GNNs for low-level filters in IceCube using GraphNeT

Bachelor's Thesis in Physics
Elite Study Program

handed in by
Fabian Wohlfahrt
08/05/2024

Erlangen Center for Astroparticle Physics
Friedrich-Alexander-Universität Erlangen-Nürnberg

Supervisor: Prof. Dr. Claudio Kopper
Advisor: Dr. Christian Haack

Abstract

In this thesis, I present two Graph Neural Networks (GNNs) trained for filtering tasks in IceCube: StartingEventFilter and DoubleBangFilter. DoubleBangFilter is designed to detect double bang events but currently shows suboptimal performance, with a low purity of $6.68 \cdot 10^{-6}$ and an efficiency of 58.3% using an $f_{0.1}$-optimized cut. Strategies to improve upon this are discussed however. Conversely, StartingEvent-Filter, analogous to the existing HESE and MESE filters but covering the entire detector volume, demonstrates very good performance in the medium-low energy range using an $f_{0.15}$-optimized cut. Starting events are retained with an efficiency of 63% and the purity is 40 times higher than that of MESE at $1.11 \cdot 10^{-3}$. Using a cut of 0.9995 it even achieves never-before-seen perfect purity over a wide energy range at the cost of reduced efficiency at lower energies, while keeping 49% efficiency at above $100 TeV$.

Zusammenfassung

In dieser Arbeit stelle ich zwei Graph Neural Networks (GNNs) vor, die für den Einsatz als Filter in IceCube entworfen und trainiert wurden: StartingEventFilter und DoubleBangFilter. DoubleBangFilter ist konzipiert, um Double-Bang-Events zu erkennen, zeigt jedoch derzeit eine suboptimale Leistung bei einem $f_{0.1}$-optimierten Schwellenwert, mit einer niedrigen Sample-Reinheit von $6.68 \cdot 10^{-6}$ bei einer Effizienz von 58.3%. Möglichkeiten, diese zu verbessern, werden allerdings diskutiert. Im Gegensatz dazu zeigt StartingEventFilter, der analog zu den bestehenden HESE- und MESE-Filtern ist, aber das gesamte Detektorvolumen abdeckt, eine sehr gute Leistung im mittel-niedrigen Energiebereich zeigt bei Verwendung eines $f_{0.15}$-optimierten Schwellenwertes. Mit einer Effizienz von 63% werden Starting-Events beibehalten und mit $1,11 \cdot 10^{-3}$ ist die Sample-Reinheit 40 mal höher als jene von MESE. Bei Verwendung eines Schwellenwertes von 0.9995 wird sogar eine nie gesehene perfekte Sample-Reinheit erzielt, dafür ist dann jedoch die Effizienz für geringere Energien reduziert, überhalb von $100 TeV$ werden allerdings 49% Effizienz beibehalten.

Acknowledgement

This project was conducted at Erlangen Center for Astroparticle Physics (ECAP). Thanks for everyone involved in making this possible, especially my advisor Christian Haack and my colleague Marvin Hoffmann for the great collaboration, as well as for the welcoming atmosphere at the institute. Also big thanks to Rasmus Ørsøe and the GraphNeT team for their helpful advise concerning any issue that came up concerning the model training.

Erlangen Centre for Astroparticle Physics

Contents

1 Introduction

The IceCube Neutrino Observatory, located at the South Pole, is a groundbreaking facility in neutrino astronomy. This cubic-kilometer detector, embedded in Antarctic ice, observes high-energy neutrinos that offer insights into the universe's most energetic phenomena. Neutrinos' weak interaction with matter allows them to traverse vast distances unimpeded, making them excellent cosmic messengers. IceCube was constructed with three main scientfic goals: measuring the diffuse high-energy cosmic neutrino flux, identifying point sources and determining the neutrino flavor ratio. These quantities will give insights into neutrino production scenarios and the inner workings of their sources. To date, a high-energy neutrino flux has been detected from active galactic nuclei in 2018 and 2023 ([1] [2]) and the Milky Way in 2023 ([3]).
IceCube measures neutrinos indirectly by detecting the Cherenkov radiation emitted when they interact with the ice. This is done using more than 5,000 photomultiplier tubes (PMTs) deployed in the glacial ice. Unfortunately, the raw detector data are highly contaminated with background events caused by air shower-induced muons with a signal-to-noise ratio (SNR) of about one to one million. It is therefore essential to purify the data and increase the SNR using specially designed filters that can discriminate between signal and background events, discarding the latter.
For the filters to produce workable data selections with acceptable SNRs, they must be incredibly accurate. To achieve even a modest SNR of one to one, assuming the filter does not falsely discard any neutrino events as background, only one in one million background events can be falsely accepted as signal. This extraordinary required performance results from the poor raw SNR. Designing such accurate filters, though challenging, has been achieved at the cost of long processing times due to the sophisticated underlying algorithms. Some filters can take over an hour to process individual events. In order to safe on processing time, these slow high level filters should not be run on too many events, it is therefore advisable to optimize lower level filters running before them to pre-filter as much background as possible.
Graph neural networks (GNNs) have proven to be a powerful tool in IceCube data analysis, having already been applied successfully in energy, direction and vertex reconstruction tasks as well as filtering tasks as demonstrated by [4]. The development of the GraphNeT Python library for machine learning in IceCube and other neutrino telescopes has accelerated the adoption of GNNs drastically, leading to a surge in research efforts involving them since 2022. Consequently, the idea emerged to apply GNNs to filters. These filter networks promise to combine the best of both worlds: highly accurate filtering performance with significantly fast runtimes. Additionally, GNNs offer an easy framework to quickly and conveniently build any desired filter. Traditionally, developing a filter requires extensive and time-consuming work to determine the optimal geometric characteristics of the events to be selected and finely tune all parameters. With neural networks, however, designing a filter is as

simple as training the model with an adequate loss function that maps events to their specific group according to their labels provided in the training data. The training algorithm then handles the complex and otherwise tedious task of finding the best criteria and specifications for event discrimination by tuning the model parameters, saving users considerable time and effort. The approach also allows for the creation of multi-classifiers, that separate events into multiple different groups instead of just two (pass, not pass) like current filters do. Potentially one could even create completely new filters previously unimaginable due to the complexity of the required discrimination criteria and algorithms.
The current online filter infrastructure (which is the pre-filtering done at the south pole) is rather outdated by now, with many filters being almost a decade old or more. It is therefore set to be replaced soon, providing the perfect opportunity to upgrade to AI based filters like GNNs that promise superior performance.
This thesis provides a preliminary exploration of this idea and presents results from two test networks: a starting event filter ('StartingEventFilter', subsection 6.1) and a double bang filter ('DoubleBangFilter', subsection 6.2). StartingEventFilter is compared to the corresponding existing online filters HESE and MESE and demonstrates the huge potential of GNNs in replacing them. For DoubleBangFilter no suitable filter for comparison was found, however a lot can be learned from it about the limits of GNN filters even when viewed standing alone.
Based on the results and findings from these test models, conclusions about the prospects of GNN filters in IceCube are drawn and possible next steps regarding them are discussed.

2 Neutrinos

2.1 History of Neutrino Physics

2.1.1 Prediction

In beta-decays of radioactive nuclei the decaying nucleus emits an electron and an antielectronneutrino. Before the prediction of the neutrino, physicists thought that the decay only emits an electron, which under consideration of energy-momentum-conservation implied a fixed energy for the electron. This was however inconsistent with observations which clearly showed a continuous energy spectrum. Wolfgang Pauli proposed the existence of the (antielectron-)neutrino in a 1930 letter to his "[r]adioactive" colleagues in order to fix the inconsistency (he called it the neutron, the name neutrino was only coined a year later by Enrico Fermi; the particle known as neutron today is a different one) ([5]), the logic behind it being that if the beta-decay emits two particles, the fixed energy can distribute over both of them, allowing for a continuous spectrum of the electron.
After the discovery of the neutron in 1932 by James Chadwick, Enrico Fermi came

up with the correct theory of beta-decay in 1934: a neutron in the decaying nucleus undergoes the following decay reaction:

$$n^0 \longrightarrow p^+ + e^- + \overline{\nu_e}.$$

[6]. The proton thereby stays bound in the nucleus while the electron and neutrino are being emitted.

2.1.2 Experimental Discovery

The first experimental detection of neutrinos happened in 1956 in the Cowan–Reines neutrino experiment ([7]). It was based on beta-capture, an effect in which a neutrino (in this case from a nuclear reactor) hits a proton of a nucleus and in rare instances reacts to produce a neutron and a positron.

$$\overline{\nu_e} + p^+ \longrightarrow n^0 + e^+$$

The positron would then annihilate with an electron producing two gamma photons and the neutron would be absorbed by a prepared cadmium solution also producing a gamma photon in the process. This special three gamma signature of the beta-capture process is unique, making it possible to confirm the existence of neutrinos with it.

2.2 Neutrino Sources

2.2.1 Astrophysical Neutrinos

As can be seen in Figure 1 the largest astrophysical neutrino flux on earth stems from our sun. It runs mostly on the pp-fusion-process, which has the net fusion reaction ([9]):

$$4p^+ \longrightarrow^4 He^{2+} + 2e^+ + 2\nu_e (+26.731 MeV) \tag{1}$$

Accordingly for every ^4He core fused in the sun two electron neutrinos are produced with a spectrum peeking in the hundreds of TeVs as can be seen in Figure 1.
Neutrinos also play an essential role in supernova-explosions ([10]). Even though their interaction-cross-sections are extremely small, during a supernova collapse the stellar matter becomes so extremely dense, that the neutrinos from fusion processes can no longer escape the infalling matter, they instead build up pressure inside, which eventually overcomes the gravitational binding and causes the highly energetic explosion in which also the neutrinos are released. The resulting spectrum of neutrinos can again be seen in Figure 1.

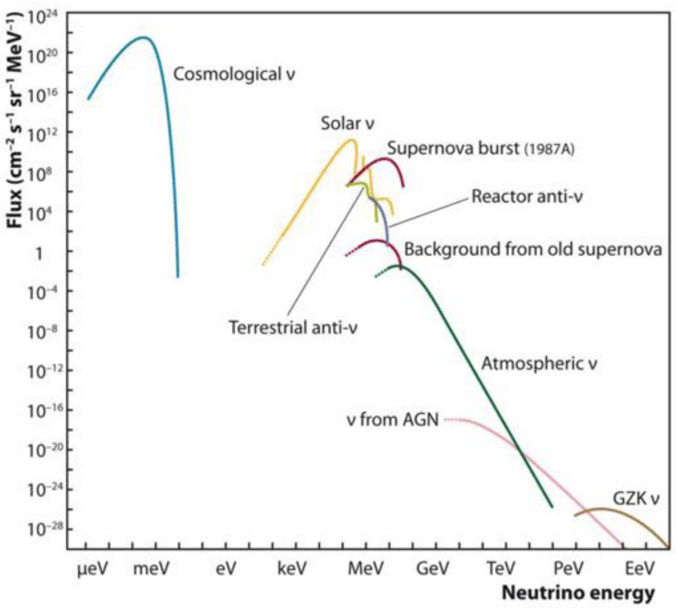

Figure 1: The spectrum of neutrinos as observed at earth for different sources. The figure has been taken from [8].

2.2.2 High Energy Astrophysical Neutrinos

The astrophysical neutrino sources mentioned above only produce neutrinos up to the MeV regime. There are however also astrophysical neutrinos of far higher energies reaching the PeV regime and above as can be seen in Figure 1 (ν from AGN). These are produced in the decays of high energy mesons, mostly pions and kaons, that themselves are produced in interactions between cosmic rays ([11], chapter 11.4).
In CR sources like supernova remnants (SNR) or active galactic nuclei (AGN) charged cosmic rays are accelerated to insane energies up to the EeV regime through processes like diffusive shock acceleration ([12]).

High energy protons from these sources can then interact with other protons or photons in the surrounding radiation field to produce pions or heavier mesons. The underlying interactions look as follows ([13], [14]):

$$\begin{aligned}
p^+ + p^+ &\longrightarrow p^+ + p^+ + pions \\
\gamma + p^+ &\longrightarrow n^0 + \pi^+ \\
\gamma + p^+ &\longrightarrow p^+ + \pi^0 \\
\gamma + n^0 &\longrightarrow p^+ + \pi^-
\end{aligned}$$

The production of other mesons like kaons works analogously. These mesons inherit a considerable fraction of the primary proton or neutron's energy. They are unstable themselves and decay and thereby produce neutrinos in the following ways ([11], equations 6.1, 6.2 and chapter 11.1):

$$\begin{aligned} \pi^{\pm} &\longrightarrow \mu^{\pm} + \nu_{\mu}(\bar{\nu_{\mu}}) \qquad (BR \approx 100\%) \\ K^{\pm} &\longrightarrow \mu^{\pm} + \nu_{\mu}(\bar{\nu_{\mu}}) \qquad (BR \approx 63.5\%) \\ \pi^{0} &\longrightarrow 2\gamma \end{aligned} \tag{2}$$

The thereby produced muons subsequently also decay, producing more neutrinos ([11], equation 6.3).
In conclusion pion-decays produce both gamma rays and neutrinos, so wherever there are gamma rays being produced by pion-decays, there must also be neutrinos being produced.
Not all gamma rays are produced through pion-decays though, the gamma emission of pulsars for example is theorized to stem from inverse-Compton-scattering ([15]). This is an elastic scattering process between an electron an a photon, which exchange energy in the process. Thereby an uhe electron accelerated by the pulsar, can scatter off a low energy photon, transferring a lot of energy to it and thus making it a gamma ray.
So, when searching for neutrinos coming from a source on the basis of there being gamma rays, one needs to be careful about which process the gammas stem from.
Sources, where an astrophysical neutrino flux is expected from, are amongst others ([11], chapter 18.7):

- supernova remnants (SNRs)
- active galactic nuclei (AGNs)
- gamma ray bursts (GRBs)
- galaxy clusters

The low flux of neutrinos from these sources combined with the low detection efficiency (dictated by the low interaction cross-sections of neutrinos) makes it very hard in practice to identify the sources. There are only three sources that could be confirmed via IceCube measurements so far:

- the blazar TXS 0506+056 (an AGN) in 2018 ([1])
- the Seyfert-II-galaxy NGC 1068 (also an AGN) in 2022 ([2])
- the Milky Way as a whole in 2023 ([3])

2.2.3 Atmospheric Neutrinos

When cosmic rays are incident on the earths atmosphere they produce air-showers because they have large interaction cross sections with it ([16]). In hadronic showers as created by protons or other nuclei meson production occurs that produces neutrinos in their decays according to Equation 2. This is the exact same process as that of the high energy astrophysical neutrino production, just that it happens on earth.

As the dominant decay channel of both pions and kaons (the mesons most commonly created in showers) is the muonic one, the atmospheric neutrino flux is dominated by (anti-)muon neutrinos ([11], chapter 6.6). For low energies, where the muons decay before reaching the ground, there is however also a significant fraction of electron neutrinos (stemming from their decays) at a flavor ratio of 1:2 ($\overset{(-)}{\nu_e} : \overset{(-)}{\nu_\mu}$) ([11], chapter 6.1). The flux of atmospheric neutrinos is zenith dependent, as can be seen in Figure 2 (right). This is mainly a geometric effect of the position of the observer wrt. the atmosphere. At energies in the TeVs neutrino absorption in the earth starts playing a role and further decreases the flux of upgoing muons, figure 2 in [17] shows the probability for neutrino transmission through the earth in dependence of zenith and energy. Figure 2 (left) shows the relation of the magnitudes of the fluxes of atmospheric and astrophysical neutrinos, for energies up to ca. $30TeV$ the atmospheric flux dominates and at energies higher than that the astrophysical flux dominates.

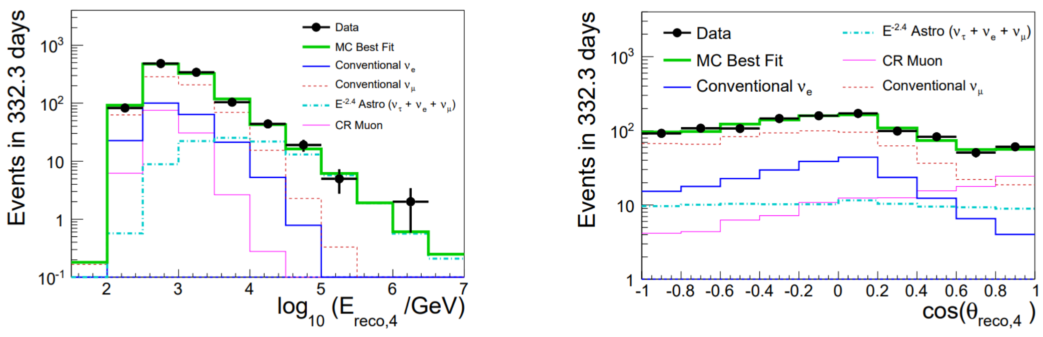

Figure 2: left: The spectrum of atmospheric and astrophysical neutrinos, the figure was taken from [18], figure 6. At about $30TeV$ the astrophysical flux starts dominating the atmospheric one. ; right: the zenith dependence of the two fluxes, the figure was taken from [18], figure 8. The atmospheric flux is zenith dependent, having its maximum at a zenith of $\theta = 0$, while the astrophysical neutrino flux is zenith independent.

For a highly detailed summary of atmospheric neutrinos see [11], chapter 6.6.

3 IceCube

The huge success of the Japanese (Super-)Kamiokande experiment (1996-) ([19]) and the Canadian Sudbury Neutrino Observatory (1999 - 2006) ([20]) in measuring the solar neutrino flux and verifying the existence of neutrino oscillations proved the concept of detecting neutrinos via Cherenkov radiation of their interaction products using photomultiplier tubes. However it was realized that in order to detect neutrinos of higher energies (as are expected from high energy astrophysical sources), for which the flux is orders of magnitude lower, one needs to build a way bigger detector, one with a volume on the cubic-kilometer-scale. So the planning of what lead to the IceCube neutrino detector began. Initially, as a further proof of concept (for using ice instead of water as the interaction-medium) AMANDA, a set of 19 test strings, was deployed between 1996 and 2000 ([21]). When those tests were successful and the funding was approved, construction for IceCube began in January 2005 and was finished in January 2011. The project cost a total of 279 million US$ ([21]).

3.1 Detector Setup

This Chapter summarizes the most important information on the IceCube neutrino detector on the basis of [21] and [22], which describes the entire detector instrumentation in much detail.
The IceCube neutrino detector detects light emitted by particles through Cherenkov radiation ([23]). Neutrinos themselves are uncharged, so they do not emit Cherenkov radiation. That's why in order to detect them, they have to react and produce charged secondaries in the process. Given the low interaction cross section with other particles, only a tiny fraction of the neutrinos incident on the detector is detectable. In order to capture as many of them as possible, neutrino detectors have to have very large instrumented volumes. This makes it economically impossible to build the detector completely from scratch, so one relies on naturally occurring media to build the detector into. For this there are two options: either one uses water in form of lakes or seas/oceans or one uses ice, like the antarctic glacial ice. The latter is chosen for IceCube. Conveniently there is a US-american polar research station (Amundsen-Scott-Station) directly at the south pole ([24]), which provides useful infrastructure that is also used for the IceCube project.
The IceCube detector itself covers a volume of roughly $1km^3$ in the ice (thus the name Ice<u>Cube</u>). Into this volume 2.5km deep holes were drilled into which strings fitted with so-called DOMs were deployed and the holes were allowed to freeze again. DOM is short for Digital-Optical-Module, they are glass spheres 35cm in diameter that form a pressure housing to protect the electronics inside. These electronics can be summarized into three parts. First, there is a photo-multiplier-tube (PMT), that converts incident photons into an electrical signal. Second, there are circuits to digitize that signal and then send it up to the IceCube laboratory on the surface. Lastly, there are LED flashers that are used to measure the optical properties of the

glacial ice. In Figure 3 one such DOM is shown with its most important components labeled.

