Universität Münster

BACHELOR THESIS

Training and Performance Characterization of a GNN-based Binary Classifier for Noise Cleaning in IceCube-Gen2

Supervisor:
Prof. Dr. Alexander Kappes

Second examiner:
Prof. Dr. Anton Andronic

A thesis submitted in fulfilment of the requirements for the degree of
Bachelor of Science *at* Universität Münster
by

Jonas Selter

AG Kappes
Institute for Nuclear Physics

August 2025

Declaration of Academic Integrity

I hereby confirm that this thesis on “Training and Performance Characterization of a GNN-based Binary Classifier for Noise Cleaning in IceCube-Gen2” is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

Signature: _____

Date: _____

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

Signature: _____

Date: _____

Contents

1 Introduction

Astronomy is often described as the oldest science practiced by humans. Long before the development of sophisticated scientific methods or mathematical concepts, people looked up at the night sky and used the stars for timekeeping and navigation. Later, technological advances such as the telescope, which collects more light, and the spectrometer, which analyzes the wavelengths of starlight, expanded the range of astronomical observations.

It was not until the 20^{th} century that the era of *multi-messenger astronomy* began. In contrast to traditional astronomy, which relies mainly on light as a source of information, multi-messenger astronomy combines complementary signals — including gravitational waves, cosmic rays, and neutrinos — to gain a more complete understanding of astrophysical phenomena. Instead of observing only the photons produced in an event, researchers now search for other messengers that can be traced back to the same origin. Each messenger has its own advantages and challenges.

This thesis focuses on **neutrinos**, which interact only via the weak interaction. Unlike photons or charged particles, they can escape dense astrophysical environments and travel vast distances through the universe without being absorbed or deflected. While charged particles are bent by magnetic fields and thus lose directional information, and high-energy photons can be absorbed over long distances, neutrinos remain largely unaffected — aside from flavor oscillations. Their weak interaction makes them ideal astrophysical messengers. In particular, high-energy neutrinos are often referred to as a *smoking gun* for hadronic processes, because from the standard model it is expected that they are only produced in significant amounts through interactions involving protons or other hadrons. Unlike gamma rays, which can also be produced in leptonic processes and are easily absorbed, neutrinos offer a clearer and more direct link to cosmic ray acceleration.

However, the very property that makes neutrinos so valuable — their low interaction rate — also makes them incredibly difficult to detect. To overcome this, massive detectors are required. One of the largest of these is the one-cubic-kilometer IceCube detector [1], located at the South Pole.

IceCube detects neutrinos indirectly by measuring Cherenkov light emitted by secondary charged particles. This light is observed by photomultiplier tubes (PMTs) enclosed in optical modules and deployed deep into the Antarctic ice. However, not all detected photons originate from neutrino interactions — a significant portion stems from noise sources, primarily radioactive decays in the glass of the optical modules. This noise can interfere with tasks such as reconstructing the neutrino's direction. Therefore, effective noise-cleaning methods are essential.

Although IceCube currently employs well-established algorithms based on time-space cuts to suppress noise, these methods face limitations when applied to the upcoming IceCube-Gen2 detector [2]. IceCube-Gen2 will have a significantly increased number of optical channels and higher noise rates, requiring more scalable and adaptable approaches.

This thesis investigates an alternative strategy based on **Graph Neural Networks (GNNs)**. Deep learning methods, especially those designed to process structured data such as graphs, are promising

tools for addressing complex, high-dimensional problems like noise classification in neutrino detectors. The specific goal of this work is to assess the suitability of a GNN-based noise cleaning method — using the DynEdge architecture — considering only the IceCube-Gen2 optical array, and not considering the strings of the original IceCube experiment, since the used optical modules are different. For this purpose, based on a previous work for the IceCube-Upgrade [3], a GNN is trained and its performance evaluated for the future IceCube-Gen2 detector. This work can be regarded as a proof of concept, exploring the conditions under which such a model could offer an advantage over classical cleaning techniques.

This thesis begins with a theoretical introduction to the IceCube and IceCube-Gen2 detectors, the key ideas of machine learning, and the GraphNet framework in Chapters 2 and 3. Subsequently, Chapter 4 describes the process of training a GNN-based model for the purpose of noise cleaning. In Chapter 5, the model's performance is evaluated and compared to the traditional IceCube noise cleaning method. The thesis concludes with a summary and outlook in Chapter 6.

2 Neutrino Astronomy

This chapter provides the theoretical foundation for understanding IceCube and the IceCube-Gen2 optical array, offering an introduction to the sources of background noise in their optical modules and the classical approach to noise cleaning in IceCube.

2.1 Neutrinos and Their Properties

In the Standard Model, neutrinos come in three different flavors: ν_e, ν_μ and ν_τ, each one associated with a corresponding charged lepton: the electron, muon and tau. After the postulation made in 1930 by Wolfgang Pauli [4], the first experimental evidence of neutrinos occurred in the 1956 Poltergeist experiment [5]. Neutrinos oscillate in flavor as they propagate, a discovery awarded with the 2015 Nobel prize [6]. A direct consequence of this phenomenon is that neutrinos have mass. Still, the exact mass has not been measured to this day. There is, however, an upper limit for the neutrino mass, resulting from the KATRIN experiment [7]. For instance, the most recent upper value for the electron neutrino mass has been set to $m_\nu \leq 0.45\,\mathrm{eV}$ at a 90% confidence level [8].

Out of all standard model particles, neutrinos are the second most abundant ones in the universe, being outnumbered only by photons. The high-energy neutrinos that IceCube aims to detect are believed to originate mostly from pion and kaon decays, which result from hadronic interactions in some of the most extreme environments in the universe [1].

One of the main goals of neutrino astronomy, and thus of IceCube, is to trace the sources of **high-energy cosmic rays**. These highly energetic particles, most of which are protons, were discovered by Victor Hess in the early 20^{th} century as a form of ionizing radiation [9]. During a balloon flight, Hess observed that the ionization rate increases with altitude, indicating that the radiation must originate from outer space. In a later experiment conducted during a solar eclipse, he demonstrated that this radiation could not be traced back to the Sun, suggesting that it must have galactic or even extragalactic origins.

However, the precise sources of cosmic rays remain one of the great unsolved mysteries in physics. Neutrino astronomy offers a promising way to address this question. It is believed that high-energy cosmic rays are produced in some of the most extreme environments in the universe, such as active galactic nuclei (AGNs). Since hadronic interactions are expected to occur in these environments, the production of high-energy neutrinos is a natural consequence.

While neutrinos are electrically neutral and do not interact via the electromagnetic or strong forces, and their small mass makes them barely affected by gravity, they do interact through the weak interaction. In the context of neutrino detection in high energy ranges, deep inelastic scattering processes are most relevant, which can occur for both charged and neutral current. A neutrino ν_ℓ (antineutrino $\bar{\nu_\ell}$), with ℓ indicating the corresponding lepton flavor, interacts with a nucleon N upon collision, exchanging either a W-boson (Charged Current, CC) or Z-boson (Neutral Current, NC). In the former case, the interaction yields a lepton ℓ^- (antilepton ℓ^+) and a hadronic cascade X. The lepton ℓ can be an electron e, a muon μ, or a tau τ, depending on the flavor of the interacting neutrino. In the latter

case, the neutrino's flavor remains unchanged, and the interaction produces a hadronic cascade [10]:

$$\overset{(-)}{\nu}_\ell + \mathrm{N} \xrightarrow{W^\pm} \ell^\mp + \mathrm{X}\ (\mathrm{CC}), \qquad \overset{(-)}{\nu}_\ell + \mathrm{N} \xrightarrow{Z^0} \overset{(-)}{\nu}_\ell + \mathrm{X}\ (\mathrm{NC}).$$

2.2 Principles of Neutrino Detection

While the low interaction rate of high-energy neutrinos is a major advantage, it also creates significant challenges for their detection, the reconstruction of their paths, and ultimately the study of their origins. To overcome this, large-volume detectors are used to observe high-energy neutrinos.
Neutrinos are detected indirectly by means of secondary charged particles produced in neutrino interactions via the Cherenkov effect [12]: If a charged particle propagates in a dielectric medium at a speed faster than the speed of light in this medium, the wavefronts produced by the rapidly moving particle interact constructively, creating Cherenkov radiation. The conically shaped waves are carried along the particles path, and the particle's speed can be characterized by the angle θ between the path traveled and the wavefront of the emitted light. This can be seen in figure 2.2.1, where the angle θ is given by:

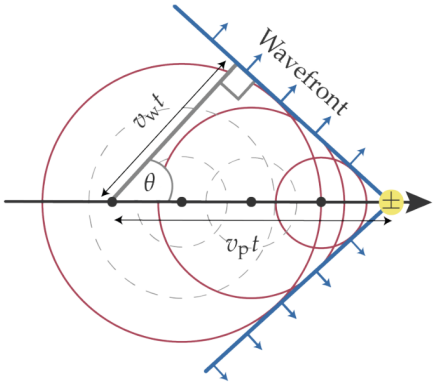

Figure 2.2.1: Wavefronts of a particle travelling at a speed beyond the speed of light in a medium interfere constructively, producing Cherenkov radiation. The angle θ depends on the particle's velocity. Image taken from [11].

$$\cos\theta = \frac{v_\omega t}{v_p t} = \frac{1}{\beta n},$$

where v_ω is the velocity of the Cherenkov wavefront, t is the time, v_p is the velocity of the particle, n is the refractive index of the medium, and $\beta = v_p/c$ is the particle's velocity in units of speed of light.

