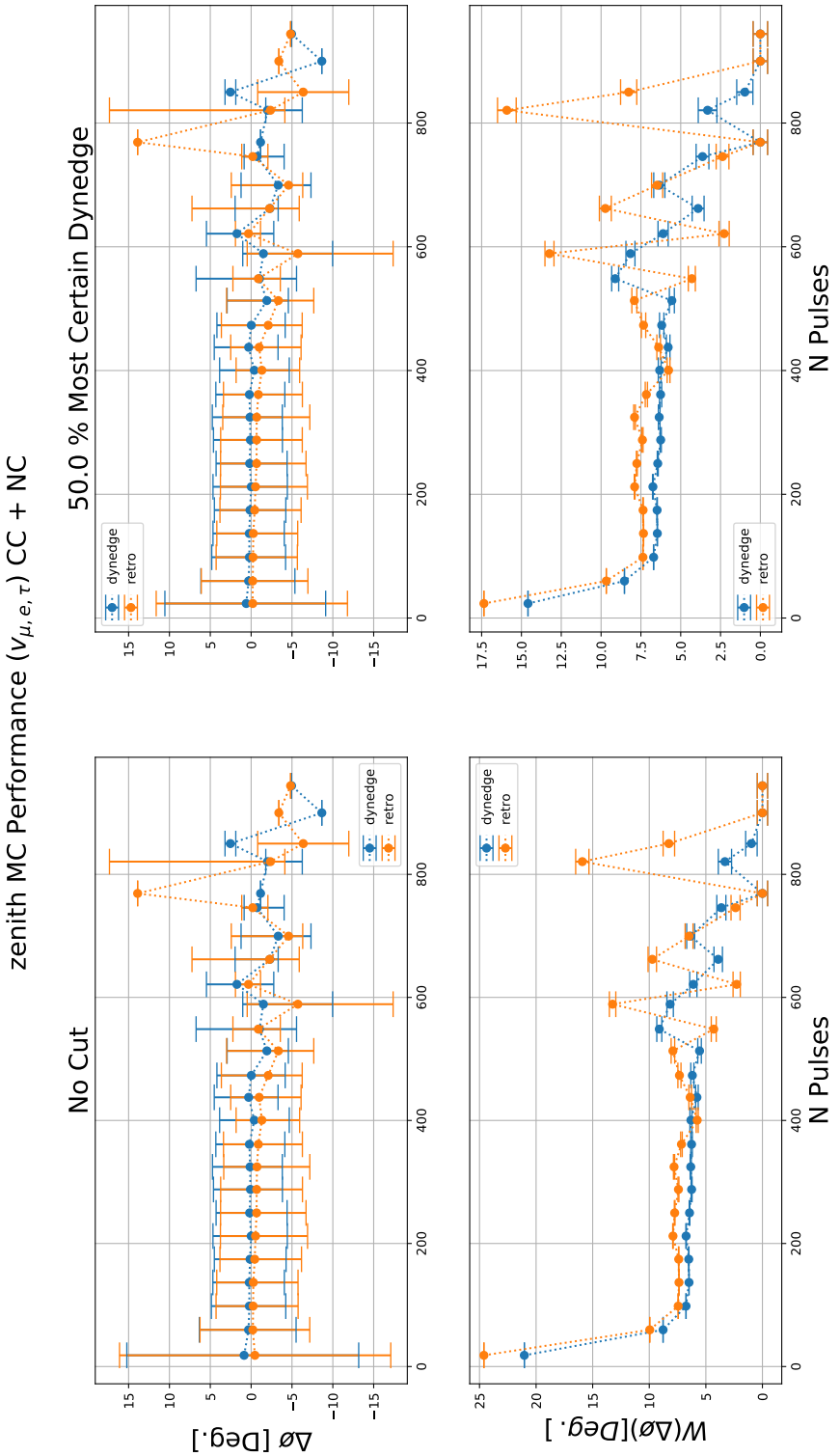


FIGURE A.18: Performance plot for *zenith* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model as a function of number of pulses in each neutrino event.