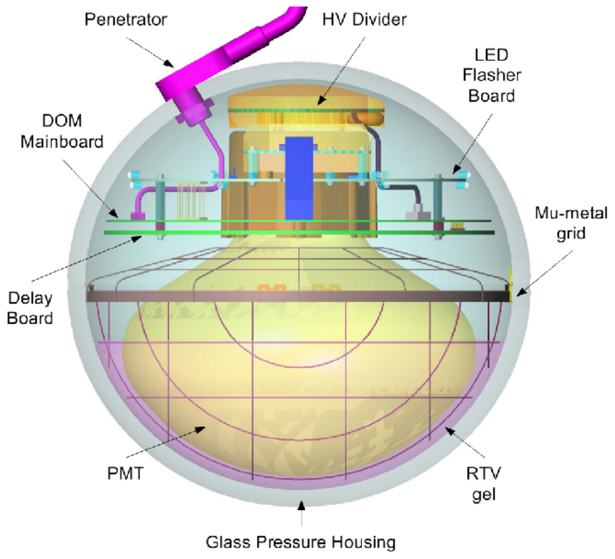

Figure 3: cross section view of a digital optical module (DOM) ([25]), the core component: the downward facing PMT is shown in gold

The detector has a total of 86 strings arranged in a hexagonal pattern, which hold a total of 5160 DOMs, as is visualized in Figure 4. The string spacing is 125m and the distance between two DOMs on the same string is 17m for the largest part of the detector. The DOMs are placed from a depth of 1450m to 2450m beneath the ice surface. In the very center of the detector the spacings are smaller, 70m between strings and 7m within the strings, this region is called DeepCore and has a lower energy threshold for detecting neutrinos than the rest of the detector has, it can therefore be used to study neutrino oscillations.
Additionally, there is a system called IceTop, which, as the name suggests, is a surface detector array on the surface of the ice above IceCube. It detects cosmic rays, that IceCube itself cannot detect, because they don't reach deep enough into the ice.
Lastly, there is the IceCube Lab, which is essentially a big server that collects, preanalyzes and filters the data coming from the DOMs before it sends them to University of Wisconsin-Madison via satellite.

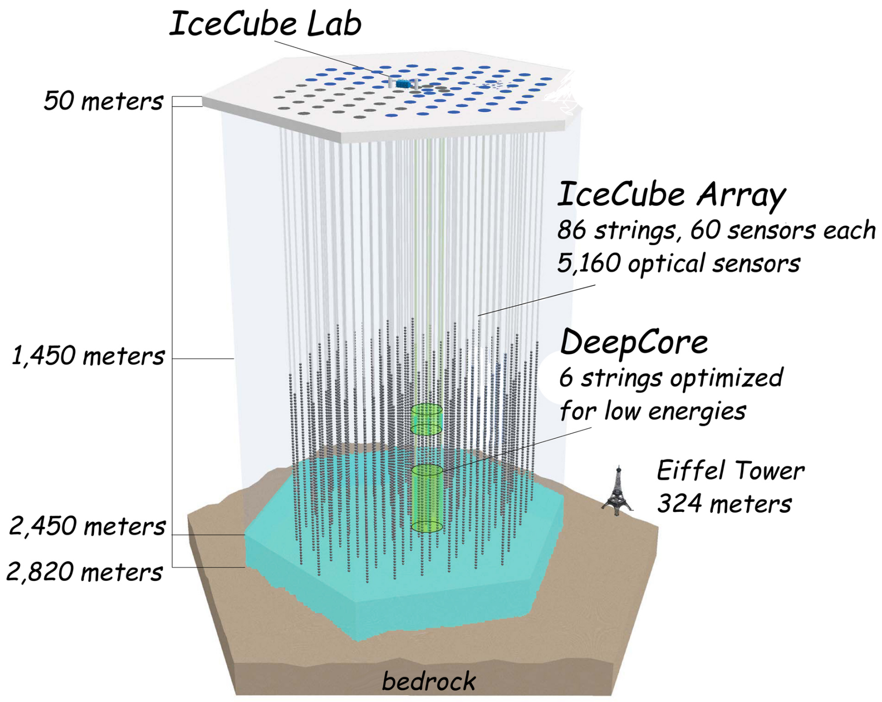

Figure 4: schematic view of the IceCube detector ([26]), the Eiffel Tower is shown to demonstrate the enormous size of the detector

3.2 Neutrino Interactions in Matter

As explained above, neutrinos can only be detected indirectly, i.e. when they interact with the ice and produce charged secondary particles, which emit Cherenkov radiation.
The primary neutrino interactions in IceCube's high energy range of $100 GeV$ to $100 PeV$ can be summarized into three groups: two kinds of deep inelastic scattering (DIS) (NC and CC) and the Glashow-resonance ([27], VI and VII).

3.2.1 NC-interactions

NC is short for neutral-current. These are interactions in which the neutrino exchanges a Z^0-boson with a nucleus, which destroys the nucleus, causing a hadronic cascade ([27], VI and VII). It is therefore a deep inelastic scattering interaction. The neutrino is left unchanged except for the momentum transfer, the interaction is therefore only visible via the hadronic cascade and looks the same for all neutrino flavors, so that one cannot distinguish between them.

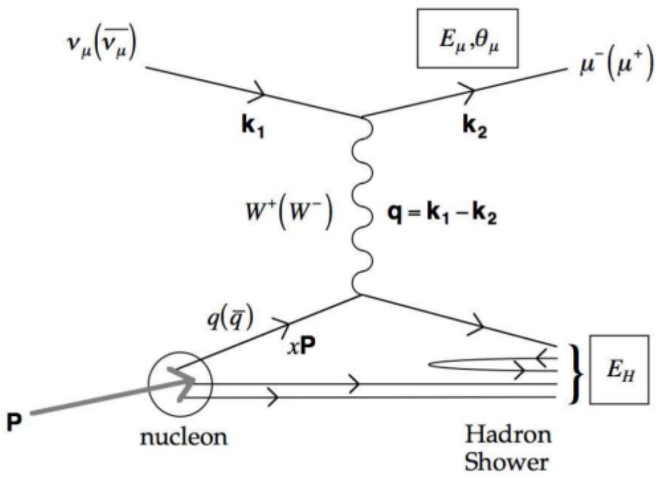

Figure 5: The Feynman diagram of a deep inelastic scattering (DIS) CC-interaction; the NC-interaction looks essentially the same, but with a Z^0-exchange and consequentially an outgoing neutrino instead of a charged lepton. (graphic taken from [27], figure 27)

3.2.2 CC-interactions

CC is short for charged-current. These are deep inelastic scattering interactions similar to the NC ones. The difference is here that the neutrino exchanges a W$^\pm$-boson with the nucleus and thereby turns into the corresponding (anti-)lepton ([27], VI and VII). This resulting (anti-)lepton produces different signals in the detector depending on its flavor (see subsection 3.3), making it theoretically possible to distinguish the parent neutrino's flavor, this the big advantage of CC-interactions over NC-interactions. It is notable that the CC-interaction discriminates between neutrinos and antineutrinos. As a result for energies in the hundreds of GeVs the interaction cross section is twice as large for neutrinos as it is for anti neutrinos ([27], figure 28), for very large energies in the tens of TeVs or more the difference becomes hardly notable however ([27], VII). For these high energies the ratio of the interactions cross sections between CC and NC is roughly 2.4 ([27], VII and figure 29).

3.2.3 Glashow-resonance

The Glashow-resonance (GR) is the interaction, where an electron antineutrino interacts with an electron of an atom to form a W$^-$-boson. Due to energy-momentum-conservation, this is a resonant interaction, with a peek at $6.3 PeV$ and a resonance width of $2.08 GeV$ ([27], VII). Around its peek the Glashow-resonance is by far the dominant interaction for antielectron neutrinos ([27], figure 29). The

W$^-$-boson has an extremely small lifetime and thus instantly decays into either a lepton/antineutrino-pair (electron: 10.7% branching ratio (BR), muon: 10.6% BR, tau: 11.4% BR) or a quark/antiquark-pair (67.4% BR) ([28]). Due to the many different decay channels Glashow-resonance events can look very different in the detector.

3.3 Signal Types

3.3.1 Cascades and Tracks

Events in IceCube can be and look very different, a detailed but still incomplete list of classifications can be found in table Table 5 (coming up with a complete list is very hard if not impossible as there's so many interactions and particles (see subsection 3.2) and containment types at play). What can however be distinguished are two types of detector signals that the neutrinos' interaction products cause: cascades and tracks/spurs. They are fundamentally different in their signal geometry: cascades are point-like light sources, while tracks and spurs are emit light along a line.

Whether a particle causes a cascade or a track/spur depends on its interaction cross sections with the nuclei of the ice as well as its lifetime. These two factors determine, how far a particle can propagate through the ice. Particles with very large cross sections and/or very short livetimes will loose all of their energy quickly after formation in a comparably short distance of a few meters to a few tens of meters, this is seen as a cascade. Particles with smaller cross sections and longer livetimes can propagate through the ice for distances on the order of hundreds of meters or kilometers and deposit their energy in multiple spatially separated interactions along the way, this is seen as a track/spur.

Hadrons are all cascading particles as they interact dominantly via the strong force and therefore have large interaction cross sections, the characteristic length of hadronic cascades is given by the nuclear interaction length, which is $90.8cm$ for ice ([29]).

Gamma photons (in IceCube's energy range) are also cascading particles, as they have a small Bethe-Heitler interaction length of $50.5cm$ ([30], figure 34.19).

Charged leptons behave very differently depending on their flavor. They do not interact strongly so the dominant interactions here are electromagnetic. They can be split into the continuous component: the ionization loss and the stochastic component: bremsstrahlung, pair production and photonuclear interactions ([31], 9 Formulae provides detailed parameterizations of their respective cross sections). The ionization loss is very roughly energy independent, while the stochastic losses increase approximately linearly with energy, so that they dominate at the relevant high energies in IceCube ([31], see figure B.2). The stochastic energy losses decrease massively with the lepton mass.

Consequentially the electron with its light mass has a very high loss rate, making it

interact in short succession after formation and therefore a cascading particle. The characteristic length of such an electron induced cascade is given by the radiation length, which is $39.3cm$ in ice ([30], figure 34.19).
Muons have a mass about 207 times larger than that of the electron making their loss rate orders more than four orders of magnitude smaller ([31], see figure 35 and 36), such that they can propagate through the detector and cause a track. The range of a track is about $1.5km$ at $400GeV$ and about $20km$ at $1PeV$ ([31], figure 25). The finite livetime of muons plays an insignificant role at high energies such that the muon only decays after having lost nearly all its energy in interactions.
Taus have a mass of again about 17 times larger than that of the muon, making their loss rate again much smaller which means that they can travel essentially freely through the detector. For their signal type the new term 'spur' is introduced here to semantically stress the far lower intensity of their otherwise track-like signature. Spurs are usually short lived nevertheless, this is due to the small livetime of the tau (it travels on average only about $49m/PeV$ before decaying). In its decay the tau either produces a muon (17.4% branching ratio (BR)), which leaves a track, or it produces an electron (17.8% BR), which creates an electromagnetic cascade, or it decays hadronically (rest: 64.8% BR), which creates a hadronic cascade ([32]).
In terms of reconstruction the different signal types offer very different possibilities: Cascades are great for energy reconstruction, because a rather large fraction (in some cases even all) of the neutrino's energy is deposited within the detector volume through the respective cascade ([33]).
The line like signature of tracks makes them at the other hand very well suited to accurately reconstruct the neutrino's arrival direction and thus identify potential astrophysical neutrino sources ([34]).

3.3.2 Double Bangs

One particular event type that sticks out is the so-called double bang (also known as double cascade). It corresponds to the case when a tau is first produced from a tau neutrino in a CC-interaction and then subsequently decays either electronically or hadronically. I has a unique signature in that it has two spatially separate cascades: the hadronic one from the CC-interaction and the electromagnetic or hadronic one from the tau decay ([35], 2.1). All other events have at most one cascade with the exception of an electron neutrino CC-interaction which has an electromagnetic and a hadronic cascade but they are spatially coincident so that they appear as one). The separation length of the two cascades is given by the decay length of the tau, it increases linearly with energy at a rate of $49m/PeV$, because of this energy dependence only double bangs at high energies have sufficient separation to be able to recognize them properly, subsection 6.2 discusses this in more detail. Because double bangs are exclusively caused by tau neutrinos detecting one is equivalent to detecting a tau neutrino. Given that they are additionally (for sufficient separation of the two cascades, i.e. high enough energies) easy to distinguish from other event types in the detector, they present an optimally suited approach for tau neutrino research.

3.4 Atmospheric Background

3.4.1 Atmospheric Neutrino Background

One source of background to IceCube's astrophysical neutrino measurements are the atmospheric neutrinos described in subsubsection 2.2.3. They are hard to distinguish from astrophysical neutrinos on a per-event basis, because they are physically exactly the same particles. Commonly they are dealt with statistically, in spectrum analyses one simply subtracts the well known atmospheric spectrum (shown in Figure 2 from the total measured neutrino spectrum in order to obtain the astrophysical one. One way that has proven effective for dealing with atmospheric neutrinos on a per-event basis is looking for coincident muons. As muons are produced in the same air showers as atmospheric neutrinos, they are accompanied by them (at least ones coming from the south where the muons are not absorbed by the earth). [36] describes and uses this approach to measure the astrophysical neutrino flux.

3.4.2 Atmospheric Muon Background

The second and much bigger source of background in IceCube are the already mentioned atmospheric muons. They are produced in CR induced air showers as well, more precisely both muons and neutrinos are produced in the same meson decays as can be seen in Equation 2, so that for every neutrino also a muon (or a different lepton, but muon production is dominant) is produced ([11], chapter 6.1, 6.2). [11], chapter 6.3 elaborates in detail on the flux of muons in the atmosphere. Other than neutrinos, of which the vast majority flies through the detector without interacting and is thus unnoticed, muons incident on the detector are always visible via their tracks. Therefore the rate of muon background events is much larger than that of neutrino ones. Electrons and taus from air showers do unlike muons not produce background events, as electrons will cascade at the latest when hitting the surface of the ice or earth and taus decay before reaching the detector. Also muons have one big limitation in reaching the detector, they can only do it if they come from the southern hemisphere (above the detector), as the ones coming from the north would need to traverse thousands of kilometers of rock of the earth, which they cannot do as they loose all their energy in interactions on the way and eventually decay before reaching the detector. Even though only muons from half of the atmosphere can reach the detector, the background they cause is still dominant over neutrino events by a factor of 10^6 ([37]), meaning that for every detected neutrino event there are one million atmospheric muon events, it is therefore essential to the neutrino research in IceCube that one can reliably filter out this dominant muon background, filters deigned to do this are described in subsection 3.5. [37] gives a

detailed characterization of the atmospheric muon background.
In practice only the muon background plays a role for filters, this is why whenever the word background comes up in the following, it refers to background muons, not neutrinos.

3.5 Filters and Selections

Filters are boolean discriminators that test an event for a condition and either outputs True is the condition is met or False if the condition is not met. There's two categories of filters in IceCube: online and offline filters:
Online filters run on the live datastream of the detector at the south pole. They exist because the raw detector trigger rate, which varies between $2.5kHz$ and $2.9kHz$ ([22]), is too large to be uplinked to the north via satellite, this is however necessary for quick and efficient data transfer and analysis. Online filters are therefore designed to reduce the rate of events to be transferred to a manageable one somewhere in the upper tens of Hz by throwing away clear background events right away while maintaining a high efficiency (falsely rejecting as little signal events as possible). In order to be able to run them on the live datastream they have to be optimized to have short runtimes and are therefore based on simple reconstruction algorithms and cuts. All events that do not pass online filtering get stored on hard drives at the pole and need to be physically be shipped north.
Offline filters run independently of the datastream on the saved events. Offline filters have the ultimate goal of producing selections of events with high purity (a small contamination with background events) and good reconstructions that are suitable for event-level analysis. To achieve this many different filter stages are run consecutively, those are the so called filter levels. The higher the filter level, the higher the purity of the sample it produces and the better the involved reconstruction techniques. In order to achieve the required filtering performance high-level filters need to be very sophisticated and complex which is why they tend to have long runtimes, in rare cases even on the order of hours.
The following outlines the filters included in the online filtering stage which is relevant in this thesis.

3.5.1 Muon Filter

The Muon Filter ([38]) is the first online filter that is run on all triggered events in IceCube, it is designed to select neutrino track events.
To do this a likelihood reconstruction is performed, which gives both a direction estimate and a log-likelihood value for the hypothesis of the event being a track event. The filter decision is then made according to different cuts in two regions, up- and down-going which are defined via the reconstructed zenith angle from the fit. These regions are fundamentally different because in the up-going region one does not have to deal with background muons as they are absorbed by the earth as

described in subsubsection 3.4.2 whereas in the down-going regions there is lots of background muons. In the up-going region a cut on the log likelihood is made. The down-going region is further divided up into different zenith regions and in each of them a cut on the total number of photoelectrons (NPE) in the event is made with different thresholds, no cut on the log-likelihood is made here. As different cuts are made in different regions the filtering is not isotropic but discriminates events by their arrival direction.
The filter rate of the Muon Filter is about $30Hz$.

3.5.2 Cascade Filter

The Cascade Filter ([39]) defines two regions according to the zenith angle from the Muon Filter's LLH fit. For up-going events a log-likelihood value estimates how likely the event is a cascade and then a decision is made according to a specified threshold. For down-going events additional cuts are made on a quantity related to the tensor of inertia to quantify the roundness of the event geometry as well as on a velocity estimate from a simple line fit.
The filter rate of the Online Cascade Filter is about $30Hz$.

3.5.3 Online Level 2 Muon Filter

The Online Level 2 Muon Filter ([40]) is applied to events that passed the Muon Filter and has the purpose of further refining its selection. It is not run right away but after the Muon Filter in order to save on computing time.
It is based on an MPE fit, which is more accurate than the LLH fit used in the Muon Filter and then defines different cuts on four different regions defined via the reconstructed zenith angle of the event. The cut quantities include next to various quantities of the MPE fit also the log-likelihood from the previous LLH fit as well as the number of photoelectrons measured in the event.
The filter rate of the Online Level 2 Muon Filter is about $5Hz$, reducing the Muon Filter rate by a factor of 6.

3.5.4 EHE Filter

The EHE Filter (**E**xtremely **H**igh **E**nergy Filter) ([41]) is a very simple one. It returns True for all events that have a total number of photelectrons (NPE) larger than 1000, where only non-DeepCore DOMs are considered to have a constant spacing. The purpose of this filter is to select events with an energy of $100TeV$ or more for cosmogenic neutrino search and relativistic monopole search.
The filter rate of the EHE Filter is about $1Hz$.

3.5.5 HESE / MESE

HESE (**H**igh **E**nergy **S**tarting **E**vent Selection) and MESE (**M**edium **E**nergy **S**tarting **E**vent Selection) ([42]) are designed to give selections of neutrino events who's primary interaction happened inside the detector volume (so-called starting events). In non-starting events the outer DOMs are always fired first, as they are triggered when the charged detector primary(ies) enter the detector, this most importantly also includes the muon background events. In starting events in contrast thereto the outer DOMs stay dark when the neutrino primary flies past them and the inner ones close to where the interaction happens fire first. This allows for a simple prescription to separate them apart. HESE and MESE define a veto region on the outer DOMs and then counting how many of the initial hits in an event happened within this veto region. Only if a certain threshold is not exceeded thereby, the event is accepted. The difference between HESE and MESE is merely the charge-preselection cut and the exact veto regions and other parameters they use/set in order to be sensitive to high energy events only (HESE) or medium and high energy events (MESE). The advantage of starting event filters is that they give a flavor and direction agnostic (up to fundamental effects of the detector geometry) selection of neutrino events.
The filter rate for HESE is very small at about $10mHz$, MESE has a filter rate of about $11Hz$.

3.6 Event Simulation

Simulations are an essential part of research in IceCube (as in any modern particle physics experiment). They are traditionally required to validate proposed experimental setups as well as fine tune data analysis methods, i.e. verifying that they are suitable to reliably and undoubtedly detect and identify neutrinos in the obtained data. In recent years with the uprise of artificial intelligence simulations have gain in importance even more, as they are now heavily used for training machine learning models for data analysis as it is believed that these can by far overcome the performance of the existing non-AI methods.
To get an accurate imitation of reality one naturally has to simulate every single process that happens in and around the detector from the incident cosmic particles to the final detector output. Many of the involved processes, like particle interactions or scattering of photons in the ice, are of probabilistic nature and need to be taken care of via Monte-Carlo sampling. The simulations are therefore called Monte-Carlo-simulations to distinguish them from ordinary deterministic simulations.
The entire simulation is split up into multiple individual steps as each of them needs to be performed using different programs, the resulting simulation chain can be seen in Figure 6.