In addition to requiring a dielectric medium, a neutrino detection medium must also be transparent so that Cherenkov photons can propagate and be detected. Water in all its forms is an excellent candidate, as it meets both criteria and occurs naturally in large quantities on Earth, enabling the construction of kilometer-scale detectors.

2.3 The IceCube and IceCube-Gen2 Detectors

The IceCube Neutrino Observatory [1] is an astrophysical experiment located at the South Pole, aiming to detect cosmic neutrinos. It has been in operation since 2011. The detector consists of 86 strings deployed into the Antarctic ice at depths between 1450 and 2450 meters, with 60 attached Digital Optical Modules (DOMs) on each string. In total, the instrumented volume of the detector covers approximately one cubic kilometer. Each DOM contains a large photomultiplier tube (PMT), making it sensitive enough to detect single photons. In the coming years, several extensions of the detector are planned, rendering it more sensitive to neutrinos at both the lower and higher energy regimes. In addition to the IceCube Upgrade [13], which is scheduled for deployment during the Antarctic summer of 2025/26, one major planned extension is IceCube-Gen2 [2]. Figure 2.3.1 shows a schematic view of the layout in which the IceCube-Gen2 strings will be deployed.

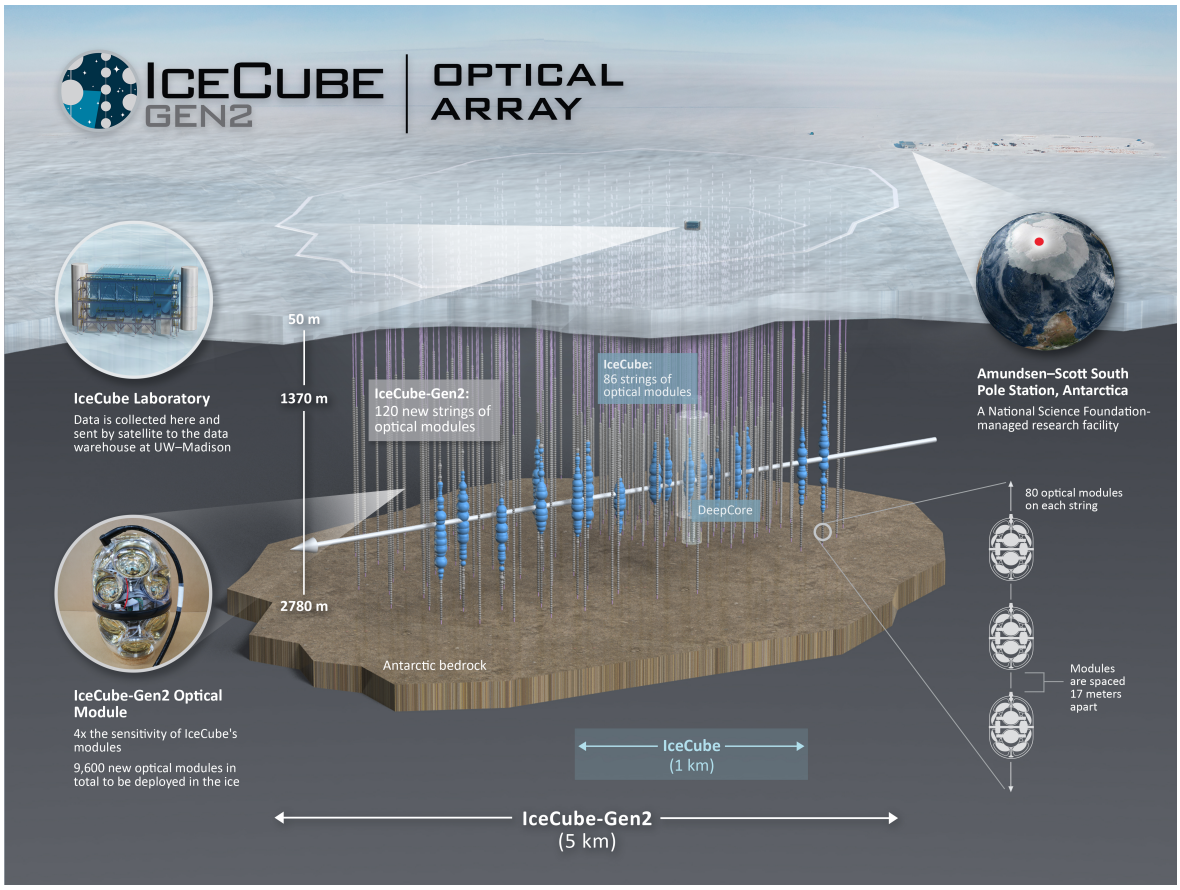

Figure 2.3.1: Diagram of the string positioning in the IceCube-Gen2 optical array. The optical module shown on the left-hand side, the Gen2DC-16, is displayed in an exploded view in Figure 2.3.2. Image courtesy of the IceCube Collaboration.

For the IceCube-Gen2 optical array alone, the detector volume will increase to approximately eight cubic kilometers. Furthermore, a radio array and a surface air shower array will be added. The main goals of IceCube-Gen2 are the following [14]:

- Resolve the high-energy neutrino sky from TeV to EeV energies.
- Investigate cosmic particle acceleration through multi-messenger observations.
- Reveal the sources and propagation of the highest energy particles in the Universe.
- Probe fundamental physics with high-energy neutrinos.

In contrast to the Digital Optical Modules (DOMs) which are already deployed and being used in IceCube, the IceCube-Gen2 optical array will be featuring advanced modules containing multiple PMTs, allowing for at least four times more effective area and intrinsic angular resolution. One design candidate is the Gen2DC-16. This candidate features 16 4-inch diameter PMTs within an elongated glass vessel. An exploded view can be seen in Figure 2.3.2. The final module design to be deployed in the IceCube-Gen2 optical array has not yet been finalized.

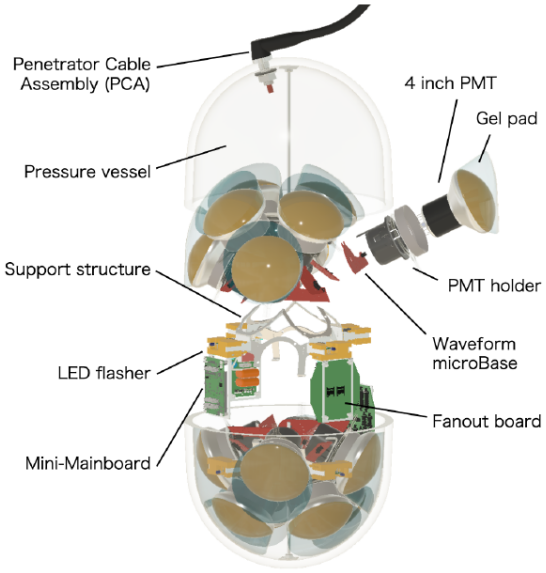

Figure 2.3.2: Exploded view of the Gen2DC-16. Image taken from [15].

2.4 Background Noise in IceCube and IceCube-Gen2

The light observed in IceCube-PMTs does not only come indirectly from neutrinos or atmospheric muons, but there is background light, too, which is traceable to other sources and accounts for noise in measurements. Since the antarctic ice is very old, it is not a source of background. However, PMT dark rate and, above all, radioactive decays in the glass vessels surrounding the optical modules are.

There are different types of background noise which are detected in the IceCube experiment, and will be in the future extensions—including the IceCube-Gen2 optical array. The main source of noise for the IceCube-Gen2 optical array is expected to be coming from radioactive decays in the glass of the optical modules. These originate from different radioactive isotopes in the material, such as ^{40}K or isotopes from the three natural decay chains ^{238}U, ^{235}U and ^{235}Th [11]. In the decay, particles are emitted that can produce Cherenkov radiation, creating noise hits on the PMTs. In addition, the energy deposited by these particles can induce scintillation and thus generate further background noise. For IceCube-Gen2, the described noise is currently modelled using detailed Geant4 simulations [16].