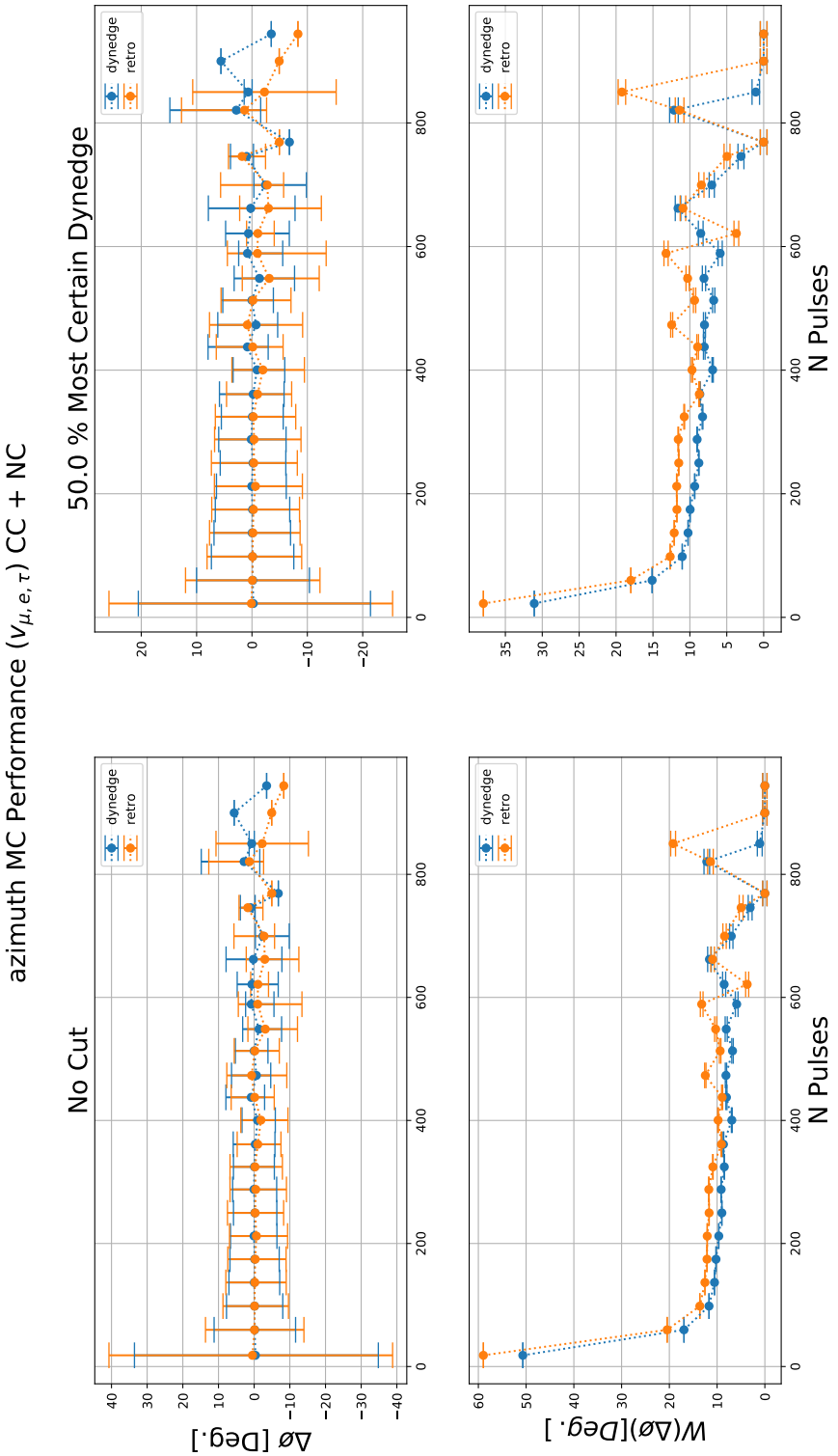


FIGURE A.19: Performance plot for *azimuth* regression from the $(\nu_e, \nu_\mu, \nu_\tau)$ -model as a function of number of pulses in each neutrino event.

Appendix B

Quick guide: NBI HEP Cluster

Below is a collection of good-to-know information for NBI students wishing to use the NBI Cluster. Most high-performance computational clusters are set up as Linux servers and this is the case for the NBI HPC Cluster too, for this reason this little guide contains basic examples of some of the most relevant Linux commands.

Login

Make sure that you have received login credentials. You can obtain login credentials by talking to your thesis supervisor. **It is a good idea to put in your home ip address in the application**, otherwise you will only be able to access HEP when you're at NBI. If you forgot to do that, you can mail them and beg for forgiveness. It is also a good idea to setup two-factor authentication - this gives you the ability to temporarily white-list an outside ip. When you have your credentials, open up your favorite terminal. I've been using **anaconda prompt** from the Anaconda python distribution. You can log in via the **secure shell** command:

```
ssh username@server.domain
```

You'll then be asked to write your password. If you login for the first time you'll be prompted with

```
The authenticity of the host 'server.domain' can't be established.
CDSA key fingerprint is {some long hash}
Are you sure you want to continue connecting (yes/no)?
```

Type yes and hit enter - now you're in! Here's a list of servers I used during my work at NBI:

- hep01.hpc.ku.dk
- hep02.hpc.ku.dk
- hep03.hpc.ku.dk
- hep04.hpc.ku.dk

hep01 - hep03 is **cpu only**, whereas hep04 contains two **RTX3090** GPU's.

example

```
ssh username@hep02.hpc.ku.dk
```

notes

It is possible to setup your ssh in such a way that you do not have to write your password every time you attempt to login. There's plenty of guides on how to do this online.

Installing Python

Using an anaconda distribution is the easiest way to install python of a higher version that what hep comes with. Installation guide is available on anaconda's own webpage. I've placed an anaconda installer on HEP which you can use by running the command

```
./ /groups/hep/pcs557/tmp/Anaconda3-2020.11-Linux-x86_64.sh
```

'Mounting'

You can access your directory on hep by 'mounting' your home directory on hep directly as a disk image on your pc. Doing so will make it appear as a new hard drive. Doing so is easiest on Linux machines. It can be done on windows using a tool named 'Putty'. Plenty of guides available online.

Using Modern Editors Directly on hep

If you mount your hep directory to your machine, you'll be able to edit scripts using your favorite editors. If you do not mount hep (which I didn't), you can access your scripts on hep using ssh via the editor **Visual Studio Code**. A guide to this is available here.

Good Practice

It is custom to check for available resources before you run your jobs on any of the machines, and to have a reasonable sense on how demanding your code is. If your code eats up all available RAM, or clogs up the CPU, it could impact the jobs that other people are currently running. You can check if CPU and RAM usage via

```
top
```

which shows a table containing current jobs, their CPU usage and RAM usage. Interpreting the table from **top** can be a little difficult so please ask if in doubt. You can get a better gauge on available RAM using

```
free -h
```

You can check GPU resources by installing the python package **gpustat** which is available via pip. Once installed, you can run

```
gpustat --watch
```

to get a top-like status of utilization and available VRAM.

For reference: All hep servers have at least 128gb of RAM and more than 24 cores.

Useful Commands

Changing Directory

You can **change your current directory** using the

```
cd
```

command. The general file structure on hep´is

```
'\groups\group-name\username\your_home_directory'
```

So if Einstein were a member of hep, and had a folder named 'thisisrelative' in his home directory, you could check it out via

```
cd '\groups\hep\einstein\thisisrelative'
```

notes

When you're accessing your own home directory, you don't have to write out the full path every time, as the following paths

```
'\groups\hep\yourusername\yourcoolfolder'
'~\yourcoolfolder'
```

are equivalent.

Copying files

You can copy files from your PC to hep via **scp**. The general syntax is

```
scp copy_this place_it_here
```

Suppose you had a file named 'results.txt' located in your C drive and you wanted to copy it onto your home directory in hep in a folder named results. You could do so by

```
cd C:\
scp  results.txt yourusername@hep02.hpc.ku.dk:~/results/results.txt
```

If you want to copy a whole folder from your PC you need to add **-r** :

```
cd C:\
scp -r  folder_on_c-drive yourusername@hep02.hpc.ku.dk:~/results/folder_on_c-drive
```

If you wanted to copy files the other way around, e.g. from hep to your PC, all you have to do is to reverse the arguments:

```
scp -r  yourusername@hep02.hpc.ku.dk:~/results path_to_where_you_want_it_on_your_pc
```

notes

The disk space is shared between hep servers. So if you copy something to a location on hep02, it will be available on hep04. As an alternative, you can install an sftp program like WinSCP - this reduces copying to-and-from hep to a drag-and-drop operation.

List Content of Directory

You can print a list of contents of a folder by entering

```
ll path_to_folder
```

if you do not specify a path, the command prints the contents of the folder you're currently in.

Creating a Folder

You can create a folder by

```
mkdir path_to_folder\foldername
```

if you do not specify a path, the folder is created in the folder you're currently in.