[43] describes the following eight steps:
- Generation: First the stream of incoming cosmic primary particles is generated, their energy as well as arrival direction and time are thereby randomly sampled

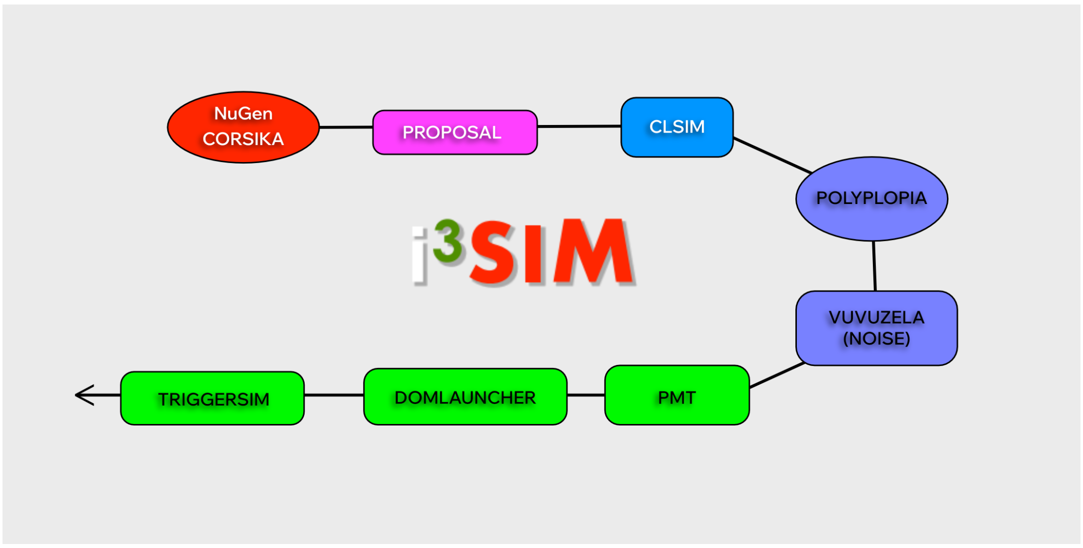

Figure 6: A schematic representation of the individual programs forming part of the simulation chain. In the first step either NuGen or CORSIKA (or another generator) is used, depending on the simulation type, the rest of the chain is always the same. (Graphic taken from [43] and modified)

from a spectrum. This spectrum is chosen flatter than the real one in order to simulate enough high energy particles for adequate statistics. This is done via so-called generators, here **NuGen** ([44]) is used for neutrinos and **Corsika** ([45]) is used for cosmic rays. NuGen then forces interactions of the primary neutrinos in the detector and propagates resulting secondary particles until they reach a specified volume around the detector. Corsika also does propagation but there is no need to force interactions, as they happen often enough by default. In the propagation stochastic loss processes of leptons are approximated as a continuous process using average losses per distance to save computational resources, as the precise losses far outside the detector won't be detected, so that only the total energy loss matters.

- Lepton propagation: The propagation close to and inside the detector is done by **PROPOSAL** ([46]), it propagates leptons accurately throughout that volume simulating individual discrete stochastic losses, this is required to get a realistic signal in the detector. For hadrons there is **CMC** ([47]), which simulates approximately the shape of their cascades.

- Photon propagation: Having simulated the particle interactions and losses **CLSIM** ([48]) is used to simulate the Cherenkov radiation emitted thereby and propagate the resulting photons through the ice until they are absorbed or hit a PMT. During the propagation lots of scattering occurs which depends heavily on the exact structure of the glacial ice.

- Background: As the atmospheric muon background flux is six orders of magnitude higher than the astrophysical neutrino one, there will statistically always be

a few muons hitting the detector in temporal coincidence with a neutrino event. This can hardly be separated into individual events, therefore making it necessary to also simulate this background. To do this **Polyplopia** ([49]) randomly samples pre-simulated atmospheric muons from a Poisson distribution and adds their photon signals to the high energy signal.
- Noise: There's also non particle induced noise in the PMTs that has to be simulated. There is noise that originates from radioactive decays, which cause fluorescence light in the glass of the DOM's pressure housing, as well as thermal noise in the PMT itself, both of these are simulated using **Vuvuzela** ([50]).
- PMT response: Once all photons that are incident on a PMT have been simulated the expected waveform they excite in the PMT is calculated. The responsible software is called **PMTResponseSimulator** ([51]), it relies on calibration measurements of the PMTs.
- Pulse reconstruction: These PMT waveforms are in reality analog signals that have to be digitized to be able to be stored and further processed. This is replicated together with all finesses of the DOM electronics through **DOMLauncher** ([52]). [53] describes in chapter 4, how the pulse reconstruction algorithm works.
- Triggering: The last step is then to run **TRIGGERSIM** ([54]), which runs the same triggers as in the real detector operation, in order to analyze the continuous datastream and identify the individual events contained therein that are saved in individual frames of the simulation's final I3 file.

After this the simulation chain is complete. The events are subsequently filtered (just like the real ones) to produce selections of them that can be used for specific applications (e.g. track filter for track reconstruction purposes).

4 Graph Neural Networks

Graph neural networks (GNNs) are a certain type of neural network that is used for neutrino detector data analysis.
They are preferred over other model architectures like CNNs because graphs are capable of expressing very complex geometries and other relations and structures of data as will be elaborated on in more detail in subsection 4.1. In IceCube DOMs are arranged in a hexagonal pattern, which cannot be expressed in a matrix without distorting it and thereby destroying the geometry. This necessitates the usage of graphs, which then also allows gives the potential for representing more abstract relations involving not only the spatial information of DOM hits but also other properties like time and charge. This high degree of customizability promises to be able to create the most powerful networks possible for our tasks in IceCube.

4.1 Graphs

Graphs are mathematical objects that consist of two components: a set of nodes and a set of edges between those nodes. [55] gives a detailed introduction to graph theory, a graphical representation of a graph can be seen in Figure 7.

Nodes are individual objects that can in essence be anything, examples for this will be given below.
Edges define relations between the nodes and can depending on the type of node have different interpretations. There are two types of edges, ones that are directed and ones that are not. Directed edges are used, when the relation between node A and node B is different from the relation between node B and node A, while undirected edges are used, when the relation in both directions is the same.
Due to the versatility of their edges graphs allow for the representation of certain types of data with complex or abstract structures that would not be possible using classical methods like vectors or matrices. An example for such a structure is the hexagonal geometry of the DOM positions in IceCubeas described above.
Figure 7 shows an everyday example for a graph: a social network. In that case the nodes represent people and store information about them like name, age, the city they live in etc.. Edges are then relationships between those people, like friendship, family, work colleagues, etc.. Here one can clearly see the difference between the two types of nodes. If node A represents a parent and node B their child, then the edge going from A to B must encode the relationship "parent to", while the edge from B to A must encode the relationship "child of", so the edges are directed. However if an edge represents two people being friends, then the relation is the same in both directions, making it an undirected edge.

Also images can be represented as graphs. Their pixels then are the nodes of the graph, storing their RGB values (in case of a colored image). Edges are drawn between adjacent pixels to be able to uniquely define the image, without the edges the pixels could be arranged any way.
One can also use graphs to represent a public transport net. Here the nodes denote the bus stops and train stations and store important information about them like position, availability of toilets and food and more. The graph edges denote the means of transport storing information like the time they leave, their type (bus or train) and things like their capacity or barriers for disabled people.
In programming nodes are usually denoted by their so-called feature vector, which stores all their information in a column vector, they are generally not vectors in the sense of fulfilling the axioms of a vector space.
Edges can be expressed in two ways: as individual edges or as the so-called adjacency matrix. Which method is preferable depends on the type of graph one is dealing with. They are divided into two types depending on their number of edges: sparse graphs and dense graphs. There is however there no exact definition, when a graph is sparse and when it is dense, it is really just a rough classification.
Sparse graphs are graphs that have rather few edges. To define their edges one uses

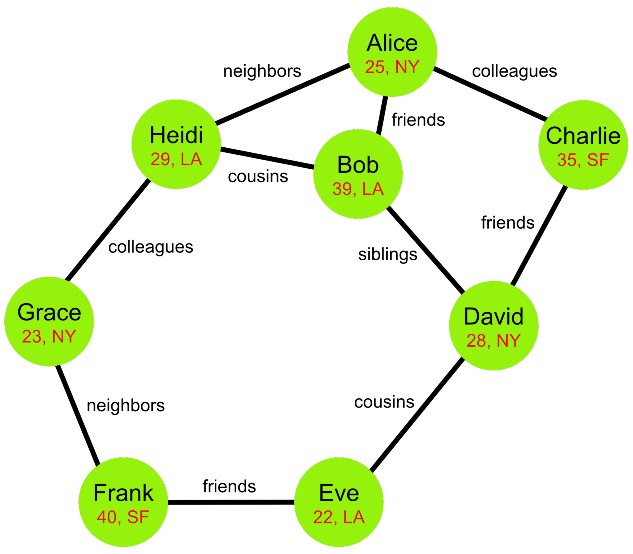

Figure 7: A popular example of a graph: a social network. The nodes represent individual people and information about them (here their name, age and city of residence); the relations between these people are represented by the graphs edges.

individual edges defined by their starting and terminating node (edge indices) and the feature vector storing possible information they represent (this is often simply a scalar value called weight, which defines the importance of that edge), so defining an edge this way requires three pieces of information. The advantage is that one does not waste memory on the vast majority of edges that do not exist (which means they have a value (weight) of 0).
Dense graphs are on the other hand graphs with lots of edges. In that case one preferably uses the adjacency matrix to define their edges. The advantage here is that one only needs the edges' weights to define them, because the position they have within the adjacency matrix already encodes which nodes they connect via the indices of the matrix elements. This way one only needs one piece of information to define an edge, but also occupies memory for non-existent (weight$= 0$) edges, which is, why the adjacency matrix is used for dense graphs, where not many edges have a weight of 0.

4.2 Layers

GNNs, just like any other type of neural network, are in essence big, highly parameterized functions. Their architecture usually consists of layers. A layer is thereby a simple parameterized function, in the easiest case a simple matrix multiplication. Applying many of these layers consecutively introduces a lot of trainable parameters and can therefore build a highly capable model. Constructing models via layers is

easy, as the layers can be pre-implemented in dedicated machine learning libraries like PyTorch in Python ([56]), while still allowing for the construction of very individual neural networks by arranging the layers in different ways.
In the following some common types of layers will be listed.

4.2.1 Linear Layers

The simplest type of layer are linear (actually affine) layers, often referred to as fully connected (FC) layers. FC layers apply an affine function to their input vector. This implies they should not be simply applied one after the other, because matrix multiplication is associative and consequently they would then merge into a single layer.

4.2.2 Activation Functions

So in between two linear layers some non-linearity is required. This is achieved through so-called activation functions.

$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \tag{3}$$

ReLU ([57]) is a very simple activation function, it sets negative inputs to zero. It has one disadvantage though: it is not strictly monotonically increasing i.e. the derivative for negative values is 0. This makes training difficult, because all negative values 'look' the same to the training algorithm; ReLU is also not differentiable at $x = 0$ (see subsubsection 4.4.3 and subsubsection 4.4.4 for what role differentiation plays in training). Activation functions that have none of these problems are tanh ([58])

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

, which ranges differentiably from -1 to 1, and Sigmoid ([59]), which ranges (also differentiably) from 0 to 1.

$$Sigmoid(x) = \frac{1}{1 + e^- x} \tag{5}$$

4.2.3 Pooling Layers

Pooling layers are non-parameterized. They simply reduce the size of the current data by aggregating certain parts of the vector or matrix. Aggregation hereby can be addition, averaging, taking the maximum or the median. This is very useful, if the model already has enough parameters to be able to fulfil the task it is intended for, but the output data format is too large and needs to be reduced without adding more parameters.

4.2.4 Graph Convolutions

The most popular and widely used layers on graphs are graph convolutive layers, they are what makes GNNs so powerful because they conserve and use the properties of graphs effectively. They are an adaption of classical convolutive layers. There are many different prescriptions on how to perform convolutions on graphs, a very simple one is the GraphConv-Layer ([60]):

$$\vec{x}_i' = \underline{\underline{W_1}}\vec{x}_i + \underline{\underline{W_2}}\sum_j e_{ji}\vec{x}_j.$$

Here $\vec{x_i}$ is the feature vector of the node i and e_{ji} the weight of the edge going from node j to node i. $\underline{\underline{W}}$ are the trainable weight matrices of the convolutive layer.
Graph convolutions can like classical convolutions be applied to inputs of arbitrary size, which is oftentimes necessary, if one deals for example with social networks as described above and there are new users registering thereby making the graph of the network larger.
Graphs and graph convolutions have however one big advantage over vector- or matrix-representations of data and convolutions: they are permutation-equivariant, which means the following:
One needs to assign some numbering scheme to the nodes and edges in order to be able to refer to them. However this numbering scheme can be applied in any way, it really does not matter. This is because the relations between the nodes are not defined by their number (so there is no special relation between nodes i and $i + 1$ just because their numbers are consecutive) but by the edges between them, which are also subject to the ordering scheme, so that they will always connect the same two nodes. This is called permutation equivariance and has to hold also for the convolutions, the output needs to be the same for all numbering schemes on the input graphs it will just also inherit the same numbering scheme (this is why it is called equivariance and not invariance). If a layer is not permutation-equivariant it is not a valid graph layer, because this would imply that two equal graphs (which have different numbering schemes) would produce non-equal results, which obviously must not be the case.
These graph convolutive layers are combined with activation functions and pooling layers (as described in subsection 4.2) to build GNNs. A comprehensive article about graph neural networks (GNNs) can also be found in [61].

4.3 Regression vs. Classification Models

There's an infinite number of completely different models that can be built, however they can be sorted into two rough classes according to the task they fulfil: regression models and classification models ([62]). These two classes are fundamentally different in what they predict and the way they output that prediction mathematically.

4.3.1 Regression Models

Regression models are all models tasked with predicting continuous real valued quantities belonging to a datapoint. Examples are forecasting the stock market or housing prices or also social media algorithms that predict the relevance of content to a user based on their profile. In neutrino physics common regression tasks are predicting the energy or arrival direction of a particle.
Regression models are very straight-forward in their mathematical implementation, the quantity that is being predicted is a real vector and consequently also the model output is a real vector of the same dimension.

4.3.2 Classification Models

Classification models on the other hand are all models tasked with predicting the belonging of a datapoint to one of a group of classes. Classical examples here are the recognition of digits, letters or objects on images or also voice recognition (the classes hereby are the spoken syllables). In neutrino physics a common classification task is the distinction between signal and background in measurements.
The mathematical implementation of classification models is a bit more complicated, this is because what they predict is not a real quantity as in regression models but a class index. For predicting this class index one cannot simply have the model output any real value and then check what index it is closest to. Instead one has the model output a probability mass function (pmf) where every entry represents the probability the model assigns for an event to be of the respective class. One can then define a cut for each class and then check whether the corresponding output probability exceeds it or not in order to obtain a classification. This can also result in the model assigning a datapoint to either no class at all or multiple depending on how the cuts are set.

4.4 Training

In order for a model to work as intended and produce useful predictions it needs to be trained to do so. Training here means tuning the model's parameters from some random initial configuration. This is done in an iterative process called deep learning, it works by feeding a training algorithm with lots of data (ideally several hundreds of thousands to millions of datapoints) where the quantity that is supposed to be predicted (called he truth) is known. The algorithm calls the model on this training data and checks the model's accuracy in reproducing the truth, it then updates the parameters in an effort to improve the accuracy (usually not the entire training data is used at once but smaller batches from it because commonly the complete dataset is too large to fit in the computer's RAM). This process (called an optimization step) is iterated as many times as required, a set of iterations in which all batches from the training dataset are used is thereby called an epoch. The number of epochs required to rain the model adequately strongly depends on the size of the data being trained on, this can range from a few for very large datasets to several hundred or thousand epochs for small datasets.

4.4.1 supervised vs. unsupervised learning

Deep learning can be divided in two categories according to how the used training data is structured ([63]).
In supervised learning the training data is so-called labeled data, this is when the truth does not form part of the data itself but is stored in he dataset as extra labels. An example for this are models that predict the species of a plant depicted on an image: the truth (the plant's species) is not part of the image but an extra label. Labeled data is often very tedious to obtain, because the labelling has to be done manually, in the previous example this means that someone has to look at every individual image and write down the plant's species, this - whilst not taking long for an individual image - accumulates to a lot of time when doing it for hundreds or thousands of images needed for training. The long preparation time for such dataset de-facto limits the size of the training dataset, such that they are often very small when compared to datasets for unsupervised learning.
In unsupervised learning on the other hand the truth forms part of the training data, this allows for much larger training datasets than in supervised learning, as the time-intensive labor of manually labeling the data is not required. An example for this are language generating models like GPT-4, that generate answers from a provided text input. These can simply be trained by feeding them with bits from existing texts and letting it predict a next word or sentence, the truth is then simply the word or sentence that actually follows in the text.
The training in this thesis was done via supervised learning, however luckily there was no need for tedious manual labelling of data, as the data used is artificially created simulation data that automatically comes with the required truth labels. Unfortunately this does not mean that no work needed to be done at all to prepare the data for training as is described in subsection 5.1.

4.4.2 Loss Function

At the heart of deep learning sits the so-called loss function. A loss function is a real-valued scalar function that assesses the accuracy of the model's predictions and has a minimum (most often 0) for when the accuracy is perfect. It is explicitly dependent on the model's predictions and their corresponding truth, however the model's predictions are dependent on its parameter configuration making the loss function implicitly dependent on it, too. This is what makes it possible to tune the parameter configuration of the model to realize accurate predictions by minimizing the loss function.

For classification models there is essentially only one loss function in use: CrossEntropyLoss ([64]):

$$\textbf{CrossEntropyLoss}(\underline{pmf}, truth) = -ln((P(truth)) \tag{6}$$

It selects the pmf's entry corresponding to the true class ($P(truth)$) and then takes the negative ln of it, this returns 0 for $P(truth) = 1$ (perfect prediction of the class) and goes to ∞ for $P(truth) \to 0$ (completely wrong prediction of the class). The name CrossEntropy stems from it's mathematical similarity to the entropy from statistical physics.

4.4.3 Backpropagation

Computationally the minimization of the loss function is implemented through an algorithm called backpropagation in conjunction with an optimizer.
Backpropagation is an algorithm to numerically calculate the gradient of a differentiable function. It is essentially a computationally efficient implementation of the multivariate chain rule for differentiation ([65]). It is called backpropagation because it starts at the last layer and works its way back from there. This backward direction is very efficient because every time a 0 appears in a partial derivative (which happens when a layer does not depend on the variable it is being differentiated with respect to) the calculation of further derivatives in that branch of the chain rule is not necessary as multiplying them to 0 yields 0 anyway. This way a quite considerable amount of calculations can be saved making the gradient calculation more efficient and therefore quicker.

4.4.4 Optimizer

The optimizer takes the gradient of the loss function and calculates an update for the model's parameters from it. In order to be able to calculate the gradient of course both the model itself and the loss function have to be differentiable. An optimizer algorithm can be as easy as subtracting a multiple (called the learning rate) of the gradient from the configuration, this is called the stochastic gradient descend optimizer (SGD) ([66]). Due to the limited size of the training batches both the loss value and it's gradient deviate statistically from their expectation values, this also means that executing the model with the same parameter configuration on different batches produces slightly different losses and gradients. For this reason the simple GD optimizer is disfavored and ones that account for the statistics as best as possible are used instead.
The most commonly used of these is called Adam. A detailed description and analysis of it can be found in [67], here's the relevant description of the algorithm taken from this paper:

$$\vec{g} = \vec{\nabla} f(\vec{\theta}) \tag{7}$$
$$\vec{m}' = \beta_1 \cdot \vec{m} + (1 - \beta_1) \cdot \vec{g} \tag{8}$$
$$\vec{v}' = \beta_2 \cdot \vec{v} + (1 - \beta_2) \cdot \vec{g^2} \tag{9}$$
$$\vec{\hat{m}}' = \frac{\vec{m}'}{1 - \beta_1^t} \tag{10}$$
$$\vec{\hat{v}}' = \frac{\vec{v}'}{1 - \beta_2^t} \tag{11}$$
$$\vec{\theta}' = \vec{\theta} - \alpha \cdot \frac{\vec{\hat{m}}'}{\sqrt{\vec{\hat{v}}'} + \vec{\epsilon}} \tag{12}$$
$$\tag{13}$$

Here θ are the model's parameters, $\vec{g}$ is the gradient of the loss function wrt. to them, $\vec{m}$ and $\vec{v}$ are the first and second running momentum averages respectively (both are initialized as 0), $\vec{\hat{m}}$ and $\vec{\hat{v}}$ their bias-corrected versions, $\vec{g^2}$ is the element-wise squared vector of the gradient. The involved parameters are: the learning rate α (it controls how big the parameter updates are), the exponential decay rates for the running momentum averages β_1 and β_2 and a small parameter vector $\vec{\epsilon}$ to ensure the term under the squareroot in the denominator in the last line is always positive (i.e. non-zero), default values for these parameters are: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Adam has two big advantages over SGD through the usage of the exponential running momenta of the gradient instead of the gradient itself. These are a weighted sums of the gradient from the current step with those from the past steps thereby significantly reducing statistical fluctuations. Secondly by taking their quotient the influence of the loss-function range is eliminated (i.e. such that using twice the loss function gives the same result as using the loss function itself).