The noise is typically removed because it complicates the directional reconstruction, especially of low-energy events. The traditional noise-cleaning method used in IceCube is called Seeded RT (SRT) cleaning [17], where R and T indicate that the cuts are based on spatial and temporal correlations. The algorithm first selects photon hits that satisfy a preliminary local coincidence condition. Based on the spatial and temporal distances from these hits—referred to as seed pulses—it then applies cuts to identify noise. In the current IceCube-Gen2 simulations, two pulses are considered to be in local coincidence if their time difference is within $0.01\,\mu s$ for hits in the same optical module but different PMTs, or within $0.2\,\mu s$ for hits in neighboring optical modules.

3 Introduction to Machine Learning

This chapter gives a general overview of machine learning and explains the general workflow when working with deep learning models. Moreover, the process of training a Neural Network is explained.

3.1 Neural Networks

A fundamental concept in machine learning is the **neuron** [18], which is defined as:

$$Y = f\left(\sum w_i \cdot x_i + b\right),$$

where x_i is an input vector, w_i a weight vector, and b is a bias term. The weighted sum of inputs and bias is passed through a non-linear function f, which is commonly referred to as the **activation function** in the context of machine learning.
Some commonly used activation functions include:

- ReLU(x) [19]: ReLU $= \max(0, x)$.
- Sigmoid [20]: $\sigma(z) = 1/\left(1 + e^{-z}\right)$.

Neurons are the basic building blocks of layers. When multiple layers of neurons are stacked, this forms a Multilayer Perceptron (MLP), as illustrated in Figure 3.1.1. In this figure, each bubble represents a neuron. All layers between the input and the output are referred to as hidden layers. As the number of hidden layers increases, the network can learn progressively more complex patterns relevant to the target task: the early layers typically extract fine-grained (local) features, while the deeper layers capture higher-level (global) structures. The total number of hidden layers is a hyperparameter of the model and is typically chosen through trial and error.

The figure also depicts that each neuron in one layer is connected to every neuron in the subsequent layer. These connections, represented by lines, contain the model's weights which, together with the bias (there is only one bias per neuron), make up the trainable parameters of the network. In general, if there is more than one hidden layer, a network of neurons is referred to as a **Deep Neural Network**.

3.2 Training a Neural Network

In the previous section, neural networks were introduced, which are capable of mapping a given set of input values to corresponding target outputs. However, in order to produce accurate predictions, the network's parameters must be adjusted through an iterative process known as *training*, during which the model is exposed to a large number of training samples with known target outputs.

In order to train a model, it is crucial to define a quantity that reflects the inaccuracy of its predictions. In the context of machine learning, this quantity is known as the *loss*, and it is computed by a loss function that takes the predicted and target outputs as input.

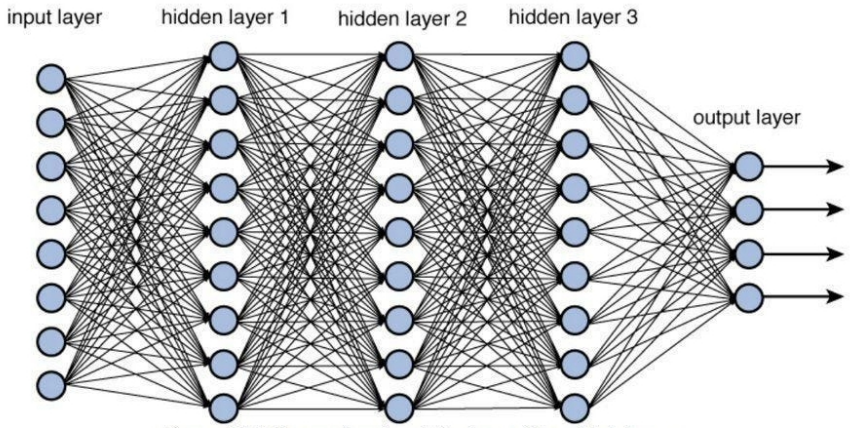

Figure 3.1.1: Visualization of a Deep Neural Network with three hidden layers. Explanation can be found in the text. Image taken from [21].

The choice of loss function depends on the assumed likelihood model for the target varible. In the case of this thesis—a binary classification task—the predicted values represent scores between 0 and 1, where 0 indicates a noise-induced hit and 1 indicates a physics-induced hit. The target variable follows a Bernoulli distribution, and the appropriate loss function is the binary cross-entropy, which corresponds to the negative log-likelihood of the Bernoulli distribution. This can be written as follows [22]:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(\hat{p}_i) + (1-y_i)\cdot \log(1-\hat{p}_i)\right],$$

where N is the total number of training examples, ($y_i \in \{0,1\}$ is the true label of the i-th sample, and ($\hat{p}_i = p_\theta(y_i = 1 \mid x_i)$ marks the predicted probability that the input features x_i for the given node belong to the positive class, given the model parameters θ. The notation $\hat{p}_i$ means that this is an inferred quantity estimated by the model, rather than the true probability. The factor $-\frac{1}{N}$ ensures that the loss is averaged over all samples and remains positive.

To optimize the model toward this minimum, the gradient of the loss can be computed over the entire training dataset at once. This gradient indicates the direction of steepest descent in the loss landscape. Updating the weights and biases in the opposite direction of the gradient is known as *gradient descent*, with the update rule:

$$\theta = \theta - \eta \nabla_\theta J(\theta),$$

where θ denotes the model parameters, $J(\theta)$ is the loss function (binary cross-entropy in this thesis), ∇_θ is the gradient with respect to the parameters, and η is the learning rate.

The equation can be interpreted as follows: the gradient of the loss function indicates the direction of steepest ascent, while its negative $-\nabla_\theta$ gives the direction of steepest descent used to update the parameters. Once this direction is determined, the remaining question is how far the parameters should be adjusted. This step is controlled by the learning rate η, which is one of the key factors in successfully training a deep learning model. If η is too small, the model converges very slowly and risks becoming trapped in a local minimum rather than the global one. Conversely, if η is too large, the parameter updates can overshoot the minimum, leading to unstable or chaotic behavior. In such cases, the model may fail to converge and oscillate around regions of high loss.

There are, however, several variations of this method. One widely used variant is *stochastic gradient descent (SGD)* [23]. Unlike the approach described above—commonly referred to as *batch gradient*

descent—SGD does not use the entire dataset to update the model parameters. Instead, it selects a single minibatch at random for each update step. This enables faster and more computationally efficient optimization, particularly for large datasets. It can be expressed as:

$$g = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L\left(x^{(i)}, y^{(i)}, \theta\right),$$

where g is the gradient vector, m' is the size of the mini-batch, ∇_θ marks the gradient operator with respect to the model parameters θ, i are the samples in the minibatch, and $L\left(x^{(i)}, y^{(i)}, \theta\right)$ is the loss for a given training sample $\left(x^{(i)}, y^{(i)}\right)$ given the current parameters θ. By means of the SGD algorithm, the network parameters are adjusted with the goal of minimizing the loss function of the model. In this thesis, SGD is used.

To address this, some optimizers incorporate a momentum term that adds a velocity-like component to the parameter updates. This helps smooth the optimization path and prevents abrupt changes in direction. The optimizer used in this work is Adam (Adaptive Moment Estimation) [24], which combines momentum with adaptive learning rates for each parameter.

In addition, whenever the model parameters are updated, it is standard practice to monitor the loss on a separate, unseen dataset, the so-called validation dataset. During training, the validation loss is expected to decrease alongside the training loss. However, if the validation loss starts to increase at some point, training is typically stopped, as this indicates overfitting. Overfitting occurs when the model stops learning general patterns from the training data and instead begins to memorize the specific examples. This leads to continued improvement on the training dataset, but reduces the model's ability to generalize to unseen data, which is the actual goal of training.

3.3 Graph Neural Networks

When selecting a neural network architecture, it is crucial to consider the symmetries of the input data. Exploiting such symmetries allows the model to generalize more effectively and to learn relevant features more efficiently. This is because if the symmetries are considered, the model will be more efficient and will be able to achieve a better performance with a smaller number of parameters, leading to less memory usage and a shorter inference time. A well-known example is the use of Convolutional Neural Networks (CNNs) for image processing [25]. While a multilayer perceptron (MLP) can technically be trained to classify images, its performance is typically poorer because it does not incorporate translation symmetry. CNNs, by contrast, are designed to be approximately *translation-invariant*: they can recognize features regardless of their position in the image, thanks to their convolutional architecture which exploits the translational structure of image data.

A Graph Neural Network (GNN) can be used to efficiently train on graph-structured data. In general, a graph consists of *nodes* connected by *edges*. In this work, each node contains information about the PMT which was hit, including its position, direction, time, and charge, while the edges encode relationships between nodes. An illustration of this representation is shown in Figure 3.3.1.