Support

The HPC HEP cluster at NBI is maintained by HPC UCPH. Their official website is here. You can contact them via **support@hpc.ku.dk** . Two Factor Authentication guide is available here.

Appendix C

Notes on i3, CVMFS and Shovel

Activating a CVMFS

CVMFS (CernVM-File System) is a virtual file system used at IceCube to distribute local, virtual environments containing IceCube software tools for development. Using CVMFS at NBI **does not** require an official IceCube account and doesn't come with a complicated installation process. It is therefore an ideal solution to get hold of IceTray, an IceCube software package used to read, write and process i3 files.

When you're signed into hep, you can activate a CVMFS by typing

```
eval /cvmfs/icecube.opensciencegrid.org/py3-v4/setup.sh
/cvmfs/icecube.opensciencegrid.org/users/Oscillation/
software/oscNext_meta/releases/latest/build/env-shell.sh
```

Notice I had to break up the last line to make it fit here. The first line selects the python version - this particular CVMFS runs Python 3.6.5. You can choose different versions by replacing the first line. To see a list of supported versions, simply

```
cd /cvmfs/icecube.opensciencegrid.org/
ll
```

You can install any libraries you want within a CVMFS without it interfering with your other installations outside the CVMFS. To install packages in the CVMFS, simply load the CVMFS and

```
pip --user install mypackage
```

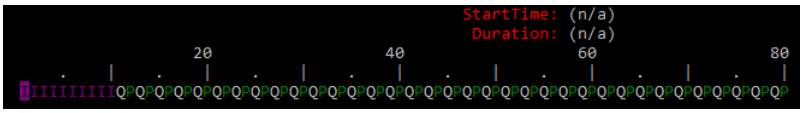
Checking out an i3 file via Shovel

Shovel is an IceCube software tool that can visualize the contents of a specific i3 file graphically. Suppose you wanted to check out this i3 file

```
/groups/icecube/stuttard/data/oscNext/pass2/genie/level7_v02.00/
120000/oscNext_genie_level7_v02.00_pass2.120000.000644.i3.zst
```

Notice I had to break up the path to make it fit here. This particular file is from **Tom Stuttards** personal directory - it's a level7 neutrino sample from the GENIE simulation. To check out the contents visually, simply load your CVMFS of choice and

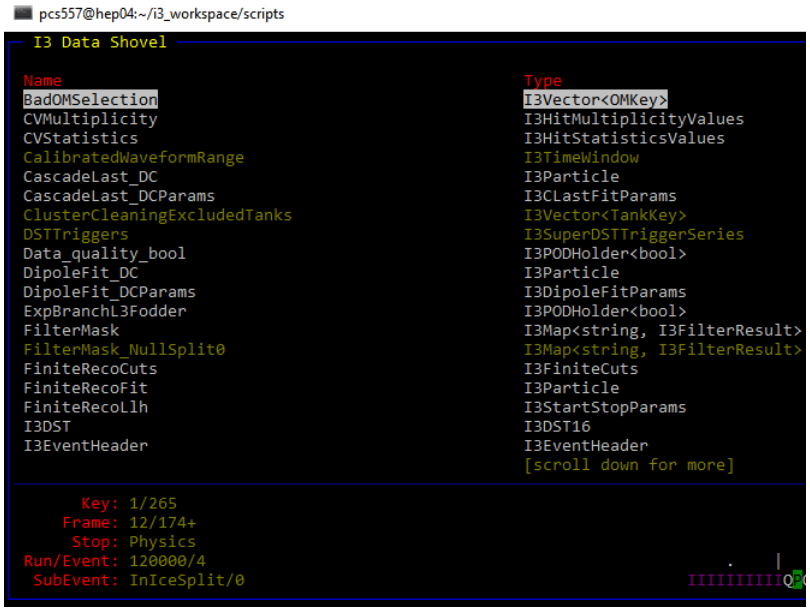
```
dataio-shovel pathtoi3file
```



if you replace **'pathtoi3file'** with the path from above, your terminal will now display the contents of the file. On the bottom right is a sequence of frames, showing which frame of the i3 you're currently browsing.

The I and Q frames are of little interest for physics analysis, as any data relating to the physical properties are stored in P (physics) frames. **In an i3 file, a P-frame represents a single event**. To change frame simply use the arrow keys. Below is a picture of the first P frame of the i3 file.

The name of the i3 keys within the P frame is shown under the **Name** column.



These keys are used to access the underlying data associated with the specific P frame, much like a key in an ordered dictionary. To much frustration, most of the keys in any P frame is not documented at all.

I3 Keys

Below is a list of i3 keys and their meaning used in this work

- $frame['I3MCTree'][0].energy$: The energy of the primary particle
- $frame['I3MCTree'][0].pos.x$: The x coordinate of the interaction vertex of the primary particle
- $frame['I3MCTree'][0].pos.y$: The y coordinate of the interaction vertex of the primary particle
- $frame['I3MCTree'][0].pos.z$: The z coordinate of the interaction vertex of the primary particle
- $frame['I3MCTree'][0].dir.azimuth$: The azimuth angle of the primary particle relative to the detector

- $frame['I3MCTree'][0].dir.zenith$: The zenith angle of the primary particle relative to the detector
- $frame['I3MCTree'][0].pdg\_encoding$: The particle identity of the primary particle as given by the pdg encoding convention in HEP
- $frame['L7\_reconstructed\_azimuth']$: RetroReco Reconstruction of azimuth,
- $frame['L7\_reconstructed\_time']$: RetroReco Reconstruction of interaction time
- $frame['L7\_reconstructed\_total\_energy']$: RetroReco Reconstruction of energy. (Notice total refers to track + cascade energy).
- $frame['L7\_reconstructed\_vertex\_x']$: RetroReco Reconstruction of position_x
- $frame['L7\_reconstructed\_vertex\_y']$: RetroReco Reconstruction of position_y
- $frame['L7\_reconstructed\_vertex\_z']$: RetroReco Reconstruction of position_z
- $frame['L7\_reconstructed\_zenith']$: RetroReco Reconstruction of zenith
- $frame['L7\_retro\_crs\_prefit\_\_azimuth\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for azimuth
- $frame['L7\_retro\_crs\_prefit\_\_x\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for position_x
- $frame['L7\_retro\_crs\_prefit\_\_y\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for position_y
- $frame['L7\_retro\_crs\_prefit\_\_z\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for position_z
- $frame['L7\_retro\_crs\_prefit\_\_time\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for interaction time.
- $frame['L7\_retro\_crs\_prefit\_\_zenith\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for zenith.
- $frame['L7\_retro\_crs\_prefit\_\_energy\_sigma\_tot']$: RetroReco Reconstruction Uncertainty for energy.
- $frame['L7\_MuonClassifier\_FullSky\_ProbNu']$: Final OscNext neutrino classifier.
- $frame['L4\_MuonClassifier\_Data\_ProbNu']]$: First OscNext neutrino classifier.
- $frame['I3GENIEResultDict']['y']$: Elasticity
- $frame[''I3MCWeightDict''][''InteractionType'']$: Interaction type
- $frame['L2\_oscNext\_bool']$: If true passed level2
- $frame['L3\_oscNext\_bool']$: If true passed level3
- $frame['L4\_oscNext\_bool']$: If true passed level4
- $frame['L5\_oscNext\_bool']$: If true passed level5