4.4.5 Schedulers

It is usually helpful to not keep the optimizer's learning rate constant throughout the entire training cycle. At some point training often stagnates and changing the learning rate after every epoch can mitigate this. An example for this is when the learning rate is too big and therefore the parameter update overshoots the local minimum that training would optimally approach with an adequate learning rate.
There's schedulers that are fixed, meaning they change the learning rate independently of the training progress, and adaptive schedulers, meaning they react to the training progress and adjust the learning rate accordingly.
In this work an adaptive scheduler is used: ReduceLROnPlateau ([68]). It monitors the training loss and does nothing (i.e. keeps the learning rate constant) while it decreases between epochs. When it does not decrease any further however it waits a

specified number of epochs (called patience) and if at the end still no improvement has happened it reduces the the learning rate by a specified reduction factor.

4.4.6 Overfitting

A common problem faced during model training is overfitting. Overfitting is what eventually happens when a model is trained on a dataset that is too small (i.e. has too few datapoints in relation to the number of parameters of the model). As training progresses the model becomes more and more accurate to the training data and learns it's features very detailedly.
Taking IceCube as an example, events are subject to a lot of randomness in when, where and how particles react. Events caused by the exact same incoming cosmic primary can therefore look very different, however only a finite amount of them can be included in the training data. During training the model will first learn the rough characteristics of different events, e.g. that a cascade is when many DOMs in a cluster are hit and a track is when DOMs are hit in a long straight line. Importantly these are not only valid for the events in the training dataset but for all others as well. Eventually, when it is done learning those general characteristics, it will start unlearning them in favor of memorizing the exact positions, times and charges of the individual DOM hits in the training data. This is called overfitting. It is a problem, as an overfitted model will then not be able to generalize the learned features to events outside the training data, because these will have different DOM hits with different positions at different times and with different charges.
Overfitting can be mitigated by using a larger training dataset, but this is often not possible because not enough data is available. Another possibility is reducing the number of parameters of the model in order to not allow for it to be precise enough to overfit or simply stopping training when overfitting starts to appear. There is also more advanced methods to try to eliminate overfitting, some of them are described in [69]. What all of these methods have in common is the need to know when and to what extend overfitting occurs (even if it is just to know whether the overfitting mitigation worked). In order to check this a second dataset (called validation dataset) is used, which is completely disjoint (only have distinct elements) from the training dataset. In regular intervals during the training (for example after every epoch) the loss function is also evaluated on this validation dataset, through comparison with the training losses overfitting can be detected: when no overfitting is taking place the losses will be identical because the data in the validation dataset is reconstructed equally well as the data in the training dataset, when overfitting takes place though the validation data is reconstructed less well than the training data such that the validation loss will be bigger than the training loss. This can best be identified by plotting both losses, at first both losses will decrease equally, but at some point, when overfitting starts occurring, the validation loss curve slows down its decrease and eventually starts rising again. In order to obtain the best-fit model the training needs to be stopped at the lowest point in the validation loss

curve.

5 Methods

5.1 Preparing Data for Training

The models were trained using simulation data from NuGen and Corsika-in-ice simulations, those are described in subsection 3.6. The exact datasets used are listed in Table 3, they cover the total energy range from $10^2 GeV$ to $10^8 GeV$ that is relevant for IceCube. They are filtered on level 2, however the rejected events were not removed but are still included (and clearly marked as such), this way the data can be both used to train and analyze networks on trigger level and level 2, the ratio of non-l2 to l2 events is about 3:1. The simulation data files are available on the IceCube Collaboration's servers in Madison, WC. They are however stored in a custom data format called I3 (.i3) which is not a suitable format for the training of neural networks, so they had to be converted to the right data format. In this case the SQL-database format (.db) was used. This conversion proved to be very time expensive both in coding the required conversion-scripts and in the runtime of the scripts themselves. This is because the conversion is not a straightforward conversion between ordinary file formats like .png to .jpg. The .i3-files store all the raw simulation data with all details, which also makes them very large files in terms of memory consumption, so when converting them one needs to specify which information should be put into the database. That is what makes coding the conversion scripts take so long. When I started there was already a script available which transfers very basic information about the event into a database, it comes with a package called GraphNet, which was also used for the training of the models themselves, a description of it is given in subsection 5.2. The script calls so-called extractors, which are precisely the functions that tell the converter which information from the simulation to put in the database. There are two extractors: a feature extractor and a truth extractor, the output of each of them is saved in a separate table within the database: the pulse table and the truth table respectively. The feature extractor extracts the detector response to the event in form of the InIceDSTPulses, which are the individual photon hits to the DOMs as reconstructed with the DST pulse unfolding algorithm described in [53], chapter 4. The saved pulse features are most importantly the DOM position, the hit time and the charge deposited in the PMT. The truth extractor extracts the truth information of the event: this is in the basic script only the cosmic primary's energy and direction and a few other labels. This is unfortunately insufficient for the specific ML applications me and a colleague of mine were going to apply them to. We needed more truth information like the event classification in my case or the energy lost in cascades and tracks in the case of my colleague. For calculating these a script already existed, it can be found in [70], so my work could have been as easy as simply modifying the existing script to function

as a truth extractor. However I set out to improve the script and add a bunch of new labels and a bunch of new classifications (increasing the total number of possible classifications from 22 to 34) all in an effort to describe the event as precisely as possible within the databases so that they could be used by anyone in the collaboration no matter the specific applications they have in mind. This was very successful, a list of all truth labels included in the final databases as well as a list of all possible event classifications can be found in Table 2 and Table 5 respectively, the complete documentation of the scripts can be found in [71] in Madison, if you have access to the Madison servers. However during the coding a bunch of seemingly unrelated challenges arose, like finding the closest approach of a particle on it's trajectory to the detector. This is why in the end it took me more than a month to finish the scripts.
When the data processing was started it was quickly realized that the scripts' runtime was very long, a large number of jobs were even stopped because they exceeded the $48h$ runtime limit on the Madison servers. This is because some employed functions need to be implemented in C++ in the future, currently they run in Python, which is comparably slow. Particularly calculating the number of photoelectrons (mcpe) caused in the detector by signal and background particles took a very long time, therefore the scripts were accelerated by removing these two labels, the average runtime was thereby reduced by a factor of more than 5. Unfortunately that only helped a little bit, because many jobs were still exceeding the runtime limit. Precisely it was the Corsika files running for so long, the NuGen on the other hand were running a lot quicker. To quantify this: a NuGen file took on average about $2min$ while the Corsika files very often exceeded the servers runtime limit of $48h$. There's at least two reasons for this: firstly the Corsika events contain muons bundles from air showers, these bundles often contain lots of muons, that together with the losses in their tracks lead to much larger particle numbers than in NuGen events. This according to my estimate makes the processing of Corsika approximately 20 times slower. The other reason that makes it take longer is that a Corsika file is on average 3.2 times larger than a NuGen file (272MB vs 86MB). Combining both factors suggests that a Corsika file should on average be processed after about $2h$ and therefore not nearly exceed the runtime limit. This means that either my estimate of the slowing due to larger particle numbers is way off or there is another effect that can explain why Corsika files so often exceed $48h$ processing time. Either way the Corsika processing was very inefficient because the output databases were only created at the very end of the processing in order to prevent saving potentially corrupted files that would cause problems during training. This however also means that all jobs that were left unfinished due to the runtime exceeding did not produce any output, therefore wasting computing resources. This together with the at times low availability of cluster nodes made $150GB$ of Corsika files take about three weeks to process, while $300GB$ of NuGen were done after about an hour (when the availability of nodes was good).

The consequence of the long Corsika processing time was that not all I3-files of the datasets could be processed, but only rather few of them, however the total training databases in the end were still over one Terabyte in size and therefore definitely large enough. A list of the individual filesizes and numbers of events for the final databases can be found in Table 4, the databases themselves ly in Madison in [72].

5.1.1 Event Classification

Event classification is the process of assigning an event to one of a set of discrete event classes, thereby eliminating the infinite number of degrees of freedom of the event. A list of all classifications and the number of samples of each class in the final databases is given in Table 5.
An essential part of event classification is the I3MCTree given for the event in the I3 file ([73]). This mctree is structured like a family tree (where every person is instead a particle), it describes which particles were created through interactions or decays from the other particles. Due to the analogy to a family tree one speaks of parent, children, grandchildren and so on to denote particle relations to one another and in further analogy to people the creation of a particle is denoted as birth while the decay of a particle is called its death. The mctree stores all information needed about a particle: it's birth position, direction, length between birth and death (so that one can calculate the death position), energy and of course the particle type. Also a particle is called signal particle if it descends from the signal primary and background primary if it descends from a background primary. One thing to consider is the case of muons, conventionally one thinks about a track as a single muon which does energy losses on its way, however in the mctree a track is stored as child of the muon, storing all energy losses as well as checkpoint muons at random positions on the track to also keep track of the continuous energy loss due to ionization. To stick with the common interpretation the checkpoint muons as well as the energy losses are deleted from the mctree, in order to still keep track of the muons energy along the track a track object class was implemented that saves the energy losses and checkpoints. Another important part of classification is to have a measure of containment, in other words on needs to know, weather a particle is inside or outside the detector. For this a convex hull around the detector is defined, which runs 50m outside of the outermost DOMs, the distance of 50m is an arbitrary choice but seems reasonable. The hull is chosen to be convex despite the one seemingly non convex corner of the detector, because there are at most two intersections of a line with it thus making calculations easier. In order to classify an event one starts out finding the detector primaries or simply primaries, those are signal particles that intersect the detector with the exception of neutrinos which only count when they interact within the detector because otherwise they are invisible. Afterwards three cases are differentiated: events with no, multiple and one detector primary.

5.1.2 events with no primaries

The case where there is no detector primary is only distinguished into a few cases as the event is happening outside of the detector and the detector response therefore is rather weak and only a few DOMs are triggered. One finds the brightest signal particle to the detector and then classifies according to that, if there is no brightest signal particle because all of them are completely dark the event is classified as 'background' and if something unexpected happens like a nucleus (which cascades far away from the detector in the atmosphere) being the brightest particle the event is classified as 'other_uncontained'.

5.1.3 events with multiple primaries

When there is multiple detector primaries there's only two rough classes being differentiated: 'bundle' and 'other_multiple'. A 'bundle' is the result of an energetic hadronic cascade where through pi production and consequent decay lots of muons are created that then intersect the detector, this is one of two causally connected event with multiple primaries one expects (causally connected hereby means that the multiple primaries share a common source and are not completely unrelated particles that just happen to enter the detector simultaneously). The other being a track/spur doing a muon pair production process and then entering together with one or both of these muons, this however happens so infrequently that it is being classified together with any unexpected event as 'other_multiple'. In a future update to the classification script muon pair production events might get their own class despite of their rareness.

5.1.4 events with one primary

Events with exactly one detector primary are the most interesting ones as they cause a strong detector response and a lot of DOMs to fire (depending on the energy obviously). They therefore allow for the most accurate reconstruction of the event of all of the three event cases (bundles, though also potentially causing a strong detector signal don't tell a lot about the particle that caused the cascade because the hadronic cascades they originate from don't look different for different particle types that cause them). For this reason these events are split up into a lot of classes in order to not loose accuracy. The algorithm traces the entire particle sequence traversing the detector, starting with the detector primary and ending with a cascade at the end of the sequence or a particle leaving the detector. These particle sequences then each correspond to a class, unexpected sequences (for example ones that are physically not possible and must therefore be caused by simulation errors) are again classified as 'other'. A list of all classifications with explanation of their corresponding particle sequence can be found in Table 5.

5.1.5 Simweights

In order to produce more high energy events in the simulations one does not use the actual neutrino- or CR-spectra from measurements but less steep one like $\propto E^{-1}$. For analysis one however wants realistic event rates to get accurate results in histograms for example. This is where simweights comes into play, it is a Python package which assigns a so-called simweight to every event ([74]). This simweight is a frequency in Hz which corrects for the false assumed spectrum in the simulation as well as for any modified physics (i.e. interaction crosee sections) made during simulation, it can then be used as a weight in histograms to reproduce the physically expected spectral event rates. In order to calculate it one has to supply simweight with the true spectrum. For Corsika the Geisser-Hillas-5-component-spectrum ([75]) that comes with SimWeights was used. For NuGen it is not quite so simple because both the cosmic neutrino flux and the atmospheric neutrino flux (which is zenith dependent) that is created in CR showers have to be considered. The cosmic flux is rather straight forward: the single power law fit from [76] is used, as it is only for muon neutrinos the assumption that the flux for all flavours is equal (flavor ratio 1:1:1) was made, this is what is theoretically expected ([77]). The atmospheric flux on the other hand cannot be described so simply because it is zenith dependent. Conveniently there exist 2d spline fits for them that could be used, they extrapolate the spectrum between a set of known points that were obtained using air shower simulations, these splines can be found on the Madison servers in [78].
When the databases were finished it was realized, that there was a problem with the simweights as for the included filters the rates were by a factor of 3 to low. As there was no sign of a mistake in the application of the simweights library I had no idea where the problem originates from. It was therefore fudged by simply multiplying all Corsika simweights by the missing factor of 3, the NuGen ones seemed fine.

5.2 GraphNeT

GraphNet is a machine learning package for Python that is used in this project, its documenation can be found in [79]. It is an implementation of PyTorch, Pytorch Lightning and Pytorch Geometric ([56],[80],[81]) specifically designed for applying Graph Neural Networks (GraphNets) to IceCube, KM3Net et al.. It comes with four submodules: graphnet.data, graphnet.models, graphnet.training and graphnet.deployment ([79]).

5.2.1 graphnet.data

graphnet.data comes with a bunch of handy classes and functions for reading and writing data and thus converting between different data formats,one of those classes is for example the i3-to-sql-converter used in the conversion script described in subsection 5.1. graphnet.data also comes with the Dataloader class that can read in batches of events from SQL databases and provide the batches to the training algorithm.

5.2.2 graphnet.models

graphnet.models lets you define the neural networks themselves, they consist of two parts: the backbone and the tasks. The backbone is the actual big network with nearly all the layers and by far the majority of the parameters, it is usually defined with a large output dimension to allow for maximum customizability with the tasks. The tasks then transform the backbone output into the desired output of the model, this is what makes the model specific to one's application. The genius thing is that the backbone, which is the part that takes the most time upon development, on it's own can be used for any application, saving a lot of development time. A task could for example be taking an L-norm of the output to ensure positive predictions for energy reconstruction or exponentiating and normalizing the output to obtain a probability mass function for classification.

5.2.3 graphnet.training

graphnet.training comes with everything one needs to train the models, this includes loss functions like CrossEntropyLoss or MeanSquaredLoss, optimizers like StochasticGradientDescend or Adam, learning-rate schedulers and more. The base classes for the above mentioned are straight forward to understand, so coding custom classes is easily doable.

5.2.4 graphnet.deployment

graphnet.deployment is useful once one has a finished trained model, it combines loading the model and applying it to a set of data.

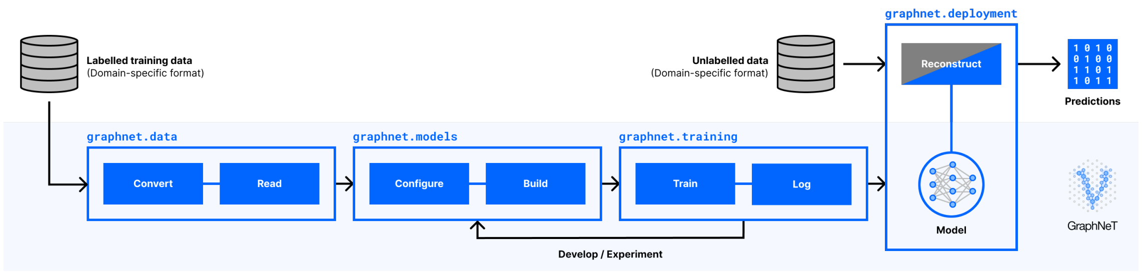

Figure 8: High-level overview of a typical workflow using GraphNeT ([82])

5.3 Model Setup

5.3.1 Grouping

In order to build filter networks for IceCube as described in section 1 the desired filtering characteristic has to first be defined. A simple but effective way to do so is via a so-called grouping scheme: a surjective mapping which maps each event via it's classification (see subsubsection 5.1.1) to one of a set of groups that it should be recognized as by the filter. The classical case of a boolean filter can thereby replicated by choosing a mapping with only two groups like "filter_condition_satisfied" and "filter_condition_not_satisfied", however the advantage of the approach taken here compared to the classical boolean filters is that one theoretically can use an arbitrary number of groups and thereby make finer predictions about an event.
Realizing such a filter as a neural network is a standard classification task, one defines a model which outputs a probability mass function to predict to which of the groups an event belongs to.
In an ideal world one train a filter with a trivial mapping between classifications and groups, so it would be able to distinguish all 34 classifications there are. This can certainly be tried out, however it does not seem any promising at all, because many event classes look virtually identical in the detector and can therefore hardly be distinguished. For example hadronic and electromagnetic cascades essentially only differ in their light yield, however if one does not know the total deposited energy this can not be distinguished. Another example is a lollipop event, this is virtually indistinguishable from a hadr_cascade or an em_hadr_cascade event because spurs are hardly ever visible due to the low interaction cross sections of the tau doing energy losses, so that one effectively only sees the cascade from the tau decay.
Because an ideal filter will not work a non-trivial grouping scheme needs to be used, the goal thereby is to find groups that are as rough as they need to be so that a well-trained filter network can distinguish them easily enough to produce reliable results whilst being fine enough to be able to draw useful conclusions from knowing the belonging of events to them.
Several filter models with different grouping schemes have been trained and tested, the results of all of which will be elaborated on in section 6.

5.3.2 Graph Definition

Before defining a graph neural network one first specify how the input data should be represented as a graph that can then get fed into such a network. As described in subsection 4.1 every graph consists of nodes and edges so both of these have to be defined.

Let's start with the nodes. These represent the DOM hits, so trivially there could be a node for every individual hit where the nodes are mathematically presented by a vector which has a component for every feature registered for the hit like the position, time and charge. This is absolutely possible but has one drawback: the number of nodes is equal to the number of DOM hits, as described in subsubsection 4.2.4 this is mathematically not a problem, however for very highly energetic events the number of DOM hits gets very large and therefore also the graph making computation slow or even running into RAM problems. To prevent this the node definition used here defines the nodes in a way that every one of them represents a DOM by calculating representative quantities from all hits of the same DOM, this technique is known as summary statistics. The exact summary statistic used are percentiles ([83]). The kth percentile ($P_k(values)$) of a set of values is the biggest value in the set for which at most $k\%$ of the values have a value smaller than it. The nodes are then defined as:

$$\text{node(DOM)} = (x, y, z, rde, hlc, (P_k(t)), (P_k(q)), N)^T \tag{14}$$

This definition gives the nodes 28 features each:
- x, y and z are the coordinates of the DOM,
- rde is the DOM's relative quantum efficiency (this is 1 for the standard DOMs and 1.35 for the DeepCore DOMs),
- hlc (hard local coinidence) is a quantity from the data acquisition (DAQ) in IceCube, very roughly it describes the quality of the reconstruction of the pulses from the measured PMT waveforms (see [22] for further information on the DAQ in IceCube),
- N is the number of hits of the DOM,
- $(P_k(t))$ are all percentiles of the hit times and
- $(P_k(q))$ are all percentiles of the reconstructed PMT-charges.