GNNs apply an idea similar to that of CNNs to graph-structured data rather, than grid-like images. In some cases, GNNs can also achieve translation invariance if the input features are given in relative coordinates. In this thesis, however, the absolute positions of PMT hits are used, so the relevant symmetry is *permutation invariance*: the network's output should not depend on the ordering of the

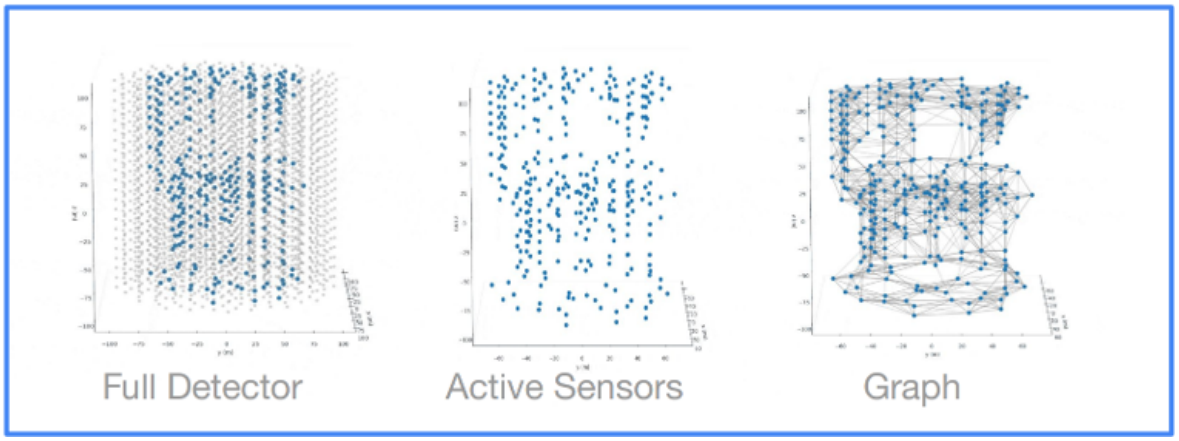

Figure 3.3.1: Different stages of representing IceCube events as point-cloud graphs. The image was created for an IceCube simulation but is equally applicable to the IceCube-Gen2 optical array. **Left:** Full detector view showing strings and DOMs, with active PMTs highlighted in blue. **Center:** Only active PMTs are shown. **Right:** Schematic graph representation, where hits on active PMTs are modeled as nodes, and connections between them as edges. Credits: Jan Weldert and Rasmus Ørsøe.

input nodes. Another motivation for using a GNN in this application is its ability to handle sparse and variable-sized data. Unlike CNNs, which require a fixed input dimension, GNNs naturally accommodate the varying number of pulses in different IceCube-Gen2 events. Since the task here is to classify individual pulses based on their local neighborhood, the graph-based architecture is a natural fit.

DynEdge

The specific GNN architecture used in this thesis is called **DynEdge** [3]. It is based on the EdgeConv convolutional operator. Given a node n_j with node features x_j, the convolved features $\tilde{x}_j$ are computed as follows:

$$\tilde{x}_j = \sum_{i=1}^{N_{\text{neighbors}}} \text{MLP}(x_j, x_j - x_i),$$

where the multi-layer perceptron (MLP) processes both the features of node j and the difference between node j and its neighboring nodes i. The upper limit of the sum, $N_{\text{neighbors}}$, is set to 8 in this work. A diagram of the DynEdge model architecture is shown in Figure 3.3.2.

DynEdge uses a dynamic EdgeConv, meaning that the 8 connections to the nearest neighbors are recomputed throughout the forward pass of the network. After the first EdgeConv layer, the model has learned features, which live in an abstract space; then, the connections to the nearest neighbors are updated based on distance in this learned space, not just physical distance. Every time the model passes through an EdgeConv, the graph is rebuilt based on the current node features.

3.4 Graphnet

GraphNet [26] is an open-source library designed for deep learning applications in neutrino telescopes. It has been shown that machine learning performs remarkably well in neutrino event reconstruction and classification—even without explicitly considering detector geometry or the detection medium [3]. This means that different neutrino observatories produce data of similar structure, making it possible to share models and methods across experiments. GraphNet leverages this by offering

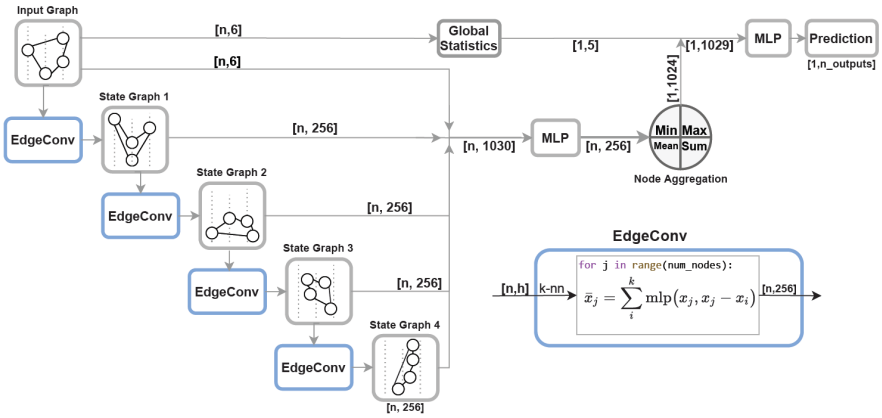

Figure 3.3.2: Diagram of the DynEdge architecture. n refers to the number of pulses, and h denotes an arbitrary number of feature columns. Image adapted from [3].

a unified and modular framework that enables collaboration and code sharing between researchers. It is written in `Python`, and built on `torch` [27], specifically on the `PyTorch Lightning` [28] framework.

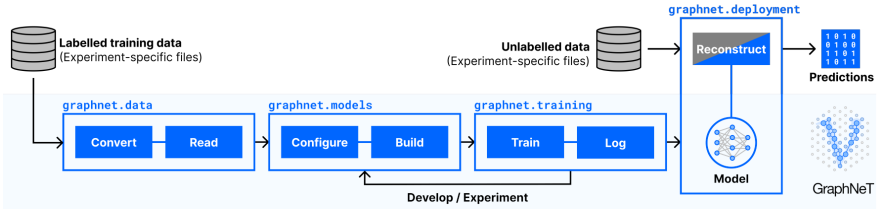

Figure 3.4.1: Diagram representing the GraphNet workflow. Image taken from [26].

Figure 3.4.1 illustrates the general workflow when using GraphNet. The diagram follows a left-to-right structure: it starts with labeled training data. In the case of IceCube, this often means `i3` files, which are highly specific to IceCube and require the IceTray [29] environment for processing. This will be explained in more detail in Chapter 4.2.1. This labelled data contains information about hits and corresponding truth labels, such as particle type, energy, or direction. The following steps in the workflow are summarized into groups:

- `graphnet.data`: This marks the first step when working with labelled data as described above. First, the raw data must be converted into a format GraphNet can use, which will later be referred to as ML-compatible format in chapter 4.2.1. In this case, the format used will be SQlite.
- `graphnet.models`: This is where the model configuration takes place. Here, a model architecture—in this case, DynEdge— is chosen, and the model's hyperparameters can be defined, such as the number of layers. After this step is completed, the actual PyTorch Geometric model is built based on the configuration.
- `graphnet.training`: Here, the labelled graphs which are given as an input are used to actually train the model to do the assigned task (in this case, binary classification). `Log` refers to saving the model checkpoints, which are needed for analysis in later steps.
- `graphnet.deployment`: After the model is successfully trained, it can be applied to unseen data, which is unlabelled, and make predictions, which are the model output. This step is also called inference.

4 Training a GNN-based model for Noise Cleaning in IceCube-Gen2

The use of a Graph Neural Network (GNN) for noise cleaning has been demonstrated in the context of the IceCube Upgrade [3]. The goal of this thesis is to apply the GNN architecture used in that work, DynEdge [3], to the optical array of IceCube-Gen2 and to train it for noise cleaning in IceCube-Gen2 simulations with the Gen2-DC16 [15] as the baseline optical module, assuming that Vitrovex [30] glass will be used for the pressure vessel. The following chapter describes the process of training the GNN-based model.

4.1 Monte Carlo Training Data

The Monte Carlo simulation used to train the model is taken from IceCube dataset 22830, containing exclusively ν_μ simulations. This way, the model will see both cascade events and track-like events in training, since ν_μ produce both neutral and charged current interactions. There are roughly two million events in an energy range from $1\,\mathrm{TeV}$ to $50\,\mathrm{PeV}$, following a power-law distribution of the form $E^{-\gamma}$, where the training dataset is characterized by the spectral index $\gamma_{22830} = 1.5$. As explained in section 2.4, the baseline optical module is the Gen2-DC16, and the noise simulation is based on detailed Geant4 simulations of radioactive decays in the optical module's glass [16], considering the Vitrovex [30] manufacturer.