- $frame['L6\_oscNext\_bool']$: If true passed level6
- $frame['L7\_oscNext\_bool']$: If true passed level7
- $frame['I3MCWeightDict']["weight"]$: Oscillation weight. Needed to produce oscillation bands.

Appendix D

The NBI Intergalactic IceCube Predictions Treaty

What You Need to Provide

Your results shall come in the form of a **pandas dataframe** saved as a *.csv*-file with a column for *event_no* and columns representing the target variables of your model. The column names in the dataframe needs to follow the variable naming convention outlined below.

Variable Naming Convention

The following is the naming convention for truth variables indigenous to the i3-files.

- *event_no* : The integer event identifier as provided by the database pipeline.
- *energy_log*10 : logarithm in base 10 of the energy of the primary particle as extracted from $frame['I3MCTree'][0].energy$
- *position_x* : The x coordinate of the interaction vertex of the primary particle as extracted from $frame['I3MCTree'][0].pos.x$
- *position_y* : The y coordinate of the interaction vertex of the primary particle as extracted from $frame['I3MCTree'][0].pos.y$
- *position_z* : The z coordinate of the interaction vertex of the primary particle as extracted from $frame['I3MCTree'][0].pos.z$
- *azimuth* : The azimuth angle of the primary particle relative to the detector as extracted from $frame['I3MCTree'][0].dir.azimuth$
- *zenith* : The zenith angle of the primary particle relative to the detector as extracted from $frame['I3MCTree'][0].dir.zenith$
- *pid* : The particle identity of the primary particle as given by the pdg encoding convention in HEP as extracted from $frame['I3MCTree'][0].pdg\_encoding$

In order to not confuse true values with predicted values, the names of your columns in the dataframe containing predictions needs have an additional **_pred**. Uncertainties should have an additional **_sigma**. **The results should have the same units and scaling as the truth variables in the database you're predicting on**.

Example

If your model regresses only *energy_log*10 you would provide a pandas dataframe in .csv format containing a column *event_no* representing the events chosen for regression and a column *energy_log*10*_pred* representing the corresponding regression of *energy_log*10. Uncertainties of this regression would be contained in *energy_log*10*_sigma*. The rows must be ordered such that i'th value in *energy_log*10*_pred* represents the regression of the energy of the i'th event in *event_no* and similarly for the uncertainties. If the energy regression model doesn't produce uncertainties, simply omit the *_sigma* column.

Submissions via *submit_results.py*

Start by making sure that your .csv file containing your results are located *somewhere* on hep. You'll need a modern version of python and the pip distributed package **sqlalchemy**. If you're unsure about any of this, please review appendix B. **A modern installation of Anaconda comes with all the packages required.**

Now, with your data on hep, all you need to do to make a submission is to call

```
python submit_results.py --database 'db' --path 'path' --model 'model' --init 'init'
```

where the arguments represents

- **database** : The name, not path, to the database containing the events which you have results for. For a list of available databases, please see the next section.
- **path** : The path to the .csv file on hep containing your results in the correct format as outlined above.
- **model** : The name of your model. Please keep it brief.
- **init** : Your initials. Please be consistent.

submit_results.py then produces a dedicated table for your predictions in the prediction database associated with the chosen database. The name of the table will be on the form **'init'_'model'**, eg. **RFO_dynedgev3_energy_log10**.

Additional Functionality

To get an overview of the databases accepting predictions currently, simply call:

```
python submit_results.py --list_databases True
```

This will print a list of databases that accepts predictions, e.g. a list of valid arguments for **db**.

An Example of Extraction

Suppose you wanted to extract predictions made on **dev_level7_oscNext_IC86_003**. By construction, the prediction database would be located at

/groups/hep/pcs557/databases/dev_level7_oscNext_IC86_003/predictions/predictions.db

You could then obtain a list of all submitted predictions by simply printing the submitted tables as:

```
import sqlite3
mc_db = 'path_to_prediction_database'

with sqlite3.connect(mc_db) as conn:
    cursorObj = conn.cursor()
    print('Submitted Predictions: ')
    cursorObj.execute("SELECT name FROM sqlite_master WHERE type='table';")
    print(cursorObj.fetchall())
```

This will print a list of submitted predictions, each on the form **'init' - 'model'**, eg. **RFO - dynedgev3_energy_log10**. Suppose you were very interested in the energy regression made by RFO. You could then extract these as a pandas dataframe on the same format as the input you're supposed to deliver by

```
import sqlite3
import pandas as pd
mc_db = 'path_to_prediction_database'

with sqlite3.connect(mc_db) as conn:
    query = 'select * from RFO_dynedgev3_energy_log10'
    predictions = pd.read_sql(query,con)
```

Easy!

Appendix E

SQLite Databases

In this appendix is a brief list and explanation of the databases produced during this work, in case you find yourself at NBI wondering what's in them. The databases are located in:

```
'/groups/hep/pcs557/databases'
```

Please notice that *event_no* is unique within each database only. E.g. event 0 in database A is **NOT** the same as event 0 in database B. All the databases below contains pulses from **SRTSplitInIcePulses**.

dev_lvl7_mu_nu_e_classification_v003

This database contains all events contained in the following directories on cobalt:

```
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/120000"
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/140000"
"/data/ana/LE/oscNext/pass2/genie/level7_v02.00/160000"
"/data/ana/LE/oscNext/pass2/muongun/level7_v02.00/130000"
```

This means that the database contains all neutrino types + muons from muongun, all at level7, which in turn means that reconstructions from RetroReco is available. Everything in this database is MC. Please note that this database does not contain pure noise events.