These quantities are not fed to the network as they are, instead they are rescaled to manageable intervals (like [-1,1]) to prevent exploding values throughout the model because of too large inputs, the following rescaling prescriptions were used:

$$(x, y, z)^T \rightarrow \frac{(x, y, z)^T}{500m} \tag{15}$$

$$rde \rightarrow \frac{20}{7} \cdot (rde - 1) \tag{16}$$

$$hlc \rightarrow hlc \tag{17}$$

$$t \rightarrow \frac{t}{10^6 ns} \tag{18}$$

$$q \rightarrow log(q) \tag{19}$$

$$N \rightarrow log(N) \tag{20}$$

The percentiles were chosen as $k \in [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$. This choice was made to give adequate summaries of the time and charge distributions, that are well spread out over the entire range from P_0 (the smallest value of a feature) to P_{100} (the largest). Summary statistics has its downsides, too: first of all by summarizing the input data one inadvertently looses some information, to keep enough information (what is enough can be estimated intuitively but ultimately needs to be figured out via trial and error) the statistics needs to be big enough, in this case this means enough percentiles need to be used. This directly leads to the second downside: for DOMs with a small number of hits there will be more percentiles than hits, which means the node will be larger than the individual nodes combined if one didn't use percentiles. So by using summary statistics one accepts making some nodes a little larger than they need to be in order to make others a lot smaller than they would otherwise be.

The edge definition is very simple, it follows the intuition that nodes that are geometrically close have a larger causal connection than distant ones. This intuition is realized by the KNN edge definition (KNN is short for k nearest neighbours) ([84]), which draws edges with weight 1 between a node and its k nearest euclidean neighbours. KNN graphs are a kind of sparse graph (at least when the number of nodes is much larger than the number of nearest neighbors, which is the case in the application here). A sparse graph should optimally be chosen because PyTorch (which GraphNeT is based on) uses the edge index formalism described in subsection 4.1 to represent graphs, which is best suited for sparse ones. It was chosen to use $k = 8$ nearest neighbours as this is the default value in GraphNeT and there was no reason to change this. One thing to note here is, that the geometry of IceCube is such that the horizontal distance of DOMs is $125m$ and thus much larger than the vertical distance which is $17m$ (for DeepCore both of these are smaller). Due to this geometry the 8 nearest neighbours all ly on the same string as the DOM itself (except for when the DOM is at the very bottom or top of the detector), therefore the edges will create subgraphs of the strings which are almost completely unconnected with each other. This is not a problem though, because the edges are redrawn regularly during execution of the model on the graph (as is described in subsubsection 5.3.3), so that information is still allowed to flow between the individual strings, just not in the first layer.

5.3.3 Backbone

The backbone used was a modified version of a pre-implemented model from GraphNet: DynEdge ([85]). The model was designed specifically for IceCube data analysis ([86]) and is therefore ideally suited for this thesis project. The fact that it comes with GraphNet by default saved a lot of work and time, which was a spare resource after the data processing took so long as described in subsection 5.1. Luckily the implementation comes with lots of arguments for maximal customizability and thus does not restrict the possibility to create optimal models for my specific applications at all, in one aspect the source-code had to be modified though as is described in the following summary of the architecture closely following the one described in [86] and the source code ([85]).

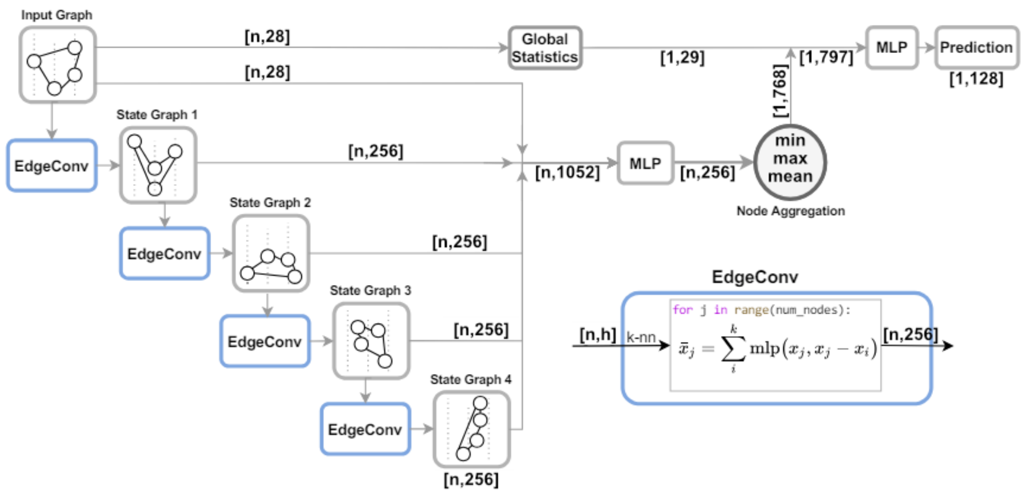

Figure 9: Schematic representation of the DynEdge model architecture taken from [86]. The graphic was modified according to the modifications done to DynEdge in this thesis. n is the number of nodes (the number of DOMs that were triggered in the event).

The fundamental building block is the EdgeConv ([87]) convolution layer, this is can be seen in the bottom right corner of Figure 9 together with the entire DynEdge architecture. It applies an MLP (multi-layer-perceptron) on a stacked vector of the current node vector and its difference with a neighboring node (one that is connected to it via an edge) and sums this over all neighbors. As activation function for the MLP ReLU is used. The important feature of DynEdge is that every time it applies an EdgeConv layer it first redraws the graphs edges according to the KNN-edge prescription described in subsubsection 5.3.2 (except for the first application, there the edges of the input graph are used as they are as otherwise the information encoded therein would be lost), this is where its name comes from: **Dyn**amic **Edge**s. The advantage of the dynamic edges according to the paper is that this allows DynEdge to learn not only features between nodes that are connected but also the optimal connections themselves.
The EdgeConv layer is applied four times to the input graph, this is the optimum according to the paper, as increasing the number of convolutional layers only increases the training time but not the performance of the model. The graphs after every convolution are then merged together with the input graph into a single one by dropping their edges (they are not needed anymore in the further steps) and stacking their respective node vectors.
The stacked nodes are then sent through a node-wise MLP and the resulting unconnected nodes are then pooled element-wise into a single vector (so the corresponding elements from each node are aggregated), this is done using three aggregation meth-

ods: minimum, maximum and mean (in the paper a fourth method: summation was used, however for high energy events with lots of nodes using summation is not recommended, because the sums can get very large ([88])). The vectors from all pooling methods are then again stacked into a larger vector.
In a second branch of DynEdge some global statistics are calculated from the input graph, this is were modifications were done to the default model implementation. In the original model the so-called homophily ratio of each of the node features is calculated, the homophily ratio of a node vector component is the ratio of node pairs where the respective node vector components are equal. In the paper this describes how the individual pulses are spread over different DOMs, however as described in subsubsection 5.3.2 in the models here all hits of the same DOM are expressed in one single node using percentiles, therefore the homophily ratio is trivially zero and therefore useless; it was therefore removed from the model. The remaining statistics are the means of each input node component (so the average position, time, etc. of the event) and a quantity representing the number of nodes of the input graph, in the usage in the paper this was the decadal logarithm of the number of pulses, here it was modified to the relative number of DOMs that were triggered in the event. The vector containing all statistics is then stacked onto the final vector from DynEdge's first branch.
The resulting large vector is then passed through a final MLP which produces the backbone output that is passed to the task. The backbone has a total of $1.4\dot{1}0^6$ parameters in its implementation here, the exact layer sizes can be found in a schematic representation of the modified DynEdge architecture in Figure 9.

5.3.4 Task

The task that transforms the backbone's output into a pmf for the prediction of the group looks as follows:

step 1:

$$\underline{x} = \underline{\underline{A}} \cdot \underline{bb} \tag{21}$$

This step takes the backbone's output $\underline{bb}$ and transforms it via the matrix $\underline{\underline{A}}$ from the backbone output dimension into the required dimension. The components of $\underline{\underline{A}}$ are thereby additional parameters of the model, their number is however absolutely insignificant compared to those of the backbone. This step is performed automatically by GraphNeT, one only needs to specify the required dimension, which is the number of groups in this case.

step 2:

$$\underline{x}' = limit(\underline{x}, 80) \tag{22}$$

This step applies an element-wise limiting function on $\underline{x}$ to produce $\underline{x}'$ who's elements are limited to the interval $[-80, 80]$, this is important so that in the next step no NaNs appear due to numerical issues of the calculation in Python. The limiting function was chosen as:

$$limit(y, L) = L \cdot tanh(\frac{y}{L}) \tag{23}$$

step 3:

$$\underline{\text{pmf}} = \frac{e^{\underline{x}'}}{\sum e^{\underline{x}'}} \tag{24}$$

This final step produces the pmf, it does so by exponentiating $\underline{x}'$ elementwise to ensure positivity and then normalizing. This function is known as Softmax and is the standard approach for producing pmfs for classification tasks in machine learning.

5.4 Training Setup

5.4.1 Loss Function

As loss function CrossEntropyLoss as described in subsubsection 4.4.2 is used.

5.4.2 Optimizer

The Adam Optimizer is used, a description of which can be found in subsubsection 4.4.4. The default parameters were kept at their default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Only the learning rate α was changed to $\alpha = 2 \cdot 10^{-5}$ as the default value $\alpha = 10^{-3}$ kept causing NaNs during training.
Adam was fed with batches of 8 events at a time, however an optimization step was executed only every 12 batches, so that the gradients from all of them could be accumulated and added in order to reduce the statistical fluctuations of the gradients before feeding them into Adam. This is despite the fact that it is designed to deal with those pretty well, but further reducing statistical uncertainties is always helpful.

5.4.3 Scheduler

To control the learning rate the ReduceLRonPlateau scheduler is used, it is explained in subsubsection 4.4.5. The patience was thereby set to 1 epoch and for the reduction factor a value of $\frac{1}{5}$ was chosen. For an unknown reason the scheduler unfortunately did not work as is explained in subsection 6.1.

5.4.4 Anti-overfitting

To deal with overfitting a method called early-stopping is used. It works by simply terminating the training if the validation loss curve has not decreased further for a certain number of epochs. As the used datasets are very large and thus epochs take a long time, the early-stopping patience was set to only 2 epochs.

5.5 Computing Setup

The GNN training was performed on FAU's RRZE computing infrastructure, which runs on the Slurm batch system with a default job runtime of maximum 24h. The 'woody' cluster ([89]) was used for CPU only jobs, it offers in total 286 nodes with Intel CPUs of different generations, the availability of the nodes is generally very good. The 'tinygpu' cluster ([90]) was used for jobs requiring GPUs, it offers different kinds of GPUs, the one that were used are the Nvidia A100 of which there are 8 nodes with 4 of them each, their availability was usually very bad as they are very performant and thus very popular. The second choice after the A100 was the Nvidia Geforce RTX3080 of which there are 7 nodes with 8 of them each. They perform a bit worse than the A100, however their availability was generally very good and one could therefore use more of them in parallel compensating for the worse performance per GPU a bit.
The training databases were initially stored on the mass storage server 'wecapstor3' which meant that during training batches had to constantly be loaded from there to the GPU RAM. As was made aware of by the server admins the bandwidth between these is quite low meaning that the GPUs had to wait for the read-in of batches to process for the longest time during training making it very slow and inefficient. Also the constant high load of about 30MB/s that was caused on the limited bandwidth was very unsolidaric towards other users as this further limited the bandwidth available to them. To fix this issue it was advised to copy the relevant data to the node's temporary directory (TMPDIR) that has a high bandwidth and low latency to the GPUs and is reserved separately for every job. The transfer rate for copying to the TMPDIR was at maximum $30MB/s$, however that rate was at times subject to extreme downward fluctuations as low as $10MB/s$ with differing read load from other users. All databases together have a total size of $1.5TB$ so copying them entirely would have taken $14h$ or more. As not all events from all databases were used for training (more on this in subsection 6.1) the copy time could be reduced significantly by creating copies of the original databases which only contain the events that were actually being trained on and copying those to the TMPDIR. Also the copying itself was parallelized which sped it up almost by a factor of 5 compared to single process copying, in the end it took on the order of an hour until from the start of a job until training could commence (when the wecapstor3 was working properly and not overloaded). All time and effort taht went into implementing the staging of the data to the TMPDIR was definitely worth it as

it decreased the required training time per epoch by a factor of 3 to 4 and therefore saved insane amounts of time overall.

5.6 Analysis Methods

5.6.1 True Positive Rate and False Positive Rate

Analysis of classifiers knows two important fundamental quantities, out of which (almost) everything else is calculated: the true positive rate and the false positive rate per group of the prediction.
The true positive rate (*tpr*) of a group is the rate at which the classifier predicts the group when the event actually belongs to that group and
the false positive rate (fpr) of a group is the rate at which the classifier predicts the group when the event actually belongs to a different group.
'Rate' thereby refers to the actual physical frequency with which corresponding events happen or pass a filter depending on context.
In order to calculate *tpr* and fpr it needs to first be defined what group the classifier predicts in terms of its output pmf. For this a cut for each group is required. A group is then predicted if the corresponding probability is larger than the corresponding cut. As this is done independently for every group all of them can independently be predicted or not, therefore it can also happen, that the classifier predicts multiple groups at the same time or no group at all, in these cases the model is inconclusive and offers not interpretation of the event (except for that it is hard to classify).

5.6.2 ROC Curve

An easy but powerful visualization of the model's discrimination performance is plotting fpr vs. *tpr* in dependence of the cut ranging from 0 to 1, the so-called ROC curve (**r**eceiver **o**perator **c**haracteristic curve) ([91]). For a cut of 0 all events are classified as 'group', so $tpr = tr$ and $fpr = fr$ are respectively equal the total rate of 'group' events and the total rate of non-'group' events. For a cut of 1 both $tpr = 0$ and $fpr = 0$, both of these edge cases are independent of the model performance. Every ROC curve therefore connects these two endpoints, for a completely random classifier, that arbitrarily guesses the group of an event, the ROC curve will simply be the diagonal between them. Generally the better a model's discrimination ability is the more the ROC curve will deviate upwards from this diagonal, allowing to read off the performance from it. For a perfect model, that classifies every single event correctly with 100% confidence, the ROC curve will run along the top and left edge of the plot.

5.6.3 AUC

A measure of discrimination performance derived from the ROC cure, which breaks it down in one scalar value, is its AUC (**a**rea **u**nder the **c**urve). It makes use of the property that better perfomance corresponds to a higher ROC curve and therefore also a larger AUC, it ranges from 0.5 for a randomly guessing classifier to 1 for a perfect one.

5.6.4 Purity and Efficiency

When it comes to choosing the best suited cut for each group of the classifier there is no ultimate right or wrong, depending on the exact application different choices might be reasonable. Two aspects are most important for choosing them: purity and efficiency.
The purity of a group selection is the ratio of the rate of actual 'group' events to the total rate of events in the selection. It quantifies by how much the selection of 'group' events returned by the classifier is contaminated by false positives [92].
The efficiency of a group selection is the ratio of the rate of actual 'group' events in the selection to the total rate of 'group' events in the original dataset. It quantifies what fraction of all 'group' events from the original dataset is included in the corresponding selection returned by the classifier [92].

$$\text{purity} = \frac{tpr}{tpr + fpr} \tag{25}$$

$$\text{efficiency} = \frac{tpr}{tr} \tag{26}$$

, where tr is the total rate of 'group' events. There is some different nomenclature between physics and classical machine learning, that's why in the provided sources purity is called precision and efficiency is called recoil or sensitivity.

5.6.5 F-Measure

In order to not have to define the cuts manually one generally defines a metric in terms of purity and efficiency that adequately represents their importance for the desired application and then chooses the cut such that that metric is maximized. A popular such metric is the f-measure [92]:

$$f_\beta = \frac{(1+\beta^2) \cdot \text{purity} \cdot \text{efficiency}}{\beta^2 \cdot \text{purity} + \text{efficiency}} \tag{27}$$

It is a generalization of the harmonic mean (the case where $\beta = 1$. Choosing different values for β allows for different grades of prioritization of either purity or efficiency ranging from $f_0 = \text{purity}$ to $f_\infty = \text{efficiency}$.
It has been noticed that due to the vastly different rates of events from different groups ($1 : 10^6$ neutrinos to background) by using the default f-measure as defined above useless results were obtained. In order to fix this the definition used here was

modified to replace the purity with a quantity I refer to as relative purity. This is essentially the purity as calculated using the relative *tpr* and *fpr* (true and false efficiencies) to eliminate the huge difference in absolute rates. Its definition is as follows:

$$\text{rel\_purity} = \frac{\frac{tpr}{tr}}{\frac{tpr}{tr} + \frac{fpr}{fr}} \tag{28}$$

, where fr is the total rate of non-'group' events and the rest of the quantities is as in subsubsection 5.6.4.

5.6.6 Confusion Matrix

After choosing the optimal cuts for the classifier one can plot a detailed analysis of which events get classified as what: the confusion matrix. It is a square matrix of dimension N_{groups} x N_{groups} where every entry lists the rate of events that belong to the corresponding group on the vertical axis and were classified as the corresponding group on the horizontal axis [91]. For multi-group classification, where there are more than two groups it has a big advantage over the ROC curve, which is that it not only distinguishes between correctly and falsely classified events but shows in detail to what extends events from certain groups get confused (therefore the name) with events from other groups. This can give indications of groups that are difficult to distinguish from one another and where one should mistrust the classification done by the model.

6 Results

In this thesis two models have been designed and trained as described in section 1: StartingEventFilter, which is analogous to the existing HESE and MESE filters, and DoubleBangFilter, for which no analogous 'standard' filter exists. Their names are chosen to be simple yet effective at describing what they do.
For the analysis of their performance the analysis methods from subsection 5.6 have been employed.
To check their robustness to the measurement uncertainties of IceCube, which are $1.2ns$ time uncertainty and 10% charge uncertainty according to [22], the analysis has in parts both been done to the models' results obtained from the unchanged training data and to those obtained from the same data but randomly perturbed according to these uncertainties.

6.1 Starting Event Filter

StartingEventFilter was designed to classify all events where a neutrino primary enters the detector and interacts inside it as 'starting_event' and all other events as 'other'. This is analogous to the existing HESE and MESE filters (see subsubsection 3.5.5), however the advantage is that this GNN version does not rely on a veto region and can therefore use the entire detector volume as a valid region for the primary neutrino interaction, this way it can theoretically also recognize starting events close to the detector walls.

As it is intended as an online filter, which need to deal with the largest amount of background contamination of any filter level, it is important to keep a good balance between signal and background in the training data. The choice was therefore made to use a 1:1 ratio selection of NuGen and Corsika events from the databases described in subsection 5.1 for training. As there are only $4.17 \cdot 10^6$ total Corsika events available the NuGen selection had to be chosen equally small, $4.2 \cdot 10^6$ events, evenly distributed over all three flavors, were used, only 10% of all available ones. This makes for a total of $8.37 \cdot 10^6$ events in the selection, but only 90% of them could actually be trained on, as the other 10% were required for validation. Ultimately the training was still performed on an amount of events that is way above the recommended margin of one million ([93]).

The training was carried out for a total of 25 epochs, each of which took about 5:30 hours including validation on the four used Nvidia RTX 3080 GPUs. The best-fit was achieved at epoch 22, after it the validation loss increased again due to overfitting, which led to the termination of the training after 2 epochs. The scheduler peculiarly did not decrease the learning rate before that as it should have according to the lower chosen patience of 1 epoch, this indicates a bug in the code that needs to be investigated. Over the course of the training the validation loss decreased by 39.8% from 0.201 after the first epoch to its minimum of 0.121 at the end of epoch 22. Towards the end of the training a few peculiar errors started appearing, these were a memory error, a NaN error as well as a runtime error. Given that these appeared randomly and were different every time it seems reasonable to assume that they were caused by issues with the hard- and/or software of the servers and are not a systematic error with either GraphNeT or my training scripts. Either way they were simply ignored and the training was restarted after an error occurred which worked well.