4.2 Processing the Simulation Files

4.2.1 Preprocessing of i3 files

The Monte Carlo simulations in the 22830 and 22831 datasets are stored in `i3` files. The `i3` format was specifically developed for IceCube simulations, and working with it requires the installation of the `IceTray` framework [29]. This framework is used for handling simulation data and processing `i3` files. As a result, the preprocessing steps were performed on IceCube's Condor-based computing cluster [31], where `IceTray` is pre-installed.

The `i3` files used in this context contain multiple frames, with each frame labeled `Q` representing one simulated event. These `Q` frames follow a dictionary-like structure, where different keys and values store various types of information about the simulated particles.

Removing On-board Noise Cleaning

The pulses in this script were initially processed with an experimental on-board noise cleaning. This step was included in the simulations to investigate preliminary noise suppression. However, this on-board noise cleaning was proven to remove a significant amount of physics pulses, which is why it was decided not to use it anymore. In addition, all events with all their noise pulses are wanted for training. Therefore, the reconstructed charge-time-pulses under the key `I3RecoPulseSeriesMapExtensions` were reprocessed from the Monte Carlo pulses of the previous step, which can be found under the

key `I3MCPulseSeriesMapExtensions`.

Setting Truth Flags

To understand how the truth flags are set, it is helpful to first examine the simulation chain used in IceCube simulations. After neutrino interaction, particles propagating through ice are simulated, including their Cherenkov light. Whenever a photon is detected by a PMT, a Monte Carlo Photo Electron (MCPE) is generated, which is traceable to the particle it originated from since it carries its particle ID. In addition, noise hits are added, which are sampled from pre-computed simulations. The MCPEs associated with noise hits carry the event-ID 0. In the next step, the MCPEs generated in the simulation are used to simulate so-called MC pulses. This is done by sampling for each MCPE an expected measured charge by the PMTs from a SPE (single photo electron) template, and also adding a time jitter, modifying the time of detection, and finally the MC pulses are converted into so-called reco-pulses by redefining them in the corresponding IceCube dataclass, applying a 0.25 photo-electron (PE) charge threshold on pulses and including a local coincidence flag. At the time being, given the current status of the IceCube-Gen2 simulation, actual waveform simulation is not performed. This makes labeling easier, as signal and noise hits can be unequivocally distinguished. Therefore, the current training allows for perfect labeling: Since each MCPE has its own label, the corresponding reco-pulse carries the same label, matching it unequivocally to either a physics or noise hit.

For actually setting the truth flags, an `i3`-module was written. Inside an `i3` file, it iterates over the `I3RecoPulseSeriesMapExtensions`, which contains the reco pulses of IceCube-Gen2. It checks the `I3RecoPulseSeriesMapParticleIDMap` to see which pulses are linked to a non-noise particle, which is indicated by a particle ID different to 0. Based on this, a `truth_flags` map is generated, where 1 corresponds to a physics hit, and 0 to a noise hit.

Converting i3 files into ML-Compatible Format

Before the data can be used as an input for machine learning, it must be converted into one of the two formats suitable for GraphNet, i.e. Parquet or SQLite. In this work, the SQLite format was chosen, which is a database format structured in tables. From the i3 files, information on PMT position, PMT direction, charge, time, and the truth flags are included for each reco-pulse. The total size of the dataset used for training in SQLite format is $1.1\,\mathrm{TB}$.

4.2.2 Calculating Range of Input Features

The input features used for training include the charge, time, and position of each particle. Before they can be used, the features must be normalized to ensure they lie within a comparable range so that no single feature biases the model during training. Moreover, in some cases this is crucial for the model to be able to converge at all. The goal of the normalization is to scale all values into the range $(-1, 1)$.

First, the charge range of the events is evaluated. In the simulation, one Monte Carlo photoelectron (MCPE) should have an average charge of 1; however, in some cases, charges can reach values of 100 or even $1{,}000$. This is because, in the simulation, MCPEs occurring within $2\,\mathrm{ns}$ on the same PMT are merged. Therefore, instead of using the raw charge, the logarithm of the charge is taken. To fit all charge values into the range $(-1, 1)$, the data would technically need to be normalized by dividing by the maximum absolute charge value. However, if outliers are present, it is undesirable for them to dominate the scaling and compress the majority of values into a narrow range. To avoid this, the 95^{th} percentile of the charge distribution is calculated, and all values are divided by this percentile. This

ensures that 95% of the inputs lie within the normalized range. In this case, the procedure yielded an upper charge value of approximately 2.76, leading to the choice of dividing the charge by 3.
Next, the time values must be normalized into the same range $(-1, 1)$. For this, the difference between the earliest and latest pulses in each event is examined. The earliest pulses reach down to $-12\,500\,\mathrm{ns}$, while the latest pulses extend up to approximately $55\,000\,\mathrm{ns}$, yielding a total span of $67\,500\,\mathrm{ns}$. To fit all times into the desired range, they are divided by half of this span and shifted to be symmetric around zero by subtracting the mean after division:

$$t_{\text{new}} = \frac{t}{33750} - 0.63\,.$$

To normalize the PMT positions, each component was divided by 2000 to fit the values into the range $(-1, 1)$. The scaling was performed globally, i.e. applied to x, y, and z together rather than scaling each axis separately. This preserves the Euclidean distances between PMTs in 3D space, which would be distorted if the coordinates were scaled with individual factors.

The PMT direction is provided in vector form (dx, dy, dz) by default. As these values are already bounded within $(-1, 1)$, no further normalization is required.

4.3 Training DynEdge

After the preprocessing of the input data, the training of the model was performed on the PALMA cluster [32], a HPC system with more than 3000 processor cores and a computing power of 30 Teraflops.

With regard to computational resources, limitations were made in the training process. Since it was observed that events containing a very large number of pulses require great amounts of memory in training, a cut was performed to train only on events containing 1×10^5 pulses or less. Since the average number of noise pulses per event is expected to be in the range of 3000, events with more than 10^5 pulses have already a relatively high signal-to-noise ratio; still, the possibility to perform inferences on events with more pulses will be discussed in Chapter 5.

As described previously, the loss on the validation dataset is monitored throughout training after each epoch. An epoch corresponds to one complete pass through the training data. If the validation loss does not decrease for several consecutive epochs, training is stopped to prevent overfitting and save computation time. This is implemented using the early stopping technique, which is assigned a patience value. In this work, the patience was set to 5, meaning the model stops training if no improvement is observed for five epochs.

The training was performed on an H200 SXM GPU with $141\,\mathrm{GB}$ of VRAM and HBM3e memory [33]. For training, a batch size of 8 and a learning rate of 10^{-4} were used. The training stopped after 23 epochs because the early stopping condition was satisfied. The entire training process, including data loading, all training epochs, validation, and model saving, took $161.13\,\mathrm{h}$ wall-clock time.

5 Performance

After noise cleaning has been applied, its performance is evaluated in this chapter. An ideal classifier would remove all noise pulses while retaining all physics pulses. By comparing the predictions $\hat{y}$ with the ground-truth labels y, each hit can be assigned to one of four categories:

- **True Positive (TP)**: A physics hit correctly classified as physics, $\hat{y} = 1,\ y = 1$.
- **True Negative (TN)**: A noise hit correctly classified as noise, $\hat{y} = 0,\ y = 0$.
- **False Positive (FP)**: A noise hit incorrectly classified as physics, $\hat{y} = 1,\ y = 0$.
- **False Negative (FN)**: A physics hit incorrectly classified as noise, $\hat{y} = 0,\ y = 1$.

To characterize the performance of the classifier, the following rates are computed:

- **True Positive Rate (TPR)**: $\mathrm{TPR} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$
- **False Positive Rate (FPR)**: $\mathrm{FPR} = \mathrm{FP}/(\mathrm{FP} + \mathrm{TN})$

The denominators in these expressions correspond to the total number of actual physics hits ($y = 1$) and actual noise hits ($y = 0$), respectively. The behavior of an ideal classifier described above corresponds to a TPR of 1, and an FPR of 0.

For testing the trained model, IceCube dataset 22831 is used. Like dataset 22830, it consists of ν_μ simulations within the same energy range, but with a higher proportion of high-energy events, since it has a spectral index of 1.1. To ensure comparability, once again only events with up to 10^5 pulses were considered, both for GNN and SRT. For testing, not the entire dataset with two million pulses was used, but rather about $10\,\%$ of the total dataset with a size of $107\,\mathrm{GB}$ in the SQLite format.