IC8611_oscNext_003_final

This database contains all events contained in the following directory on cobalt

```
"/data/ana/LE/oscNext/pass2/data/level7/IC86.11"
```

This is the entire IC86.11 **real measurements** sample at level7. Approx. 64000 events with RetroReco reconstruction. The features in this database have been scaled using the feature scalers fitted to *dev_lvl7_mu_nu_e_classification_v*003. It is therefore intended that one should train on *dev_lvl7_mu_nu_e_classification_v*003 in order to predict on this dataset. If you prefer something else, it is advised to rescale this database using whatever standardization your training set has undergone.

dev_level2_mu_tau_e_muongun_classification_wnoise

This database contains all events contained in the following directories on cobalt:

```
"/data/ana/LE/oscNext/pass2/genie/level2_v02.00/120000"
"/data/ana/LE/oscNext/pass2/genie/level2_v02.00/140000"
```

```
"/data/ana/LE/oscNext/pass2/genie/level2_v02.00/160000"
"/data/ana/LE/oscNext/pass2/muongun/level2_v02.00/130000"
"/data/ana/LE/oscNext/pass2/noise/level2_v02.00"
```

This means that the database contains all neutrino types, muongun muons and pure noise events at level2. Please notice that **the noise events have fictional truth variables**. You can select / deselect noise events by querying after $sim\_type = ' noise'$ or $abs(pid) = 1$. Exclusively to this database, a boolean variable in the truth table called *passed_lvl*2 is added. If this boolean is true, then the specific event passed the level 2 filtering, qualifying it for level3. So in essence, this database contains both level2 and level3 events. Everything in this database is MC.

Appendix F

Making Databases

This is intended as documentation for those who wish to carry on the torch with SQlite database production at NBI for the IceCube ML group. The code is available here.

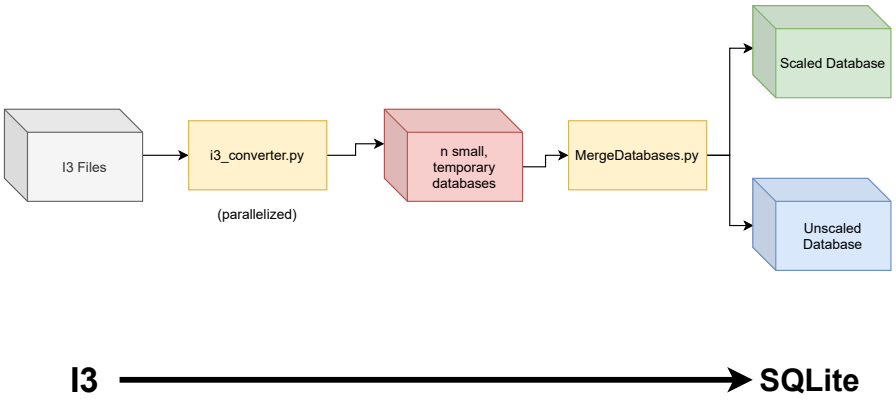


FIGURE F.1: Illustration of the I3-SQLite pipeline used in this work.

The pipeline is illustrated in F.1. The entire process is broken into three steps:

- load_cvmfs.sh
- i3_converter.py
- MergeDatabases.py

load_cvmfs.sh

In order to access the software tools to open the specially designed file formats in IceCube, a software package called IceTray is required. Getting access to this software is usually reserved to people inside IceCube who've got the credentials to download and install the environment (you know - real physicists). However, lucky for us, a much simpler alternative exists - this is the cvmfs - where no logins are required and virtually no setup needed. Therefore, the pipeline begins by loading this distributed environment such that we have the software required to open the files. **Protip:** You can install packages to the cvmfs by using:

```
pip --user install mypackage
```

i3_converter.py

i3_converter.py reads a series of directories containing i3-files and converts all these into temporary databases using multiprocessing. This script requires the cvmfs to be active. The script is called as

```
python i3_converter.py --outdir outdir --mode mode --workers n_workers
```

where **outdir** is the outdir for the temporary databases, **workers** is the number of cores used and **mode** is the read mode which can be either: **data**, **data-retro**, **mc**, **mc-retro**. Here '-retro' simply implies that reconstructions and uncertainties from RetroReco are included. Within i3_converter.py an list-object named paths exists. One must add the paths to the directories (not the files themselves!) where the i3-files you wish to convert into databases are, like so:

```
paths = []
lvl3_vmu = '~/i3_workspace/data/oscNext/genie/level3/level3_v02.00/140000'
paths.append(lvl3_vmu)
```

if you have multiple paths, simply append those as well. The script will find all i3-files and gcd-files in those directories and build temporary databases from them. If a gcd-file exists in the directory, this gcd-file will be used. Otherwise, a fail-safe gcd will be loaded and used instead.

MergeDatabases.py

This script merges all the temporary databases into one big unscaled database. Once the unscaled datbase is built, the script then fits a **sklearn.preprocessing.RobustScaler** to each variable chosen for scaling. (You can adjust which variable that gets transformed by appending the variable name to the list in the script called 'no_scale'. This script requires you to exit the cvmfs. The script is called as:

```
python MergeDatabases.py --outdir outdir --mode mode --path path --db_name db_name
```

where **outdir** is the outdir of the scaled and unscaled database, **path** is the path to the temporary databases produced by i3_converter.py, **db_name** is the name of the database, and **mode** is the mode from which the temporary databases was made, e.g. 'data', 'data-mc', 'mc' or 'mc-retro'.

FAQ

I want the pipeline to extract a feature or a truth that it currently doesn't

This requires changes to both the extracting script and the merging script. First, go to i3_converter.py and make sure the truth or feature variable is extracted in the function extract_fit_vector() and passed back to WriteDicts(). Add the feature or truth variable to either the features- or truths-dictionary by assigning it a field. Remember the name of this field. Now, go to MergeDatabases.py and add this as a column in the parts of the code where the empty scaled and the empty unscaled databases are initialized. (One has to define the name of the columns for the table on creation). Now you're done!

I want the pipeline to extract pulses from a different key than SRTInIcePulses

You can specify the name of the key in WriteDicts() where extract_fit_vector is called.

The pipeline crashes after the unscaled database has been filled

Right after the unscaled database has been created, the process of fitting scalers to it begins. This is memory intensive and if the unscaled database contains more than 10 million events, it's likely that the servers in HEP will run out of memory as it's fitting the scalers. This is because the fitting itself requires us to load the entire feature into memory - so for a 10 million event database, this would be roughly 10^8 rows of data. You can get around this by either: **a)** Make a smaller database. **b)** Use an already fitted scaler instead.