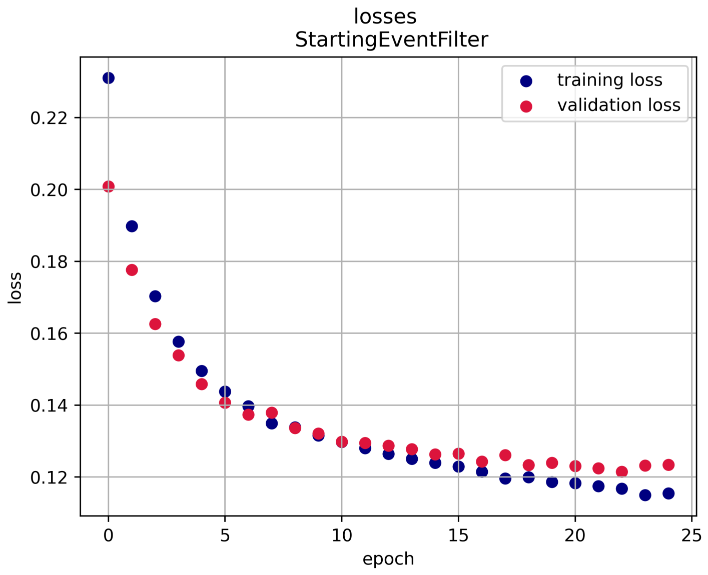

Figure 10: Training and validation loss plotted over the epochs, first signs of imminent overfitting can be seen from epoch 15 onward when the validation loss starts oscillating and no longer decreases continuously. The best-fit state is achieved at epoch 22, after which the actual overfitting starts and the validation loss starts increasing again from the minimum value of 0.121.

Figure 11 shows the ROC curves obtained from StartingEventFilter for the default validation data as well as for the same data that was randomly perturbed to simulate the measurement uncertainties in IceCube. Very importantly one can see the model's robustness to the measurement uncertainties as the perturbed curve is not even visible as it is so similar to the default one that it is completely covered up by it in the plot. The AUC of both curves is 0.962, there is unfortunately no value to compare it against so that no conclusions can be drawn from it. Figure 11 also shows the filter state at the optimal cut obtained from an f_β-optimization as described in subsubsection 5.6.5, $\beta = 0.15$ was chosen in order to obtain a total filter rate similar to that of MESE ($11Hz$) in order to be able to compare the two properly.
The optimization returned a cut of 0.6715 for the 'starting_event' group and 0.995 for the 'other' group. The resulting true and false positive rates for both groups can be seen in Figure 12, which shows the confusion matrix.

It can be seen that StartingEventFilter achieves a true positive rate of $9.62mHz$ and a false positive rate of $8.68Hz$ for starting events. The total filter rate is consequentially $8.69Hz$. This yields a purity of $1.11 \cdot 10^{-3}$ at an efficiency of 0.630. The purity of the initial trigger level data is $7.75 \cdot 10^{-6}$, in comparison that means the model was able to purify the sample by more than 2 orders of magnitude (a factor

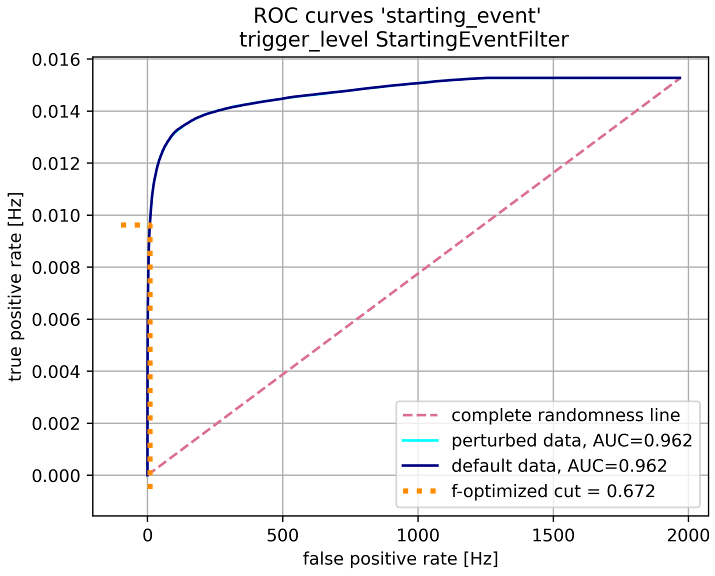

Figure 11: The ROC curve for the 'starting_event' group of StartingEventFilter for both the default and the perturbed validation data, no visible difference can be made out between them, demonstrating that the model is robust to IceCube's measurement uncertainties. The orange dotted line marks the true and false positive rates at the cut suggested by an $f_{0.15}$-optimization, which is 0.6715.

of 143).

According to the above StartingEventFilter seemingly shows very good performance, however it needs to be compared to the existing and to be replaced HESE and MESE filters to confirm this. Table 1 lists all the important quantities of the three side by side in tabular form. Unfortunately except for the total filter rate no information was available from the internal filter document [42] for HESE and MESE, the information therefore had to be estimated using the filter labels available in the training data and may not be completely accurate.

	StartingEventFilter	HESE	MESE
tpr	$9.62mHz$	$2.98 \cdot \mu Hz$*	$0.311mHz$*
fpr	$8.68Hz$	$0.961mHz$*	$11.3Hz$*
filter rate	$8.69Hz$	$< 10mHz$; $0.964mHz$*	$11Hz$; $11.3Hz$*
true rate	$15.3mHz$	$15.4mHz$*	$15.4mHz$*
purity	$1.11 \cdot 10^{-3}$	$3.09 \cdot 10^{-3}$*	$2.75 \cdot 10^{-5}$*
efficiency	0.630	0.000194*	0.0202*
purity$_{> 100TeV}$	$5.40 \cdot 10^{-7}$; 1.000**	$8.27 \cdot 10^{-4}$*	$8.91 \cdot 10^{-8}$*
efficiency$_{> 100TeV}$	0.873 ; 0.492**	0.286*	0.364*

Table 1: A table listing relevant quantities for StartingEventFilter, HESE and MESE to be able to compare the three filters. In the internal filter document [42] only the filter rates of HESE and MESE were available, the rest of the information was therefore calculated from the simulation databases described in subsection 5.1. Information obtained this way is marked with a *. The purity and efficiency marked with ** have been obtained by using a different cut of 0.9995 for StartingEventFilter that is optimized for high energies.

As HESE and MESE are optimized for different energy ranges they should be compared separately to StartingEventFilter.
Starting with MESE one can see in Table 1 a 40.4 times larger purity for StartingEventFilter than for MESE at $1.11 \cdot 10^{-3}$ vs. $2.75 \cdot 10^{-5}$. This is a drastic increase at a very similar filter rate, correspondingly also the efficiency of StartingEventFilter is much larger than that of MESE at 63.0% retained starting events vs. only 2.02% for MESE. This shows that StartingEventFilter is superior to MESE in regards to all metrics.
In order for the comparison between StartingEventFilter and HESE to be fair one needs to constraint it to the high energy range for which the latter is optimized. For this purpose purity and efficiency have been calculated for the energy range above $100 TeV$ only. It can be seen in Table 1 that in terms of purity StartingEventFilter is by over three orders of magnitude inferior to HESE at $5.40 \cdot 10^{-7}$ vs. $8.27 \cdot 10^{-4}$. In terms of efficiency however StartingEventFilter is much better than HESE at 87.3% vs. 28.6%. The cut returned by the $f_{0.15}$-optimization is done on the entire energy range and therefore optimizes the cut for medium-low energy events due to their dominant rates. If one chooses a higher cut than this in order to optimize the model for high energies the comparison looks completely different. At a cut of 0.9995 the purity above $100 TeV$ for StartingEventFilter is an unprecedented 1.000, when rounded to three decimal places, at still 49.2% efficiency. This means the selection is completely free of false positives and StartingEventFilter surpasses HESE by more than three orders of magnitude in purity while having almost twice the efficiency. Even when going a bit lower in energy to the range above $10 TeV$ the purity at the same cut of 0.9995 is still 0.991 at an efficiency of 39.2%, at above $1 TeV$ it is 1.000 at 22.7% efficiency using again the same cut. This shows that this higher cut yields an effectively perfect purity on a wide energy range, but with an efficiency that drops quite a lot for lower energies. A peculiar observation made when tuning the cut shall be noted here: the purity above $100 TeV$ experiences an extremely stark jump from $7.14 \cdot 10^{-4}$ to 1.000 when increasing the cut only very slightly from 0.9990 to 0.9991. This jump might well be not quite as stark when running StartingEventFilter on real detector data, which is why the chosen cut is a bit higher at 0.9995.
This comparison demonstrates that GNNs are far superior to classical filters with never-before-seen margins. In order to use this superiority the f_β-optimization needs

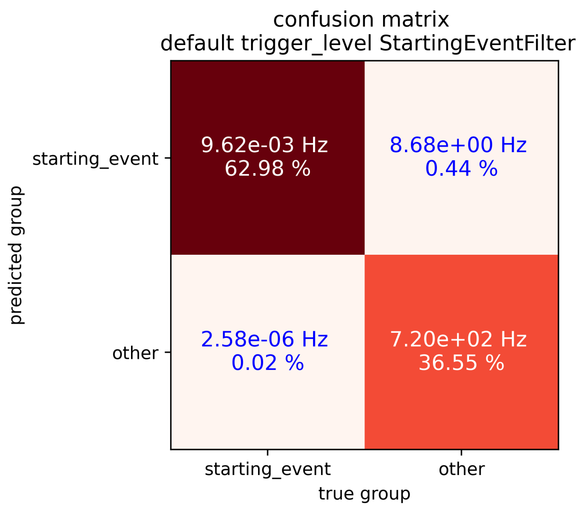

Figure 12: The confusion matrix of StartingEventFilter at the cuts from the $f_{0.15}$-optimization. The relative numbers below the rates indicate the relative rate wrt. the true rate of the corresponding group (true and false efficiencies), note that these don't add up to 100% because events can be classified as no or both groups due to the way that predictions are made (as is explained in subsubsection 5.6.2). One can see a low efficiency of not even 40% for 'other' events, as the 'other' group is of no interest however and only exists for the reason of how classifier networks function, this doesn't matter.

to be performed on specific energy ranges though, that are most important for the desired filtering application as otherwise it will only optimize for low energies. It is also shown that for a wide energy range a perfect purity of 1.0 (or very close to it) can be achieved. This is then however at the cost of quite significantly reduced efficiency for lower energies. Figure 13 shows a histogram over neutrino energy of the true and false positive rates of StartingEventFilter using the discussed perfect-purity cut of 0.9995. The plot really speaks for itself, however a short discussion of these rates is nevertheless given in the caption.

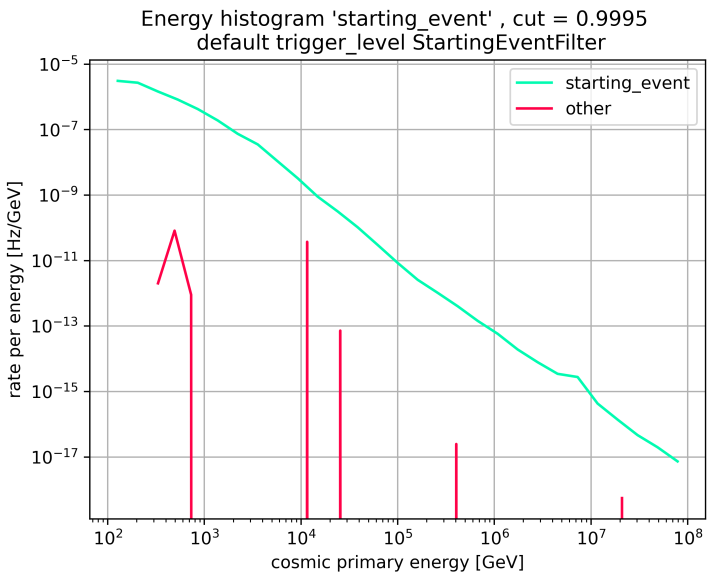

Figure 13: A histogram over energy of the respective cosmic primary of the true and false positive rates of the 'starting_event' group using the high energy optimized cut of 0.9995 discussed above. One can see only a few non-empty bins for the false positives as a logical consequence of the effectively perfect purity of the filtered event sample. The true positive rate falls of roughly with a spectral index of -2, a bit softer than the spectral index of the assumed neutrino spectrum in the simweights according to [76] of 2.37. This is a consequence of increasing efficiency with increasing energy as noted above.

6.2 Double Bang Filter

DoubleBangFilter as the name suggests is intended to produce a selection of double bang events of all energies. subsubsection 3.3.2 explains what double bangs are and why they are so interesting for tau neutrino research that one would build an extra filter dedicated to just them.
This filter is likely too specific, filtering only a single event class, to run as an online filter, it would therefore rather be used as a low-level offline filter running on the pre-filtered level 2 data. It therefore does not need to deal with quite so much background contamination as online filters do, therefore the choice has been made to use a 3:1 ratio selection of NuGen and Corsika events. This selection was again taken from the databases elaborated on in subsection 5.1. The chosen ratio still keeps an adequate balance between signal and background, but using more NuGen events than Corsika allows for using more data in total. All $4.17 \cdot 10^6$ Corsika events were again used as in the training of StartingEventFilter (see subsection 6.1), however this time $12.6 \cdot 10^6$ NuGen events could be used, which is 30% of all available ones instead of just 10%. This made for a total dataset size of $16.77 \cdot 10^6$ events, which

is twice as large as the one used for StartingEventFilter. Out of this total selection again 90% of the events were actually trained on and the remaining 10% were used for validation of the results.
Training was again carried out on four Nvidia RTX 3080 for this time in total 16 epochs, one of which took on average 11:30 hours. After the 16 epochs the training was not done, meaning that overfitting had not yet started, unfortunately the deadline to hand this thesis in on was approaching and the training had to be stopped. Consequentially the obtained best-fit model is not necessarily the optimal model that would have been obtained if the training could have finished. The validation loss started out at a value of 0.148 after epoch 0, by the end of epoch 15 it has decreased to a final value of 0.134. This presents a decrease in loss of merely 9.46%, which indicates not too much was learned by the model during training. For comparison: the loss of StartingEventFilter decreased by 39.8% during its training. However here no such weird error as in the training of StartingEventFilter ever occurred.

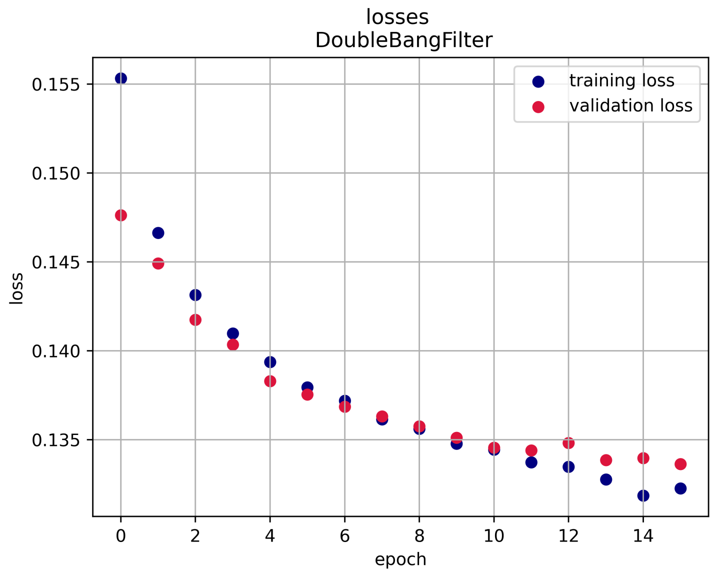

Figure 14: Training and validation loss plotted over the epochs, beginnings of overfitting can be seen from epoch 12 on, but the actual overfitting was not reached in the available training time. This also means that the validation loss would very likely have further decreased if training could have been finished.

The default and perturbed ROC curves for DoubleBangFilter can be seen in Figure 15. The robustness of the model to the relevant measurement uncertainties can again be very clearly seen in that both curves overlap entirely such that only the default curve is visible. The AUC of both curves is 0.962, interestingly this is exactly the same value as for StartingEventFilter in Figure 11, however this does not

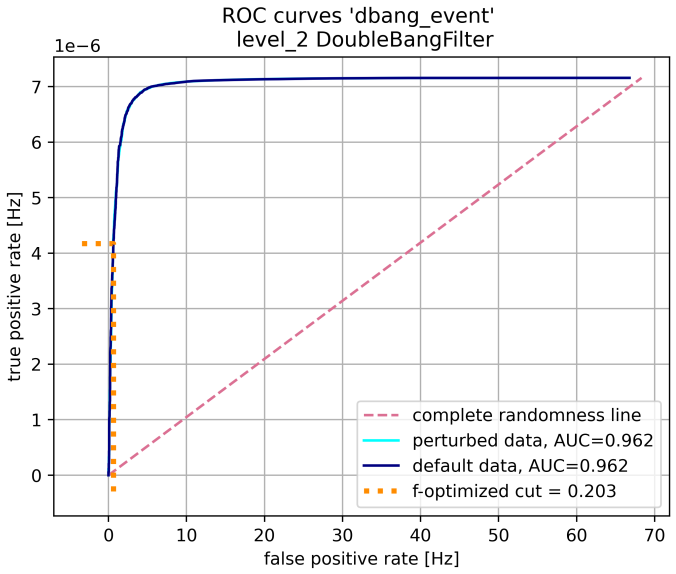

Figure 15: The ROC curve for the 'dbang_event' group of DoubleBangFilter for both the default and the perturbed validation data, just as in Figure 11 for StartingEventFilter they are virtually equal, showing that also this model is robust to IceCube's measurement uncertainties. The orange dotted line marks the true and false positive rates at the cut of 0.203 suggested by an $f_{0.1}$-optimization.

allow for the conclusion of equal performance of the two, as the underlying groups are different. An external value to compare the AUC against could not be found. An f_β-optimization has been performed again, $\beta = 0.1$ has been chosen here with the goal of achieving a low filter rate of about $1Hz$. This filter rate was chosen aiming for a good purity without neglecting efficiency too much as desirable for a low-level offline filter (subsection 3.5 briefly describes offline filters and their intentions and priorities). The $f_{0.1}$-optimization yielded a cut of 0.203 for the 'dbang_event' group and 0.996 for the 'other' group. As a result the model has a total filter rate of $625mHz$ composed of a true positive rate of $4.17\mu Hz$ and a false positive rate of $625mHz$ as can also be seen in the green dotted line in Figure 15. Therefore the purity is $6.68 \cdot 10^{-6}$ at an efficiency of 0.583. The purity is extremely poor, this seems peculiar at first considering that the $f_{0.1}$-optimization was intended to produce a sample with good purity corresponding to the intended role as low-level offline filter. An important reason for this is definitely the small purity of the input level 2 data of only $1.05 \cdot 10^{-7}$ caused by the very low true rate of double bang events of only $7.16\mu Hz$ (for comparison: starting events have a true rate of $15.3mHz$, more than 2000 times higher). However even in relation to that initial purity DoubleBangFilter only achieves an increase of a factor of 63.6, significantly lower than the factor of 143 by which StartingEventFilter purifies its input data. The logical conclusion of the low purification is that there must be a problem with the filter. Very roughly this problem is that distinguishing double bangs from other cascade events is challenging. A detailed elaboration on this is given in the following.

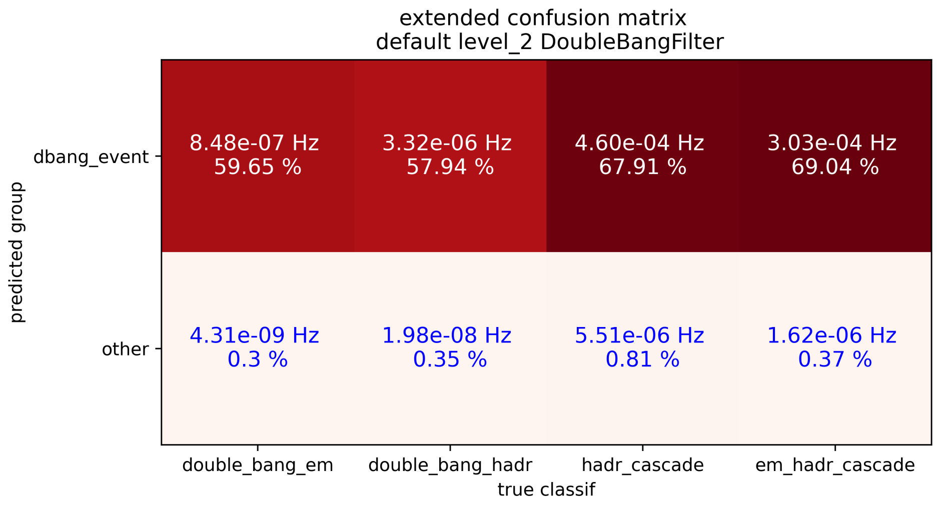

Figure 16: The extended confusion matrix of DoubleBangFilter at the cuts from the $f_{0.1}$-optimization. The relative numbers below the rates indicate the relative rate (efficiency) wrt. the total true rate of events of the corresponding classification, again note that these don't add up to 100% because events can be classified as no or both groups as the prediction for each group is made independently (see subsubsection 5.6.2). The problematic confusion of double bang and single cascade events can clearly be seen in that they are both dominantly accepted as 'dbang_event's.