5.1 Model Inference Outputs

The final deployment of the model was not performed within IceTray. Instead, the inference results are stored provisionally in a CSV file containing three columns:

- `event_no`: Indicates the event number and can be used to group all lines belonging to the same event. This allows retrieval of the number of pulses per event and matching the corresponding entries accordingly.
- `truth_flag`: Denotes whether the corresponding hit is caused by a physics particle (1) or a noise particle (0). This information is based on the truth labels, as described in Section 4.2.1.
- `target_pred`: Contains floating-point values in the range (0, 1), representing the model's prediction for the corresponding hit. Values closer to 0 indicate a higher confidence that the hit is noise, while values closer to 1 suggest a physics hit.

As a first step, the `truth_flag` and `target_pred` columns are used to generate one histogram for physics and noise hits each, which are normalized independently, which can be seen in Figure 5.1.1. It provides an overview of the model's output, revealing that the model classifies physics

hits and noise hits with high confidence. A logarithmic y-axis is used to enhance visibility across several orders of magnitude.

As expected for a well-trained model, most hits accumulate in the outermost bins (0–0.1 and 0.9–1), corresponding to predictions where the model is very confident that a hit is noise (0) or physics (1). Notably, the number of correctly identified noise hits in the lowest bin exceeds that in the highest bin by a factor of roughly 10^3, with an even stronger ratio for physics hits in the opposite bins.

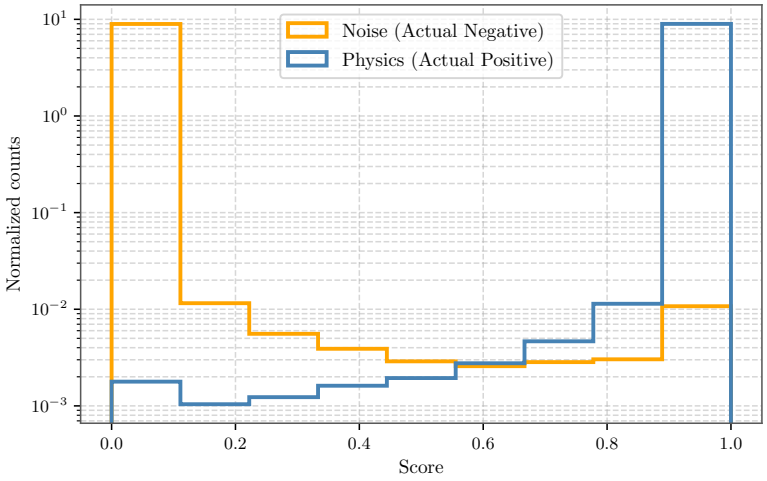

Figure 5.1.1: Comparison of the model output during inference for physics and noise pulses (binned). The two histograms were normalized independently. The model's output values range from 0 to 1, where 0 corresponds to a confident noise classification and 1 to a confident physics classification. Note that the y-axis is shown on a logarithmic scale.

5.2 ROC Curve

A common way to assess the performance of a classifier is with the Receiver Operator Characteristic (ROC) curve. In this plot, the true positive rate (TPR) is shown on the y-axis and the false positive rate (FPR) on the x-axis. Each point on the curve corresponds to a different classification threshold. Since the predictions made by the GNN are continuous values between 0 and 1, the final classification depends on the chosen threshold. For example, a hit may be classified as physics (1) if the prediction exceeds 0.5, or a stricter threshold such as 0.9 can be applied.

A perfect classifier would always assign a value of 1 to physics hits and 0 to noise hits, resulting in a TPR of 1 and a FPR of 0 for all possible thresholds. In this case, the ROC curve would reach the top-left corner of the plot shown in Figure 5.2.1. In contrast, a random classifier would have the same probability of classifying hits correctly or incorrectly, leading to equal TPR and FPR values across all thresholds. This results in a diagonal ROC curve.

The ROC curve of the GNN trained in this thesis is shown in Figure 5.2.1. Additionally, a zoomed-in version is presented in Figure 5.2.2. In this version, the curve is divided into three categories: events with a pulse count between the average number of noise pulses and one standard deviation above (blue), events below this range (orange), and events above one standard deviation (green).

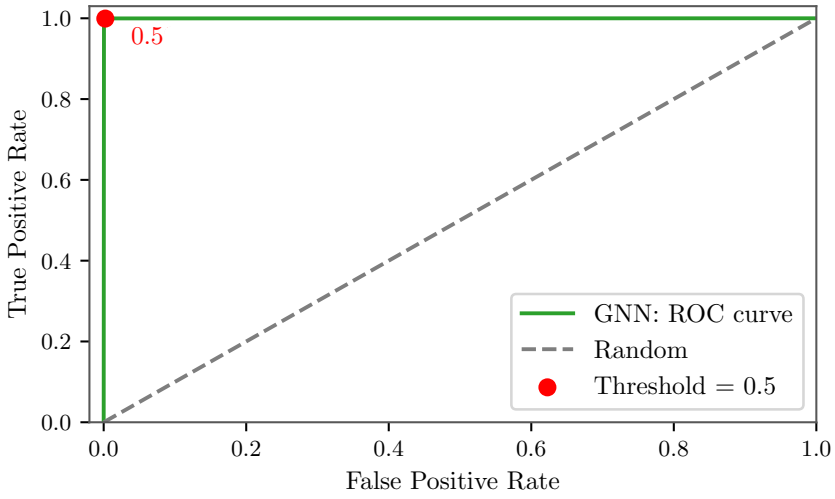

Figure 5.2.1: Receiver Operator Characteristic (ROC) curve of the trained GNN model. For better visibility, a zoomed-in version is shown in Figure 5.2.2. The diagonal line represents a random classifier, which corresponds to the line of equality.

A common way to quantify classifier performance from a ROC curve is by computing the area under the curve (AUC), where a value of 1 corresponds to perfect classification. The AUC values for the three pulse-count ranges shown in Figure 5.2.2 are listed in the legend. In addition to serving as a performance metric, the ROC curve can also guide the choice of a classification threshold. As illustrated in the figure, a threshold of 0.5 appears at different positions on the ROC curve depending on the pulse range.

For this plot, three pulse ranges were chosen: 0–2,900 pulses, 2,900–3,300 pulses, and 3,300–100,000 pulses. This choice is motivated by the fact that the average number of noise pulses per event in the training dataset is approximately $\mu \approx 2{,}940$ with a standard deviation of $\sigma \approx 406$. Events with very low pulse counts are shown in blue, those in the noise-dominated region near $\mu \pm \sigma$ in orange, and events with pulse counts above $\mu + \sigma$ in green.

Even though the GNN performs well in all three categories—as also emphasized by the combined ROC curve of all pulse counts in Figure 5.2.1—the zoomed-in version reveals that the model's TPR is slightly lower in the region of very low pulse counts. This behavior is reasonable: when the number of physics pulses is small compared to the average number of noise pulses, it becomes inherently more challenging for the model to correctly identify the few physics hits.
Ideally, one would aim for a threshold corresponding to a point near the upper left corner of the curve to balance true positive and false positive rates. However, no single threshold achieves an optimal trade-off across all pulse ranges. It is also important to note that Figure 5.2.2 is significantly zoomed in; while the points do not lie exactly in the upper left corner, they are extremely close. For context, the full ROC curve with axes ranging from 0 to 1 is shown in Figure 5.2.1, where it becomes clear that the AUC approaches that of an ideal classifier. This renders the exact threshold choice less critical overall. As seen earlier in Figure 5.1.1, the model is highly confident in its predictions. Considering all these factors, a classification threshold of 0.5 was chosen for characterizing the performance in this work.

For SRT cleaning, it is not possible to present a ROC curve, since it is not a classifier that produces continuous predictions. Instead, it applies fixed cuts based on temporal and spatial distances. Because of this, there is no adjustable threshold, and therefore no ROC curve can be created.

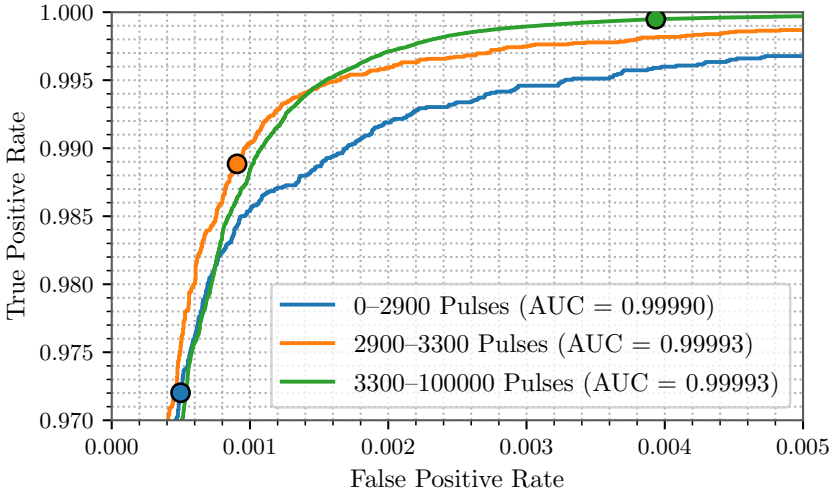

Figure 5.2.2: ROC curves for different ranges of the number of pulses per event (zoomed in). The dots on each curve mark the chosen threshold value 0.5. Each point on a curve corresponds to a different threshold, representing the trade-off between a low False Positive Rate (FPR) and a high True Positive Rate (TPR). When choosing a threshold, the goal is to approach the upper left corner of the plot, representing the best balance between retaining physics hits and rejecting noise hits. In this case, however, trade-offs had to be made depending on the pulse count, which led to this conservative threshold choice. The area under the curve (AUC) serves as a performance metric for the classifier; perfect classification corresponds to an AUC of 1.