Why is this not a one-step process?

The cvmfs comes with this annoying version of pickle that gives errors when MergeDatabases.py tries to save the dictionary with transformers as a pickle file, so one needs to exit cvmfs in order to do this. One could probably fix this, either directly or by wrapping the process into a shell script. So what are you waiting for?

Why do we need to specify 'mode'?

Maybe we dont. This was chosen because this pipeline has been developed side-by-side with changing needs in terms of data procurement. Since the pipeline needed to be flexible, e.g. to extract different things, an if-approach was chosen. In essence, the mode simply lets the script know what it can expect there to be in the i3-files. It's possible that one could write a smarter piece of code that would simply *check* which of a series of desired variables is available, and extract only those. What are you waiting for?

When I use my own script to make databases, they are quite slow to query from. Why are yours faster?

The speed of the databases comes from a crucial step in their definition. You *should* assign a primary key in the truth table (I choose *event_no*) and an indexation for the features table (I choose *event_no*). To see how this is done, have a look at MergeDatabases.py, or look it up!

Appendix G

Geometric Model

```
def __init__(self, k):
    super(Net, self).__init__()
    c = 3
    l1, l2, l3, l4, l5,l6,l7 = 8,c*16*2,c*32*2,c*42*2,c*32*2,c*16*2,3
    self.k = k

    self.nn_conv1 = torch.nn.Sequential(torch.nn.Linear(l1*2,l2),
    torch.nn.LeakyReLU(),torch.nn.Linear(l2,l3),
    torch.nn.LeakyReLU()).to(device)
    self.nn_conv2 = torch.nn.Sequential(torch.nn.Linear(l3*2,l4)
    ,torch.nn.LeakyReLU(),torch.nn.Linear(l4,l3),
    torch.nn.LeakyReLU()).to(device)
    self.conv_add2 = EdgeConv(self.nn_conv2,aggr = 'add')
    self.nn_conv3 = torch.nn.Sequential(torch.nn.Linear(l3*2,l4),
    torch.nn.LeakyReLU(),torch.nn.Linear(l4,l3),
    torch.nn.LeakyReLU()).to(device)
    self.conv_add3 = EdgeConv(self.nn_conv3,aggr = 'add')
    self.nn_conv4 = torch.nn.Sequential(torch.nn.Linear(l3*2,l4),
    torch.nn.LeakyReLU(),torch.nn.Linear(l4,l3),
    torch.nn.LeakyReLU()).to(device)

    self.conv_add4 = EdgeConv(self.nn_conv4,aggr = 'add')

    self.nn1 = torch.nn.Linear(l3*4 + l1,l4)
    self.nn2   = torch.nn.Linear(l4,l5)
    self.nn3 =  torch.nn.Linear(4*l5,l6)
    self.nn4 = torch.nn.Linear(l6,l7)
    self.relu = torch.nn.LeakyReLU()
    self.tanh = torch.nn.Tanh()


def forward(self, data):
    k = self.k
    x, edge_index, batch = data.x, data.edge_index, data.batch
    x,edge_index = KNNAmp(k, x, batch)
    a = self.conv_add(x,edge_index)
    _,edge_index = KNNAmp(k, a, batch)
    b = self.conv_add2(a,edge_index)
    _,edge_index = KNNAmp(k, b, batch)
    c = self.conv_add3(b,edge_index)
```

```
_,edge_index = KNNAmp(k, c, batch)
d = self.conv_add4(c,edge_index)
x = torch.cat((x,a,b,c,d),dim = 1)
del a,b,c,d
x = self.nn1(x)
x = self.relu(x)
x = self.nn2(x)
a,_ = scatter_max(x, batch, dim = 0)
b,_ = scatter_min(x, batch, dim = 0)
c = scatter_sum(x,batch,dim = 0)
d = scatter_mean(x,batch,dim= 0)
x = torch.cat((a,b,c,d),dim = 1)
x = self.relu(x)
x = self.nn3(x)
x = self.relu(x)
x = self.nn4(x)
x[:,0] = self.tanh(x[:,0]) # ONLY WHEN PREDICTING ANGLES
x[:,1] = self.tanh(x[:,1]) # ONLY WHEN PREDICTING ANGLES


return x
```

Bibliography

[1] A.G. Van Melsen. *From Atomos to Atom: The History of the Concept Atom*. Dover phoenix editions. Dover Publications, 2004. ISBN: 9780486495842. URL: https://books.google.dk/books?id=pARpF9A2OtMC (cit. on p. 3).

[2] Carsten Jensen. *Controversy and Consensus: Nuclear Beta Decay 1911-1934*. Springer, 2000, pp. 10–22 (cit. on p. 5).

[3] Laurie M. Brown. "The Idea of the Neutrino". In: *Physics Today* (1978). URL: http://physics.gmu.edu/~rubinp/courses/440-540/undergrad/neutrino.pdf (cit. on p. 5).

[4] J. Nieves, M. Valverde, and M. J. Vicente Vacas. "Charged and Neutral Current Neutrino Induced Nucleon Emission Reactions". In: *Acta Phys. Polon. B* 37 (2006). Ed. by K. M. Graczyk and J. T. Sobczyk, p. 2295. arXiv: hep-ph/0605221 (cit. on p. 5).

[5] "The Reines-Cowan Experiments". In: *Los Alamos Science* 25.25 (1997). URL: https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-97-2534-02 (cit. on p. 6).

[6] C. L. Cowan et al. "Detection of the Free Neutrino: a Confirmation". In: *Science* 124.3212 (1956), pp. 103–104. ISSN: 0036-8075. DOI: 10.1126/science.124.3212.103. eprint: https://science.sciencemag.org/content/124/3212/103.full.pdf. URL: https://science.sciencemag.org/content/124/3212/103 (cit. on p. 6).

[7] Norb Dernbach. "Solar Energy Generation Theory Being Tested in Brookhaven Neutrino Experiment". In: *Brookhaven National Laboratory Press Releases* (1967). URL: https://www.bnl.gov/bnlweb/raydavis/1967PR.pdf (cit. on p. 7).

[8] Bruno Pontecorvo. ""Mesonium and Anti-Mesonium"". In: *Sovietic Journal of Experimental and Theoretical Physics* (1957), pp. 549–551. URL: http://www.jetp.ac.ru/files/pontecorvo1957_en.pdf (cit. on p. 8).

[9] "Particle Phenomenology". In: *Nuclear and Particle Physics*. John Wiley Sons, Ltd, 2006. Chap. 3, pp. 71–110. ISBN: 9780470035474. DOI: https://doi.org/10.1002/0470035471.ch3. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470035471.ch3. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470035471.ch3 (cit. on p. 9).