Figure 16 shows a detailed analysis of the discrimination performance of DoubleBangFilter in what I would like to call the extended confusion matrix. It lists relevant event classification types on the horizontal axis and the corresponding rates in which they get assigned to the two available groups on the vertical axis. Looking at this extended confusion matrix one can see that both electromagnetic and hadronic double bangs can with acceptable efficiency of almost 60% be recognized as such. However both hadronic cascades from NC-interactions and electromagnetic-hadronic cascades from NuE CC-interactions are also dominantly recognized as double bangs at even higher efficiency of almost 70%. Not only do a lot of single cascade events get classified a double bangs and therefore contribute to the false positive rate (this is less of an issue because the false positive rate is absolutely dominated by CR induced background events and therefore hardly changes through the addition of the rate from single cascades), but also does this confusion reduce the efficiency with which true double bangs are recognized as such and consequentially also the true positive rate and finally the purity. The reduced efficiency can indirectly be seen in Figure 16 in that the model only makes a prediction for 60.0% of electromagnetic

and 58.3% of hadronic double bangs respectively, a significant deviation from the ideal 100%. The remaining 40% and 41.7% respectively get predicted as neither of the two groups, demonstrating the huge difficulty the model has with the discrimination between double bangs and single cascades. The underlying reason for the confusion is simple and has already been noted in [94]: The separation length of the two cascades of a double bang is determined by the tau decay length which depends on its energy ($49m/PeV$). For low energies the separation is small (e.g. $4.9mm$ at $100GeV$) and the two cascades overlap making them virtually look like a single one, it is at these energies consequentially impossible to distinguish a double bang from a single cascade event. Only when the separation is big enough such that the two cascades can be resolved in the PMT pulses (a so-called double-pulse signature) it is possible to make a distinction between double bangs and single cascades. Taking the nuclear radiation length of the (first) hadronic cascade of about $1m$ ([29]) as an approximation of the required separation, double bangs can theoretically be recognized reliably as such for energies above about $20TeV$. This presents only a lower bound however as the effective length of a cascade is actually several interaction lengths.
Figure 17 shows a histogram over the energy of the respective cosmic primary neutrino of the efficiencies with which double bangs and single cascade events get accepted as 'dbang_event's respectively by DoubleBangFilter. For low energies it is maximal as expected with both event types having similar efficiencies of about 75% at $100GeV$ and about 35% at $400TeV$. After that the efficiency curves then separate, the one for single cascades drops quickly, while the one for true double bangs stays high. Oddly however it does not approach 100% for the highest energies, as one would expect given that double bangs are very well separated there and thus easy to recognize. One can see rather strong fluctuations for double bangs at the high energy tail due to lack of sufficient amounts of corresponding events at these high energies. Correspondingly the training of the model to recognize them was likely only insufficiently possible in that case explaining the low observed efficiency in Figure 17. This hypothesis is supported be fact that the efficiency first increases, reaching a local maximum of about 50% at $3PeV$ and then decreases with large fluctuations when the lack of events becomes seemingly preeminent.

Considering both the theoretical explanation of the confusion as well as the proposed explanation behind the suboptimal high-energy behaviour seen in Figure 17, two things need to be changed in order to improve upon DoubleBangFilter in its current state: Firstly one must make an energy cut in order to only try to discriminate between physically distinguishable events. For this energy cut, looking at Figure 17, it seems adequate using $500TeV$, going a bit lower than this to e.g. $100TeV$ is however worth a try. Secondly one must use a training dataset with an increased amount of double bang events at the high energy tail to allow for proper training there. Independently of these improvements it is probably also advantageous to

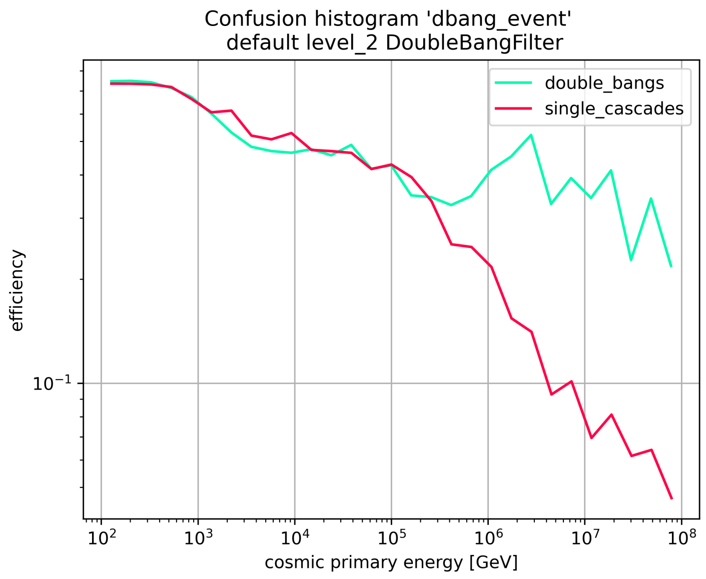

Figure 17: A histogram over energy of the respective cosmic primary neutrino of the efficiencies with which double bangs and single cascades respectively get accepted as 'dbang_event's by DoubleBangFilter. The observed maximal confusion for low energies is as expected, as well is the significantly decreased efficiency for single cascades at high energies. On the contrary to what is seen here one would however expect the double bang efficiency to approach 100% for high energies, a possible explanation of this odd behaviour is given above.

choose a higher cut than that returned by the $f_{0.1}$-optimization as the efficiency is currently rather large at 58.3% and, as subsection 6.1 as shown, sacrificing some of it can allow for extreme increases in purity.

As the superiority of machine learning based filters over classical filters has already been demonstrated in subsection 6.1 it would have been desirable to compare DoubleBangFilter to a corresponding ananlog, which is also machine learning based. Such a filter could unfortunately not be found though. While a CNN based network, which amongst others is designed to filter double bangs, is presented in [94], the comparability of the two is unfortunately not given/cannot be confirmed. This is due to a long list of reasons, the most important of which are that [94] does not provide the number of parameters the CNN has and comparing a readily trained model with DoubleBangFilter, who's training needed to be interrupted prematurely due to the deadline for this thesis, is unfair. [94] also does not use CR background simulation data in the training of the CNN, but neutrino simulation only, and does not use simweights in its analysis, both of which I regard as severe methodical flaws.

7 Conclusion and Outlook

The results obtained for StartingEventFilter presented in subsection 6.1 of this thesis demonstrate that GNNs are by orders of magnitude superior to classical filters. It surpasses the MESE filter in terms of purity by 1.6 orders of magnitude when applying the $f_{0.15}$-optimized cut. When applying the higher cut of 0.9995 is shows perfect purity, at varying, but over a wide energy range good, efficiency. This presents an unprecedented result and I can understand that people might deem it too good to be true. I assure that the analysis has been done with greatest care and nothing was manipulated intentionally. Still, in order for others to be able to reproduce the results and convince themselves, both StartingEventFilter and DoubleBangFilter as well as the employed training and analysis scripts will be uploaded to GitHub at [95] in the next days following the publication of this thesis.
It is essential to recognize that GNNs, like any other data analysis method, can however not perform magic, as is evident in the results presented for DoubleBangFilter in subsection 6.2. Even they are limited in that they can only discriminate between events that are physically distinguishable. Consequently an energy cut is still required to reliably be able to recognize double bangs. Also, as with any neural network, they need to be trained on datasets that have sufficient statistics across all energies. How GNNs compare to other neutral network architectures, specifically CNNs, could not be investigated here, as no suitable model to compare DoubleBangFilter to was found.

The conclusion of this thesis can only be that there is not only huge potential for GNNs to replace the current online filter infrastructure at the South Pole in the near future, it is even inevitable. They will earlier or later also replace the high-level offline filters. For this one might however want to use larger models than those used in this thesis, which have approximately $1.4 \cdot 10^6$ parameters, in order to increase their capability even more. Training such models will necessitate larger datasets and extended training periods, possibly spanning months, depending on the available computing infrastructure, particularly the types and quantity of GPUs.

In addition to the primary result of demonstrating the unprecedented efficacy of GNN filters, this thesis also provides valuable data labeling and conversion scripts, detailed in subsection 5.1, and new GraphNeT classes for designing and training filter networks. The prior one of these resources will be made available on GitHub at [96], while the latter will be committed to GraphNeT so that it will come with every new GraphNeT installation (the installation guide can be found in [79]).
The data processing scripts are a powerful tool for converting I3 NuGen and Corsika simulation files into versatile labeled databases applicable to any machine learning task in IceCube. However, before further use, it is necessary to investigate and resolve the issue with Corsika simweights as described in subsubsection 5.1.5. A subse-

quent step would be to extend functionality to other generators like MuonGun (used for simulating atmospheric muons with better high-energy statistics than Corsika). This will require adding a corresponding weighter-class to the simweights library, which does not currently exist. Another essential step that needs to be taken in order to improve on the scripts is implementing parts of it in C++ in order to reduce runtimes and thus enable the creation of bigger databases in acceptable times, this is of particular importance for Corsika simulations as discussed in subsection 5.1.
Regarding the model definitions of the GNN filters, it is currently only possible to define groups in terms of event classification. However, many applications, such as upgrading DoubleBangFilter (as discussed in subsection 6.2) require the ability to incorporate some sort of energy and possibly other quantities from the truth table in group definitions. Therefore, the next step is to generalize the grouping scheme described in subsubsection 5.3.1 to accommodate these requirements.
Overall, the model setup, training pipeline and GraphNeT have immense potential. Despite requiring patience with training times, they can be effectively utilized with larger datasets and models to further improve the capabilities of GNN filters in IceCube.

References

[1] IceCube Collaboration et al. "Neutrino emission from the direction of the blazar TXS 0506+056 prior to the IceCube-170922A alert". In: *Science* 361.6398 (2018), pp. 147–151. DOI: https://doi.org/10.1126/science.aat2890.

[2] R. Abbasi et al. "Evidence for neutrino emission from the nearby active galaxy NGC 1068". In: *Science* 378.6619 (2022), pp. 538–543. DOI: https://doi.org/10.1126/science.abg3395.

[3] IceCube Collaboration. "Observation of high-energy neutrinos from the Galactic plane". In: *Science* 380.6652 (2023), pp. 1338–1343. DOI: https://doi.org/10.1126/science.adc9818.

[4] Rasmus Orsoe et al. "Graph Neural Networks for low-energy event classification & reconstruction in IceCube". In: *Journal of Instrumentation* 17.11 (Nov. 2022), P11003. ISSN: 1748-0221. DOI: 10.1088/1748-0221/17/11/p11003. URL: http://dx.doi.org/10.1088/1748-0221/17/11/P11003.

[5] IceCube Collaboration. *1931 – Pauli presents hypothetical "neutron" particle*. URL: https://icecube.wisc.edu/neutrino-history/1931/01/1931-pauli-presents-hypothetical-neutron-particle/. (accessed: 06/23/2024).

[6] Enrico Fermi. "Versuch einer Theorie der β-Strahlen". In: *I. Z. Physik* (1934), 162 c). DOI: https://doi.org/10.1007/BF01351864.

[7] Los Alamos National Laboratory. *Ghost particles and Project Poltergeist.* URL: `https://discover.lanl.gov/news/1029-ghost-particles/`. (accessed: 06/23/2024).

[8] Pablo Fernandez. *Spectrum-of-all-neutrino-sources-except-accelerator-neutrinos-CB-Cosmological.png.* URL: `https://www.researchgate.net/profile/Pablo-Fernandez-10/publication/315458901/figure/fig1/AS:541071488425984@1506012903026/Spectrum-of-all-neutrino-sources-except-accelerator-neutrinos-CB-Cosmological.png`. (accessed: 08/05/2024).

[9] Yoichiro Suzuki. *Solar Neutrinos.* URL: `https://www.slac.stanford.edu/econf/C990809/docs/suzuki.pdf`. (accessed: 06/23/2024).

[10] J. Guilet et al. T. Foglizzo R. Kazeroni. "The Explosion Mechanism of Core-Collapse Supernovae: Progress in Supernova Theory and Experiments." In: *Publications of the Astronomical Society of Australia* (2015). DOI: `http://dx.doi.org/10.1017/pasa.2015.9`.

[11] Thomas K. Gaisser, Ralph Engel, and Elisa Resconi. "Astrophysical γ -rays and neutrinos". In: *Cosmic Rays and Particle Physics.* Cambridge University Press, 2016, pp. 220–235.

[12] Matthew G. Baring. *Diffusive Shock Acceleration: the Fermi Mechanism.* 1997. arXiv: `astro-ph/9711177`.

[13] Emil Bols. *Proton-Proton Central Exclusive Pion Production at $\sqrt{s} = 13TeV$ with the ALFA and ATLAS Detector.* URL: `https://cds.cern.ch/record/2288372/files/CERN-THESIS-2017-175.pdf`. (accessed: 07/31/2024).

[14] H. Gao et al. W. Chen. *The $\gamma p \longrightarrow \pi^+ n$ Single Charged Pion Photoproduction.* URL: `https://www.jlab.org/exp_prog/proposals/08/PR-08-003.pdf`. (accessed: 07/31/2024).

[15] Maxim Lyutikov. "Inverse Compton model of pulsar high-energy emission". In: *Monthly Notices of the Royal Astronomical Society* 431.3 (2013), pp. 2580–2589. DOI: `https://doi.org/10.1093/mnras/stt351`.

[16] Konrad Bernlöhr. *Cosmic-ray air showers.* URL: `https://www.mpi-hd.mpg.de/hfm/CosmicRay/Showers.html`. (accessed: 06/27/2024).

[17] IceCube Collaboration. "Measurement of the multi-TeV neutrino interaction cross-section with IceCube using Earth absorption". In: *Nature* 551.7682 (Nov. 2017), pp. 596–600. ISSN: 1476-4687. DOI: `10.1038/nature24459`. URL: `http://dx.doi.org/10.1038/nature24459`.

[18] J. Adams et al. M. G. Aartsen M. Ackermann. "Measurement of the Atmospheric NuE Spectrum with IceCube". In: *Physical Review D* 91.12 (June 2015). ISSN: 1550-2368. DOI: `10.1103/physrevd.91.122004`. URL: `http://dx.doi.org/10.1103/PhysRevD.91.122004`.

[19] Aiswarya Sivaprasad et al. “Overview of The Super-Kamiokande Neutrino Detection Experiment”. In: *Naxxatra sciences and collaborative research* (July 2021).

[20] A. Bellerive et al. “The Sudbury Neutrino Observatory”. In: *Nuclear Physics B* 908 (2016). Neutrino Oscillations: Celebrating the Nobel Prize in Physics 2015, pp. 30–51. ISSN: 0550-3213. DOI: `https://doi.org/10.1016/j.nuclphysb.2016.04.035`. URL: `https://www.sciencedirect.com/science/article/pii/S0550321316300736`.

[21] IceCube Collaboration. *IceCube*. URL: `https://icecube.wisc.edu/science/icecube/`. (accessed: 06/18/2024).

[22] IceCube Collaboration. “The IceCube Neutrino Observatory: instrumentation and online systems”. In: *Journal of Instrumentation* 12.03 (Mar. 2017), pp. 6–17. ISSN: 1748-0221. DOI: `10.1088/1748-0221/12/03/p03012`. URL: `http://dx.doi.org/10.1088/1748-0221/12/03/P03012`.

[23] I. Frank and Ig. Tamm. “Coherent Visible Radiation of Fast Electrons Passing Through Matter”. In: *Selected Papers*. Ed. by Boris M. Bolotovskii, Victor Ya. Frenkel, and Rudolf Peierls. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 29–35. ISBN: 978-3-642-74626-0. DOI: `10.1007/978-3-642-74626-0_2`. URL: `https://doi.org/10.1007/978-3-642-74626-0_2`.

[24] National Science Foundation. *Amundsen-Scott South Pole Station*. URL: `https://www.nsf.gov/geo/opp/support/southp.jsp`. (accessed: 06/27/2024).

[25] Thorsten Gluesenkamp. *Sketched lateral cut of an IceCube DOM*. URL: `https://www.researchgate.net/profile/Thorsten-Gluesenkamp/publication/215457896/figure/fig7/AS:654046566309891@1532948261501/Sketched-lateral-cut-of-an-IceCube-DOM-Taken-from-55.png`. (accessed: 06/18/2024).

[26] Berkeley Lab. *IceCube schema*. URL: `https://newscenter.lbl.gov/wp-content/uploads/2010/12/IceCube-schema.jpg`. (accessed: 06/18/2024).

[27] J. A. Formaggio and G. P. Zeller. “From eV to EeV: Neutrino cross sections across energy scales”. In: *Reviews of Modern Physics* 84.3 (Sept. 2012), pp. 1307–1341. ISSN: 1539-0756. DOI: `10.1103/revmodphys.84.1307`. URL: `http://dx.doi.org/10.1103/RevModPhys.84.1307`.

[28] Particle Data Group. *GAUGE AND HIGGS BOSONS: W*. URL: `https://pdglive.lbl.gov/Particle.action?node=S043&init=0`. (accessed: 06/27/2024).

[29] Particle Data Group. *Atomic and nuclear properties of water (ice) (H2O)*. URL: `https://pdg.lbl.gov/2020/AtomicNuclearProperties/HTML/water_ice.html`. (accessed: 07/29/2024).

[30] R.L. Workman et al. (Particle Data Group). “34. Passage of Particles Through Matter”. In: *Prog. Theor. Exp. Phys. 2022* (Aug. 2022). URL: https://pdg.lbl.gov/2022/reviews/rpp2022-rev-passage-particles-matter.pdf.

[31] Dmitry Chirkin and Wolfgang Rhode. *Propagating leptons through matter with Muon Monte Carlo (MMC).* 2016. arXiv: hep-ph/0407075 [hep-ph]. URL: https://arxiv.org/abs/hep-ph/0407075.

[32] Particle Data Group. *58. τ Branching Fractions.* URL: https://pdg.lbl.gov/2024/reviews/rpp2024-rev-tau-branching-fractions.pdf. (accessed: 06/27/2024).

[33] M. Ackermann et al. M. G. Aartsen R. Abbasi. “Energy reconstruction methods in the IceCube neutrino telescope”. In: *Journal of Instrumentation* 9.03 (Mar. 2014), P03009–P03009. ISSN: 1748-0221. DOI: 10.1088/1748-0221/9/03/p03009. URL: http://dx.doi.org/10.1088/1748-0221/9/03/P03009.

[34] IceCube Collaboration. “IceCube Data for Neutrino Point-Source Searches Years 2008-2018”. In: (2021). DOI: https://doi.org/10.21234/cpkq-k003. URL: https://arxiv.org/abs/2101.09836.

[35] D F Cowen and (forthe IceCube Collaboration). “Tau Neutrinos in IceCube”. In: *Journal of Physics: Conference Series* 60.1 (Mar. 2007), p. 227. DOI: 10.1088/1742-6596/60/1/048. URL: https://dx.doi.org/10.1088/1742-6596/60/1/048.

[36] J. Adams et al. R. Abbasi M. Ackermann. “Characterization of the astrophysical diffuse neutrino flux using starting track events in IceCube”. In: *Physical Review D* 110.2 (July 2024). ISSN: 2470-0029. DOI: 10.1103/physrevd.110.022001. URL: http://dx.doi.org/10.1103/PhysRevD.110.022001.

[37] K. Abraham et al. M.G. Aartsen. “Characterization of the atmospheric muon flux in IceCube”. In: *Astroparticle Physics* 78 (May 2016), pp. 1–27. ISSN: 0927-6505. DOI: 10.1016/j.astropartphys.2016.01.006. URL: http://dx.doi.org/10.1016/j.astropartphys.2016.01.006.

[38] Thorsten Glüsenkamp. *Muon Filter Proposal IC86-2012.* IceCube-internal document. 2012.

[39] Mariola Lesiak-Bzdak Lukas Schulte. *Request for the Online Cascade Filter and Additional Calculations at Level 2 for the IC86 2013 Cascade Filter Stream.* IceCube-internal document. 2012.

[40] Markus Voge. *2012 TFT Proposal Request for a Level2 Online Muon Filter Revised Version 1.* IceCube-internal document. 2012.

[41] Keiichi Mase. *Request for the EHE filter (2013) (ver. 4).* IceCube-internal document. 2013.