5.3 Classical SRT vs GNN Noise Cleaning Comparison

The rates introduced at the beginning of this chapter have been calculated for both models, with a focus on the True Positive Rate (TPR) and False Positive Rate (FPR). Note that TPR and the False Negative Rate (FNR) are complementary, meaning they sum to one, as do FPR and the True Negative Rate (TNR). This relationship allows the difference to 1 to be visualized more clearly.

To apply SRT cleaning to IceCube-Gen2 events, the temporal and spatial cuts from IceCube must be scaled to the larger average inter-string spacing of the IceCube-Gen2 optical array, as described in Section 2.4.

True Positive Rate

Since physics pulses that remain after cleaning correspond to physics correctly identified as such, and physics pulses before cleaning represent the actual positives, the True Positive Rate (TPR) is calculated as the number of physics pulses after cleaning divided by the number of physics pulses before cleaning. Achieving a TPR as close to one as possible is crucial, as a low TPR means that physics hits are misclassified as noise and consequently discarded.

Figure 5.3.1 compares the TPRs achieved by SRT and GNN cleaning as a function of the total number of pulses per event. For both models, the 10th, 50th, and 90th percentiles are plotted to visualize not only the median performance but also the spread, with 80% of the data points lying between the upper and lower bounds. In addition, the mean number of noise pulses per event, as well as one standard deviation $\pm\sigma$, is indicated by vertical lines. This highlights that both algorithms—especially at their lower-performing end, shown by the 10th percentile—struggle in this noise-dominated regime

and achieve very high TPRs once the number of pulses in an event exceeds a standard deviation over the mean number of expected noise pulses.

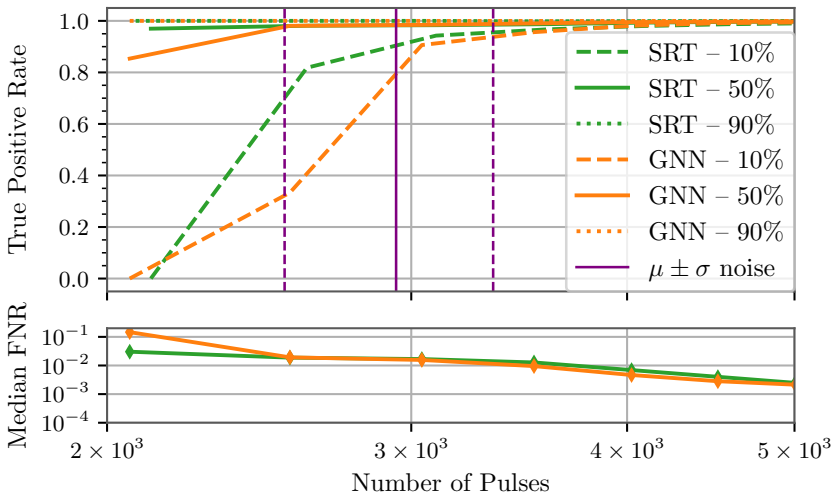

Figure 5.3.1: True positive rate (TPR) of the GNN model compared to SRT cleaning as a function of the number of pulses per event. Vertical lines indicate the mean number of noise pulses per event and the one-sigma interval. The False Negative Rates (FNRs) of the median bin centers are shown in the lower plot, highlighting the difference to a TPR of 1, which is closely approached but not reached.

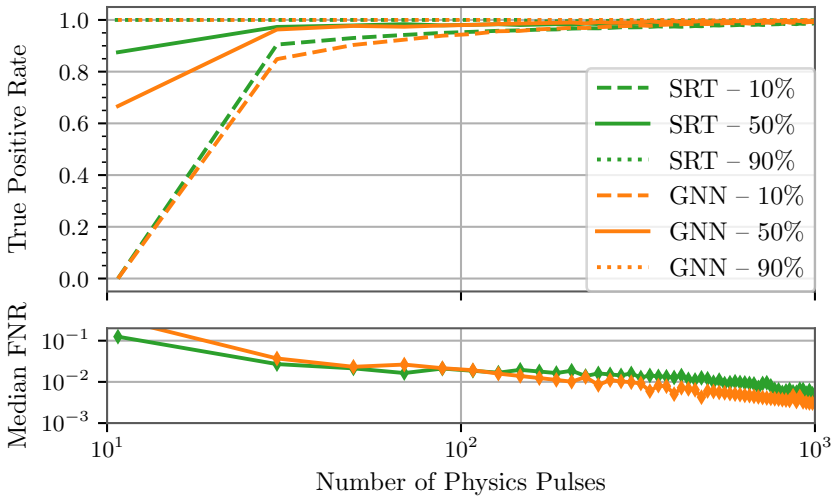

Figure 5.3.2: True positive rate (TPR) of the GNN model compared to SRT cleaning, shown as a function of the number of physics pulses per event.

The plot shows that for high pulse counts, both models converge towards a TPR of nearly 1. Moreover, the FNRs are plotted in the bottom plot of Figure 5.3.1, which clarifies that the TPR does not actually reach a value of 1. Differences become more apparent in the regime of lower pulse counts, where SRT cleaning appears to slightly outperform the GNN.

In addition, the TPR is plotted as a function of the number of physics pulses per event in Figure 5.3.2. It can be seen that the performance difference between the two algorithms is small even for low numbers of physics pulses. While there is a noticeable but small performance gap at 11 pulses,

the GNN quickly catches up, and from around 30 physics pulses onward, both algorithms perform comparably.

False Positive Rate

The False Positive Rate (FPR) reflects the model's ability to correctly identify noise hits. An ideal classifier that removes all noise would achieve an FPR of 0. Conversely, a higher FPR indicates that more noise hits remain in the dataset after cleaning, which can negatively impact subsequent steps such as event reconstruction. Therefore, comparing the FPRs of the SRT and GNN methods is essential for evaluating their effectiveness in future applications. Since noise pulses that remain after cleaning correspond to noise that was misidentified as physics, and noise pulses before cleaning represent the actual negatives, the False Positive Rate (FPR) can be calculated as the number of noise pulses after cleaning divided by the number of noise pulses before cleaning.

The results are shown in Figure 5.3.3. As before, the spread is visualized by plotting the 10th, 50th, and 90th percentiles, representing the central 80% of the data, symmetrically distributed around the median. The figure clearly highlights the superior performance of the GNN model: its ability to correctly classify noise hits significantly exceeds that of SRT cleaning. In the regime of small pulse counts up to 10,000 pulses, GNN cleaning retains a median of less than 0.5% of noise hits, while SRT cleaning retains a median of approximately 30%.
While both models show an upward trend in FPR with increasing pulse count, the GNN median remains low, between 1% and 2%. A larger view of the GNN alone is provided in Figure 5.3.4. In contrast, the median FPR of SRT cleaning rises to roughly 35% for events with 100,000 pulses.

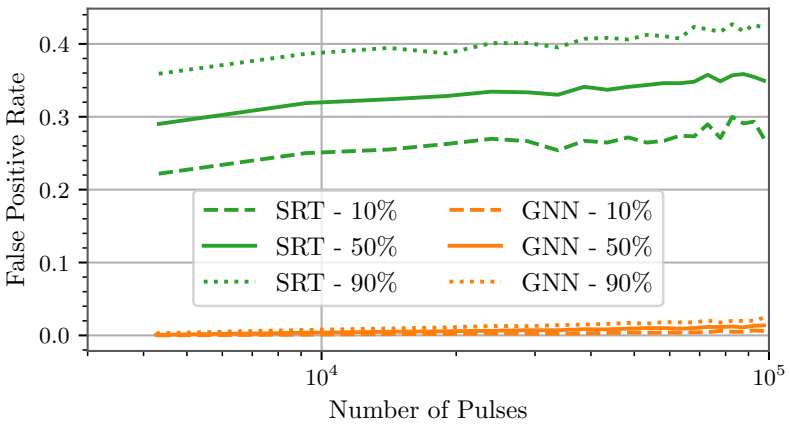

Figure 5.3.3: Comparison of the False Positive Rate of the two models as a function of the number of input pulses. The 10th, 50th, and 90th percentiles are shown, representing the central 80% of the data. The x-axis uses a logarithmic scale to enhance readability across multiple orders of magnitude.