[10] Hiroshi Nunokawa, Stephen Parke, and José W.F. Valle. "CP violation and neutrino oscillations". In: *Progress in Particle and Nuclear Physics* 60.2 (Apr. 2008), pp. 338–402. ISSN: 0146-6410. DOI: 10.1016/j.ppnp.2007.10.001. URL: http://dx.doi.org/10.1016/j.ppnp.2007.10.001 (cit. on p. 10).

[11] WERNER RODEJOHANN. "NEUTRINO-LESS DOUBLE BETA DECAY AND PARTICLE PHYSICS". In: *International Journal of Modern Physics E* 20.09 (Sept. 2011), pp. 1833–1930. ISSN: 1793-6608. DOI: 10.1142/s0218301311020186. URL: http://dx.doi.org/10.1142/S0218301311020186 (cit. on p. 10).

[12] Z. Maki, M. Nakagawa, and S. Sakata. "Remarks on the Unified Model of Elementary Particles". In: *Progress of Theoretical Physics* 28.5 (Nov. 1962), pp. 870–880. DOI: 10.1143/PTP.28.870 (cit. on p. 10).

[13] K. N. Abazajian et al. *Light Sterile Neutrinos: A White Paper*. 2012. arXiv: 1204.5379 [hep-ph] (cit. on p. 11).

[14] Wikipedia. *IceCube Architecture Diagram*. 2009. URL: https://commons.wikimedia.org/wiki/File:Icecube-architecture-diagram2009.PNG (cit. on p. 14).

[15] K. Scholberg. ""The SuperNova Early Warning System"". In: *Astron.Nachr* (2008), pp. 337–339. URL: https://arxiv.org/abs/0803.0531 (cit. on p. 13).

[16] IceCube Collaboration. ""Neutrino emission from the direction of the blazar TXS 0506+056 prior to the IceCube-170922A alert"". In: *Science* (2018), pp. 147–151. URL: https://arxiv.org/abs/1807.08794 (cit. on p. 13).

[17] Carlos P. de los Heros. ""IceCube"". In: *Nuclear Instruments and Methods in Physics Research* (2011), pp. 119–124. URL: https://doi.org/10.1016/j.nima.2010.06.042. (cit. on p. 14).

[18] R. Abbasi et al. "Extending the Search for Neutrino Point Sources with IceCube above the Horizon". In: *Physical Review Letters* 103.22 (Nov. 2009). ISSN: 1079-7114. DOI: 10.1103/physrevlett.103.221102. URL: http://dx.doi.org/10.1103/PhysRevLett.103.221102 (cit. on p. 14).

[19] R. Abbasi et al. "Limits on a muon flux from Kaluza-Klein dark matter annihilations in the Sun from the IceCube 22-string detector". In: *Physical Review D* 81.5 (Mar. 2010). ISSN: 1550-2368. DOI: 10.1103/physrevd.81.057101. URL: http://dx.doi.org/10.1103/PhysRevD.81.057101 (cit. on p. 14).

[20] M. G. Aartsen et al. "Measurement of Atmospheric Neutrino Oscillations at 6–56 GeV with IceCube DeepCore". In: *Phys. Rev. Lett.* 120 (7 Feb. 2018), p. 071801. DOI: 10.1103/PhysRevLett.120.071801. URL: https://link.aps.org/doi/10.1103/PhysRevLett.120.071801 (cit. on p. 14).

[21] R. Abbasi et al. "The design and performance of IceCube DeepCore". In: *Astroparticle Physics* 35.10 (May 2012), pp. 615–624. ISSN: 0927-6505. DOI: 10.1016/j.astropartphys.2012.01.004. URL: http://dx.doi.org/10.1016/j.astropartphys.2012.01.004 (cit. on p. 15).

[22] R. Abbasi et al. "IceTop: The surface component of IceCube". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 700 (Feb. 2013), pp. 188–220. ISSN: 0168-9002. DOI: 10.1016/j.nima.2012.10.067. URL: http://dx.doi.org/10.1016/j.nima.2012.10.067 (cit. on p. 16).

[23] Lew Classen et al. "A multi-PMT Optical Module for the IceCube Upgrade". In: *PoS* ICRC2019 (2019), p. 855. DOI: 10.22323/1.358.0855 (cit. on p. 16).

[24] IceCube Collaboration. 2009. URL: https://icecube.wisc.edu/gallery/nsf-approves-funding-for-icecube-upgrade/#modulagallery-7003-2155 (visited on 03/05/2021) (cit. on p. 16).

[25] Aya Ishihara and Ayumi Kiriki. *Calibration LEDs in the IceCube Upgrade D-Egg Modules*. 2019. arXiv: 1908.10780 [astro-ph.HE] (cit. on p. 17).

[26] Aya Ishihara. *The IceCube Upgrade – Design and Science Goals*. 2019. arXiv: 1908.09441 [astro-ph.HE] (cit. on p. 17).

[27] IceCube Collaboration. *I3 Format Documentation*. 2009. URL: https://docs.icecube.aq/icerec/V05-01-04/projects/dataio/portable_binary_archive.html (cit. on p. 19).

[28] G. Carminati et al. *MUPAGE: a fast atmospheric MUon GEnerator for neutrino telescopes based on PArametric formulas*. 2009. arXiv: 0907.5563 [astro-ph.IM] (cit. on pp. 19, 20).

[29] D. Heck et al. "CORSIKA: A Monte Carlo code to simulate extensive air showers". In: (Feb. 1998) (cit. on pp. 19, 20).

[30] C. Andreopoulos et al. "The GENIE neutrino Monte Carlo generator". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 614.1 (2010), pp. 87–104. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2009.12.009. URL: http://www.sciencedirect.com/science/article/pii/S0168900209023043 (cit. on pp. 19, 20).

[31] L. Garren et al. "Monte Carlo particle numbering scheme". In: *European Physical Journal C - EUR PHYS J C* 15 (Mar. 2000), pp. 205–206. DOI: 10.1007/BF02683426 (cit. on p. 21).

[32] IceCube Collaboration. *OscNext*. 2009. URL: https://wiki.icecube.wisc.edu/images/8/82/OscNext_v00.04.pdf (cit. on p. 22).

[33] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. ISBN: 9781510860964 (cit. on p. 22).