[42] Nancy Wandowsky Azadeh Keivani Claudio Kopper. *TFT proposal for a ”Starting Event Filter” for the 2015/16 season.* IceCube-internal document. 2015.

[43] Juan Carlos Díaz-Vélez. *Neutrino and Air Shower Simulations in IceCube.* URL: https://events.icecube.wisc.edu/event/123/contributions/6472/attachments/5487/6310/Simulation_in_IceCube_2020.pdf. (accessed: 07/09/2024).

[44] The IceTray Contributors. *neutrino-generator - Introduction.* URL: https://docs.icecube.aq/icetray/main/projects/neutrino-generator/intro.html. (accessed: 07/09/2024).

[45] Karlsruhe Institute of Technology. *CORSIKA.* URL: https://www.iap.kit.edu/corsika/index.php. (accessed: 07/09/2024).

[46] J. H. Koehne et al. "PROPOSAL: A tool for propagation of charged leptons". In: *Comput. Phys. Commun.* 184 (2013), pp. 2070–2090. DOI: 10.1016/j.cpc.2013.04.001.

[47] The IceTray Contributors. *cmc - Cascade Monte Carlo.* URL: https://docs.icecube.aq/icetray/main/projects/cmc/index.html. (accessed: 08/01/2024).

[48] The IceTray Contributors. *clsim - Overview.* URL: https://docs.icecube.aq/icetray/main/projects/clsim/overview.html. (accessed: 07/09/2024).

[49] The IceTray Contributors. *Polyplopia.* URL: https://docs.icecube.aq/icetray/main/projects/polyplopia/. (accessed: 07/09/2024).

[50] The IceTray Contributors. *Vuvuzela.* URL: https://docs.icecube.aq/icetray/main/projects/vuvuzela/. (accessed: 07/09/2024).

[51] The IceTray Contributors. *PMTResponseSimulator.* URL: https://docs.icecube.aq/icetray/main/projects/DOMLauncher/PMTRes.html#pmtresponsesimulator. (accessed: 07/09/2024).

[52] The IceTray Contributors. *DOMLauncher.* URL: https://docs.icecube.aq/icetray/main/projects/DOMLauncher/DOML.html. (accessed: 07/09/2024).

[53] M. Ackermann et al. M. G. Aartsen R. Abbasi. "Energy reconstruction methods in the IceCube neutrino telescope". In: *Journal of Instrumentation* 9.03 (Mar. 2014), P03009–P03009. ISSN: 1748-0221. DOI: 10.1088/1748-0221/9/03/p03009. URL: http://dx.doi.org/10.1088/1748-0221/9/03/P03009.

[54] The IceTray Contributors. *trigger-sim.* URL: https://docs.icecube.aq/icetray/main/projects/trigger-sim/index.html. (accessed: 07/09/2024).

[55] Robin J. Wilson. *Introduction to Graph Theory.* Addison Wesley Longman Limited, 1996. ISBN: 0-582-24993-7.

[56] PyTorch Contributors. *PyTorch documentation.* URL: https://pytorch.org/docs/stable/index.html. (accessed: 06/30/2024).

[57] PyTorch Contributors. *ReLU.* URL: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html#torch.nn.ReLU. (accessed: 06/30/2024).

[58] PyTorch Contributors. *Tanh.* URL: https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html#torch.nn.Tanh. (accessed: 06/30/2024).

[59] PyTorch Contributors. *Sigmoid.* URL: https://pytorch.org/docs/stable/generated/torch.nn.Sigmoid.html#torch.nn.Sigmoid. (accessed: 06/30/2024).

[60] PyG Team. *conv.GraphConv.* URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GraphConv.html#torch_geometric.nn.conv.GraphConv. (accessed: 06/18/2024).

[61] Maxime Labonne. *Graph Convolutional Networks: Introduction to GNNs.* URL: https://towardsdatascience.com/graph-convolutional-networks-introduction-to-gnns-24b3f60d6c95. (accessed: 06/29/2024).

[62] Kurtis Pykes. *The Difference Between Classification and Regression in Machine Learning.* URL: https://towardsdatascience.com/the-difference-between-classification-and-regression-in-machine-learning-4ccdb5b18fd3. (accessed: 06/27/2024).

[63] Julianna Delua. *Supervised versus unsupervised learning: What's the difference?* URL: https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning. (accessed: 07/01/2024).

[64] Dirk P. Kroese Reuven Y. Rubinstein. *The Cross-Entropy Method.* Springer New York, NY, 2004, pp. 251–270. ISBN: 978-1-4419-1940-3.

[65] R. Williams D. Rumelhart G. Hinton. "Learning representations by back-propagating errors". In: *Nature* 323 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[66] Aishwarya V Srinivasan. *Stochastic Gradient Descent — Clearly Explained!!* URL: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31. (accessed: 08/01/2024).

[67] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[68] PyTorch Contributors. *ReduceLROnPlateau.* URL: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html. (accessed: 06/27/2024).

[69] Shiv Vignesh. *The Perfect Fit for a DNN.* URL: https://medium.com/analytics-vidhya/the-perfect-fit-for-a-dnn-596954c9ea39. (accessed: 06/29/2024).

[70] Madison servers. /cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/RHEL_7_x86_64/metaproject
services/resources/scripts/label_events.py.

[71] Madison servers. /data/ana/graphnet/l2_labeled/documentation.txt.

[72] Madison servers. /data/ana/graphnet/l2_labeled.

[73] The IceTray Contributors. *I3MCTree.* URL: https://docs.icecube.aq/icetray/main/projects/dataclasses/i3mctree.html#i3mctree. (accessed: 07/21/2024).

[74] the SimWeights contributors. *SimWeights.* URL: https://docs.icecube.aq/simweights/main/. (accessed: 06/29/2024).

[75] Thomas Gaisser. *Cosmic ray spectrum and composition ¿ 100 TeV.* URL: https://internal-apps.icecube.wisc.edu/reports/data/icecube/2011/02/004/icecube_201102004_v2.pdf. (accessed: 06/18/2024).

[76] M. Ackermann et al. R. Abbasi. "Improved Characterization of the Astrophysical Muon–neutrino Flux with 9.5 Years of IceCube Data". In: *The Astrophysical Journal* 928.1 (Mar. 2022), p. 50. ISSN: 1538-4357. DOI: 10.3847/1538-4357/ac4d29. URL: http://dx.doi.org/10.3847/1538-4357/ac4d29.

[77] Silvia Bravo. *Measuring the flavor ratio of astrophysical neutrinos.* URL: https://icecube.wisc.edu/news/research/2015/02/measuring-flavor-ratio-of-astrophysical-neutrinos/. (accessed: 06/29/2024).

[78] Madison servers. /data/ana/Diffuse/NNMFit/MCEq_splines/v1.2.1/MCEq_splines_PRI-GSF-spline_INT-SIBYLL23c_allfluxes.pickle.

[79] GraphNeT Team. *GraphNeT - A Deep Learning Library for Neutrino Telescopes.* URL: https://graphnet-team.github.io/graphnet/index.html. (accessed: 06/25/2024).

[80] Creators of PyTorch Lightning. *WELCOME TO PYTORCH LIGHTNING.* URL: https://lightning.ai/docs/pytorch/stable/. (accessed: 06/30/2024).

[81] PyG Team. *PyG Documentation.* URL: https://pytorch-geometric.readthedocs.io/en/latest/. (accessed: 06/30/2024).

[82] GraphNeT Team. *High-level overview of a typical workflow using GraphNeT.* URL: https://graphnet-team.github.io/graphnet/_images/flowchart.png. (accessed: 06/25/2024).

[83] E. S. Keeping J. F. Kenney. "Percentile Ranks". In: Mathematics of Statistics, Pt. 1, 3rd ed. Princeton, NJ: Van Nostrand, 1962. Chap. 3.6, pp. 38–39.

[84] F. F. Yao D. Eppstein M. S. Paterson. "On Nearest-Neighbor Graphs". In: *Discrete Comput Geom* 17 (1997), pp. 263–282. DOI: https://doi.org/10.1007/PL00009293.

[85] GraphNeT team. *dynedge.* URL: https://graphnet-team.github.io/graphnet/api/graphnet.models.gnn.dynedge.html#module-graphnet.models.gnn.dynedge. (accessed: 07/05/2024).

[86] R. Abbasi et al. "Graph Neural Networks for low-energy event classification & reconstruction in IceCube". In: *JINST* 17 (2022), P11003. DOI: https://doi.org/10.1088/1748-0221/17/11/P11003.

[87] Yongbin Sun et al. Yue Wang. “Dynamic Graph CNN for Learning on Point Clouds”. In: *ACM Trans. Graph.* 38.5 (Oct. 2019). ISSN: 0730-0301. DOI: `10.1145/3326362`. URL: `https://doi.org/10.1145/3326362`.

[88] Rasmus Ørsøe. private chat.

[89] Erlangen National High Performance Computing Center (NHR@FAU). *Woody.* URL: `https://doc.nhr.fau.de/clusters/woody/`. (accessed: 07/21/2024).

[90] Erlangen National High Performance Computing Center (NHR@FAU). *TinyGPU.* URL: `https://doc.nhr.fau.de/clusters/tinygpu/`. (accessed: 07/21/2024).

[91] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, pp. 861–874. ISSN: 0167-8655. DOI: `https://doi.org/10.1016/j.patrec.2005.10.010`. URL: `https://www.sciencedirect.com/science/article/pii/S016786550500303X`.

[92] Yutaka Sasaki. “The truth of the F-measure”. In: *Teach Tutor Mater* (Jan. 2007). URL: `https://www.researchgate.net/publication/268185911_The_truth_of_the_F-measure`.

[93] Christian Haack. private conversation.

[94] Theo Glauch Maximilian Kronmueller. “Application of Deep Neural Networks to Event Type Classification in IceCube”. In: (July 2019), p. 937. DOI: `10.22323/1.358.0937`.

[95] Fabian Wohlfahrt. *GNNs LePeler bachelors thesis.* URL: `https://github.com/icecube/GNNs_LePeler_bachelors_thesis`. (accessed: 08/02/2024).

[96] Fabian Wohlfahrt. *sim data labeler db converter.* URL: `https://github.com/icecube/sim_data_labeler_db_converter`. (accessed: 08/02/2024).

[97] Particle Data Group. *43. Monte Carlo Particle Numbering Scheme.* URL: `https://pdg.lbl.gov/2019/reviews/rpp2019-rev-monte-carlo-numbering.pdf`. (accessed: 06/29/2024).

[98] IceCube Collaboration. *SimProd Dash.* URL: `https://grid2.icecube.wisc.edu/datasets#IC86-2023`. IC86-2023; (accessed: 06/25/2024).

Appendix

Table 2: a list of all truth labels in the databases with short description; NULL is used as a padding value where labels are not applicable (e.g. if an event doesn't have a third interaction). For descriptions of the filters see subsection 3.5. (see subsection 5.1 for context)

label	dtype	description
cosmic_primary_type	int	cosmic primary particle type as pdg encoding ([97])
cosmic_primary_energy	float	energy of the cosmic primary
cosmic_primary_zenith	float	zenith of cosmic primary direction of origin
cosmic_primary_azimuth	float	azimuth of cosmic primary direction of origin
classification	int	event classification as integer id, see Table 5 for detailed explanation
primary_type	int	detector primary particle type as pdg encoding
primary_energy	float	energy of the detector primary at birth
primary_zenith	float	zenith of detector primary direction of origin
primary_azimuth	float	azimuth of detector primary direction of origin
cosmic_prim_is_detector_prim	bool	are the cosmic primary and the detector primary the same particle?
detector_entry_x	float	x coordinate of detector primary's detector entry position
detector_entry_y	float	y coordinate of detector primary's detector entry position
detector_entry_z	float	z coordinate of detector primary's detector entry position
detector_entry_time	float	time of detector primary's detector entry
detector_entry_energy	float	energy of detector primary at detector entry (this is relevant for muons/taus who loose energy during propagation)
first_vertex_x	float	x coordinate of first interaction vertex of the event inside the detector
first_vertex_y	float	y coordinate of first interaction vertex of the event inside the detector
first_vertex_z	float	z coordinate of first interaction vertex of the event inside the detector

label	dtype	description
first_vertex_energy	float	energy deposited in first interaction vertex of the event inside the detector
second_vertex_x	float	x coordinate of second interaction vertex of the event inside the detector
second_vertex_y	float	y coordinate of second interaction vertex of the event inside the detector
second_vertex_z	float	z coordinate of second interaction vertex of the event inside the detector
second_vertex_energy	float	energy deposited in second interaction vertex of the event inside the detector
third_vertex_x	float	x coordinate of third interaction vertex of the event inside the detector
third_vertex_y	float	y coordinate of third interaction vertex of the event inside the detector
third_vertex_z	float	z coordinate of third interaction vertex of the event inside the detector
third_vertex_energy	float	energy deposited in third interaction vertex of the event inside the detector
visible_track_energy	float	energy deposited in tracks inside the detector
visible_spur_energy	float	energy deposited in spurs inside the detector
detector_exit_x	float	x coordinate of event's last particle's detector exit position
detector_exit_y	float	y coordinate of event's last particle's detector exit position
detector_exit_z	float	z coordinate of event's last particle's detector exit position
detector_exit_energy	float	energy of event's last particle at detector exit
bundle_vertex_x	float	x coordinate of vertex position at which muon bundle was created
bundle_vertex_y	float	y coordinate of vertex position at which muon bundle was created
bundle_vertex_z	float	z coordinate of vertex position at which muon bundle was created
bundle_energy	float	total energy of all muons in muon bundle (both inside and outside the detector)

label	dtype	description
bundle_size	int	total number of all muons in muon bundle (both inside and outside the detector)
closest_approach	float	closest approach distance of uncontained event to the detector
closest_approach_x	float	x coordinate of closest approach of uncontained event to the detector
closest_approach_y	float	y coordinate of closest approach of uncontained event to the detector
closest_approach_z	float	z coordinate of closest approach of uncontained event to the detector
num_bg_primaries	int	number of background particles that entered the detector
CascadeFilter_13	bool	Did the event pass CascadeFilter_13?
MuonFilter_13	bool	Did the event pass MuonFilter_13?
OnlineL2Filter_17	bool	Did the event pass OnlineL2Filter_17?
HESEFilter_15	bool	Did the event pass HESEFilter_15?
MESEFilter_15	bool	Did the event pass MESEFilter_15?
HighQFilter_17	bool	Did the event pass HighQFilter_17?
ScintMinBias_16	bool	Did the event pass ScintMinBias_16?
event_time	float	time, at which the detector was triggered
RunID	int	RunID of the event
SubrunID	int	SubrunID of the event
EventID	int	EventID of the event
SubEventID	int	SubEventID of the event
sim_weight	float	weight assigned to each event in order to recover the real event rates in histograms, see subsubsection 5.1.5 for detailed explanation
event_no	int	index assigned to each event to match truth table entries to pulse table entries; PRIMARY KEY

Table 3: datasets processed for training, for documentation see [98] (see subsection 5.1 for context)

dataset-id	type	spectrum	energy-range [GeV]
22612	NuGen NuE	E^{-1}	10^6 - 10^8
22613	NuGen NuE	$E^{-1.5}$	10^4 - 10^6

dataset-id	type	spectrum	energy-range [GeV]
22614	NuGen NuE	$E^{-1.5}$	10^2 - 10^4
22644	NuGen NuMu	E^{-1}	10^6 - 10^8
22645	NuGen NuMu	$E^{-1.5}$	10^4 - 10^6
22646	NuGen NuMu	$E^{-1.5}$	10^2 - 10^4
22633	NuGen NuTau	$E^{-1.5}$	10^2 - 10^4
22634	NuGen NuTau	$E^{-1.5}$	10^4 - 10^6
22635	NuGen NuTau	E^{-1}	10^6 - 10^8
22597	Corsika-in-ice 5-component	E^{-2}	$5 \cdot 10^2$ - 10^7

Table 4: final databases created from the datasets from Table 3; They can be found in [72] on the Madison servers. There is a strong imbalance between NuGen and Corsika events (43,542,915 vs. 4,216,310), which is not a problem though, because it is easily mitigated by selecting only parts of the databases for training. (see subsection 5.1 for context)

filename	size [GB]	no of events
22612	236	345,900
22613	125	1,602,308
22614	50	5,125,665
NuE total	411	7,073,873
22644	197	305,043
22645	75	1,714,343
22646	188	22,151,506
NuMu total	460	24,170,892
22633	85	10,173,744
22634	61	1,489,233
22635	288	635,173
NuTau total	434	12,298,150
NuGen total	1305	43,542,915
22615	189	4,216,310
Corsika total	189	4,216,310
total	1494	47,759,225

Table 5: a list of all event classes with explanation and the number of corresponding events in the final databases (see subsection 5.1 for context)

primary	classification	explanation	no. samples
none	background	events that were triggered by background only	21,240,507
none	uncontained_cascade	cascade interactions outside the detector	2,361,747
none	skimming_track	muons passing by outside the detector	376,707
none	skimming_bundle	(muon) bundle passing by outside the detector	15,485
none	skimming_spur	taus passing by outside the detector	129,448
none	other_uncontained	other events without detector primaries	428
multiple	bundle	multiple particles (mostly muons and neutrinos) originating from air showers	1,849,401
multiple	other_multiple	other events with multiple detector primaries	1,616,456
$\overset{(-)}{\nu}$	hadr_cascade	all nu NC interactions	2,477,831
$\overset{(-)}{\nu}_e$	em_hadr_cascade	nue CC interactions	2,477,878
$\bar{\nu}_e$	glashow_electron	glashow resonance; decays into electron	796
$\bar{\nu}_e$	glashow_hadr	glashow resonance; decays hadronically	6,546
$\bar{\nu}_e$	glashow_starting_track	glashow resonance; decays into muon; leaves detector	0
$\bar{\nu}_e$	glashow_contained_track	glashow resonance; decays into muon; decays inside detector	0
$\bar{\nu}_e$	glashow_starting_spur	glashow resonance; decays into tau; leaves detector	33
$\bar{\nu}_e$	glashow_lollipop_hadr	glashow resonance; decays into tau; decays hadronically	366
$\bar{\nu}_e$	glashow_lollipop_em	glashow resonance; decays into tau; decays into electron	102

primary	classification	explanation	no. samples
$\bar{\nu}_e$	glashow_tau_starting_track	glashow resonance; decays into tau; decays into muon; leaves detector	68
$\bar{\nu}_e$	glashow_tau_contained_track	glashow resonance; decays into tau; decays into muon; decays inside detector	23
$\overset{(-)}{\nu}_\mu$	starting_track	numu CC interaction; muon leaves detector	3,298,716
$\overset{(-)}{\nu}_\mu$	contained_track	numu CC interaction; muon decays inside detector	2,249,677
$\overset{(-)}{\nu}_\tau$	inverted_lollipop	nutau CC interaction; tau leaves detector	25,419
$\overset{(-)}{\nu}_\tau$	double_bang_em	nutau CC interaction; tau decays into electron	507,023
$\overset{(-)}{\nu}_\tau$	double_bang_hadr	nutau CC interaction; tau decays hadronically	2,022,532
$\overset{(-)}{\nu}_\tau$	inverted_lollipop_starting_track	nutau CC interaction; tau decays into muon; leaves detector	207,807
$\overset{(-)}{\nu}_\tau$	inverted_lollipop_contained_track	nutau CC interaction; tau decays into muon; decays inside detector	276,983
$\mu^\pm$	throughgoing_track	muon traverses detector completely	4,404,484
$\mu^\pm$	stopping_track	muon enters detector and decays inside it	2,139,305
$\tau^\pm$	throughgoing_spur	tau traverses detector completely	20,311
$\tau^\pm$	lollipop_em	tau enters detector; decays inside it into electron	2,461
$\tau^\pm$	lollipop_hadr	tau enters detector; decays inside it hadronically	8,432
$\tau^\pm$	tau_starting_track	tau enters detector; decays inside it into muon; leaves detector	2,322
$\tau^\pm$	tau_contained_track	tau enters detector; decays inside it into muon; decays inside detector	7
any	other	other events with exactly one detector primary	152

Declaration

I hereby declare that this thesis has been authored by myself and only myself. No contribution to it has been made by others, except for any exchange of ideas that normally happens between colleagues working at the same institute as well as aid provided by my advisor according to his role as such.
I also declare that I have made clear whenever presented information has been taken from external sources and provided suitable references on where to find said work, as far as possible.
I lastly declare that my thesis has not been used before for another assignment, neither in its entirety nor in parts.

place, date signature (Fabian Wohlfahrt)