[34] M. G. Aartsen et al. "Measurement of atmospheric tau neutrino appearance with IceCube DeepCore". In: *Physical Review D* 99.3 (Feb. 2019). ISSN: 2470-0029. DOI: 10.1103/physrevd.99.032007. URL: http://dx.doi.org/10.1103/PhysRevD.99.032007 (cit. on p. 23).

[35] SmartLaw. *SmartLaw Products Page*. URL: https://www.smartlaw.com.sg/ai-for-criminal-sentencing/ (cit. on p. 25).

[36] Reuters Staff. *China to bar people with bad 'social credit' from planes, trains*. URL: https://www.reuters.com/article/us-china-credit/china-to-bar-people-with-bad-social-credit-from-planes-trains-idUSKCN1GS10S (cit. on p. 25).

[37] Andrew Senior et al. "Improved protein structure prediction using potentials from deep learning". In: *Nature* 577 (Jan. 2020), pp. 1–5. DOI: 10.1038/s41586-019-1923-7 (cit. on p. 25).

[38] Nvidia Inc. *NVIDIA DLSS 2.0: A Big Leap In AI Rendering*. URL: https://www.nvidia.com/en-gb/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/ (cit. on p. 26).

[39] Alvaro Sanchez-Gonzalez et al. *Learning to Simulate Complex Physics with Graph Networks*. 2020. arXiv: 2002.09405 [cs.LG] (cit. on p. 26).

[40] David Furmo. *A Gentle Introduction To Neural Networks Series — Part 1*. URL: https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc (cit. on p. 27).

[41] Frederic B. Fitch. "Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133." In: *Journal of Symbolic Logic* 9.2 (1944), pp. 49–50. DOI: 10.2307/2268029 (cit. on p. 27).

[42] Daniel Jakubovitz, Raja Giryes, and Miguel R. D. Rodrigues. *Generalization Error in Deep Learning*. 2019. arXiv: 1808.01174 [cs.LG] (cit. on p. 30).

[43] Iosif Pinelis. "On inequalities for sums of bounded random variables". In: *Journal of Mathematical Inequalities* 2 (Jan. 2008). DOI: 10.7153/jmi-02-01 (cit. on p. 30).

[44] Brady Neal et al. *A Modern Take on the Bias-Variance Tradeoff in Neural Networks*. 2019. arXiv: 1810.08591 [cs.LG] (cit. on p. 31).

[45] A. Sperduti and A. Starita. "Supervised neural networks for the classification of structures". In: *IEEE transactions on neural networks* 8 3 (1997), pp. 714–35 (cit. on p. 33).

[46] Zhiqian Chen et al. *Bridging the Gap between Spatial and Spectral Domains: A Survey on Graph Neural Networks*. 2020. arXiv: 2002.11867 [cs.LG] (cit. on pp. 33, 34).

[47] Z. Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386 (cit. on p. 33).

[48] Bingbing Xu et al. *Graph Wavelet Neural Network*. 2019. arXiv: 1904.07785 [cs.LG] (cit. on p. 34).

[49] Xinye Chen. *A Note on Spectral Graph Neural Network*. Dec. 2020. DOI: 10.13140/RG.2.2.27579.03364/1 (cit. on p. 34).

[50] Pavel Kurasov. "Graph Laplacians and topology". In: *Ark. Mat.* 46 (Apr. 2008), pp. 95–111. DOI: 10.1007/s11512-007-0059-4 (cit. on p. 34).

[51] Muhan Zhang and Yixin Chen. *Link Prediction Based on Graph Neural Networks*. 2018. arXiv: 1802.09691 [cs.LG] (cit. on p. 35).

[52] Kiran K. Thekumparampil et al. *Attention-based Graph Neural Network for Semi-supervised Learning*. 2018. arXiv: 1803.03735 [stat.ML] (cit. on p. 35).

[53] Alejandro Escontrela. *Convolutional Neural Networks from the ground up*. URL: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1 (cit. on p. 35).

[54] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML] (cit. on p. 36).

[55] Nicholas Choma et al. *Graph Neural Networks for IceCube Signal Classification*. 2018. arXiv: 1809.06166 [cs.LG] (cit. on p. 37).

[56] Shah Rukh Qasim et al. "Learning representations of irregular particle-detector geometry with distance-weighted graph networks". In: *The European Physical Journal C* 79.7 (July 2019). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-019-7113-9. URL: http://dx.doi.org/10.1140/epjc/s10052-019-7113-9 (cit. on pp. 37, 39).

[57] Dawit Belayneh et al. "Calorimetry with deep learning: particle simulation and reconstruction for collider physics". In: *The European Physical Journal C* 80 (July 2020). DOI: 10.1140/epjc/s10052-020-8251-9 (cit. on p. 37).

[58] Matthew T. Dearing, Xiaoyan, and Wang. *Analyzing the Performance of Graph Neural Networks with Pipe Parallelism*. 2020. arXiv: 2012.10840 [cs.LG] (cit. on p. 38).

[59] Yue Wang et al. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV] (cit. on p. 42).

[60] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV] (cit. on p. 42).

[61] Sachin Kumar and Yulia Tsvetkov. *Von Mises-Fisher Loss for Training Sequence to Sequence Models with Continuous Outputs*. 2019. arXiv: 1812.04616 [cs.CL] (cit. on p. 47).

[62] Hongyang Gao and Shuiwang Ji. *Graph U-Nets*. 2019. arXiv: 1905.05178 [cs.LG] (cit. on p. 50).

[63] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. *ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations*. 2020. arXiv: 1911.07979 [cs.LG] (cit. on p. 50).

[64] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. *Self-Attention Graph Pooling*. 2019. arXiv: 1904.08082 [cs.LG] (cit. on p. 50).

[65] Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. *Understanding Attention and Generalization in Graph Neural Networks*. 2019. arXiv: 1905.02850 [cs.LG] (cit. on pp. 50, 51).

[66] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG] (cit. on p. 51).

[67] Andrew Brock et al. *High-Performance Large-Scale Image Recognition Without Normalization*. 2021. arXiv: 2102.06171 [cs.CV] (cit. on p. 52).

[68] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120 [cs.LG] (cit. on p. 54).

[69] Liyuan Liu et al. *On the Variance of the Adaptive Learning Rate and Beyond*. 2020. arXiv: 1908.03265 [cs.LG] (cit. on p. 55).

[70] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on p. 58).

[71] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG] (cit. on p. 58).

[72] Marie-Julie Rakotosaona et al. *PointCleanNet: Learning to Denoise and Remove Outliers from Dense Point Clouds*. 2019. arXiv: 1901.01060 [cs.GR] (cit. on p. 86).