The general upward trend observed in the false positive rate (FPR) for both GNN and SRT cleaning can be attributed to the increasing likelihood that noise pulses lie very close to physics pulses in space and time as the total number of pulses grows. In such cases, separating noise from signal becomes difficult or impossible. This effect becomes more pronounced with a higher number of physics pulses and applies to both SRT and GNN cleaning.

Furthermore, when examining the rise of the GNN curves in Figure 5.3.4, it is important to consider the chosen classification threshold. As shown in Figure 5.2.2, the threshold of 0.5 is suboptimal for events with high pulse counts; the corresponding green point on the ROC curve is shifted to the right relative to the optimal top-left corner. Consequently, for these events, the FPR is higher than it would be with a stricter threshold, such as 0.9. This trade-off was made consciously in the case of this thesis, prioritizing noise cleaning in events with lower pulse counts, where it has the greatest impact. Therefore, the rising FPR at high pulse counts is consistent with expectations. Nevertheless, for future applications, this should be kept in mind: if noise cleaning is to be applied to events with very high pulse counts, a stricter threshold may be appropriate to reduce the FPR.

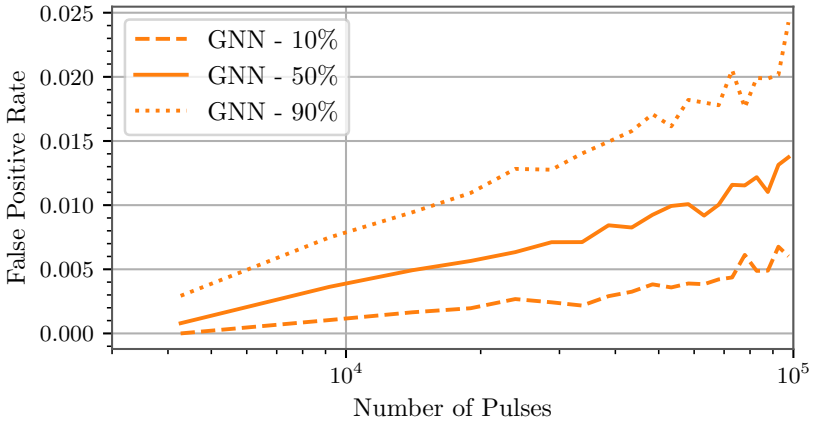

Figure 5.3.4: False Positive Rate (FPR) of the GNN only. The x-axis uses a logarithmic scale to enhance readability across multiple orders of magnitude.

5.4 Resource Usage and Optimization

To assess the benefits of using a GNN-based model compared to SRT cleaning, it is important to consider not only accuracy but also the computational resources required during inference. Even the most effective noise-cleaning model is of little use if its resource requirements cannot be met in practice. Therefore, this chapter examines the computation time and memory usage of GNN inference in comparison to SRT cleaning.

5.4.1 Memory usage (RAM)

The memory usage is also an important limiting factor in the performance of a model. Especially with regard to high-energy events which will be detected by the IceCube-Gen2 optical array, containing a large number of pulses, it is crucial to be aware of the model's behavior for large inputs. The RAM needed to perform inference is plotted as a function of the number of pulses per event in figure 5.4.1.

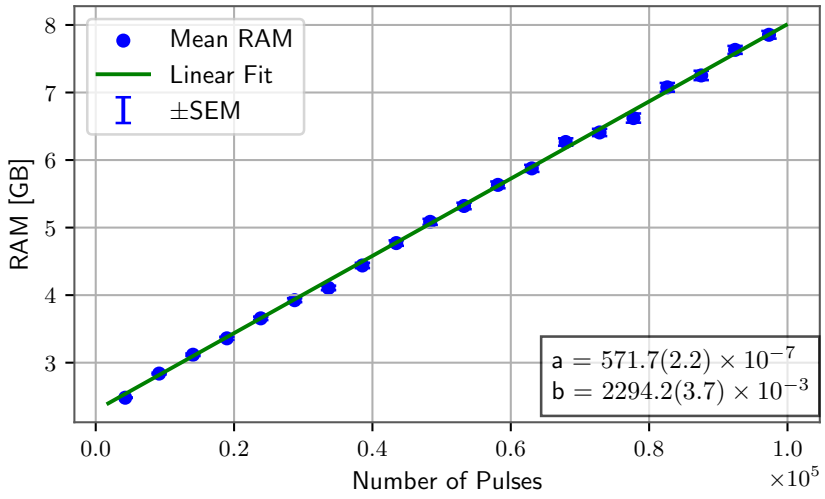

Figure 5.4.1: RAM usage required by the GNN model during inference as a function of the number of input pulses. The data is fitted with a linear function $y = a \cdot x + b$, where y is the RAM usage in gigabytes (GB) and x is the number of pulses. The fit parameter a has units of GB/pulse, and b has units of GB. The parameter values are shown in the bottom right corner of the plot. SEM refers to the standard error of the mean.

For SRT cleaning, accurately measuring the RAM usage per event is more challenging. This computation cannot be performed natively within the module-based IceTray software, and substantial modifications to the C++ code would be required—changes that are beyond the scope of this thesis. Nevertheless, an upper limit of 4 GB is estimated based on the peak memory usage observed in dataset 23170, where SRT noise cleaning was applied to PeV events containing up to several million pulses.

In comparison, GNN cleaning requires more memory, particularly for high-energy events with pulse counts exceeding 10^5. The linear fit indicates that at 10^5 pulses, the GNN uses roughly twice the memory defined as the upper limit for SRT cleaning, with usage increasing linearly beyond that point. In its current form, GNN noise cleaning therefore becomes computationally expensive at high pulse counts. However, the most critical regime for noise cleaning is at low energies with fewer pulses, where noise has a stronger impact on directional reconstruction. In this range, the GNN's memory usage remains reasonable, staying below $4\,\mathrm{GB}$ for pulse counts under 30,000.

5.4.2 Computation Time

Running inference with the GNN has been shown to require computation time that scales quadratically with the number of input pulses. This is shown for CPU in Figure 5.4.2, and for GPU in Figure 5.4.3. For tracking the inference time, in both cases the batch size was changed to 1, since a deployment without batching in IceTray is planned. A quadratic fit was performed to enable extrapolation of the computation time for events exceeding 100,000 pulses. Comparing the behavior of computation time between CPU and GPU shows that while the GPU does offer a faster inference time, the advantage is not as large as it might be expected, with CPU requiring roughly $69\,\mathrm{s}$ for 10^5 pulses, and GPU requiring $51\,\mathrm{s}$. One possible explanation for this is the batch size of 1: A single graph with 10^5 pulses may not supply enough parallel work to fully take advantage of the GPU's computational power. This is an indication that batching should be considered an option for the deployment in IceTray.

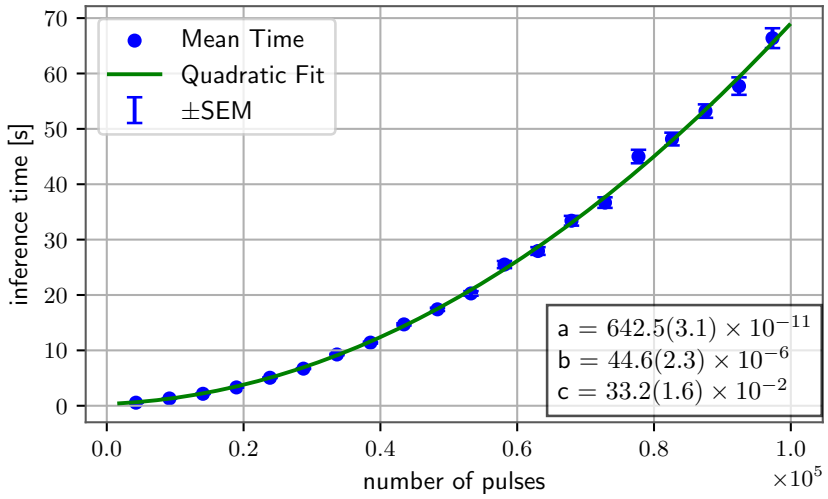

Figure 5.4.2: Quadratic fit to the inference time required by the GNN on a CPU as a function of the number of pulses per event (binned). The fit was performed using $y = a \cdot x^2 + b \cdot x + c$, where y is the inference time in s, and x is the number of pulses. To ensure consistent units, the fit parameter a has units of s/pulse^2, while b is given in s/pulse and c is given in s. The values of the fit parameters are shown in the bottom right corner. SEM is the standard error of the mean.

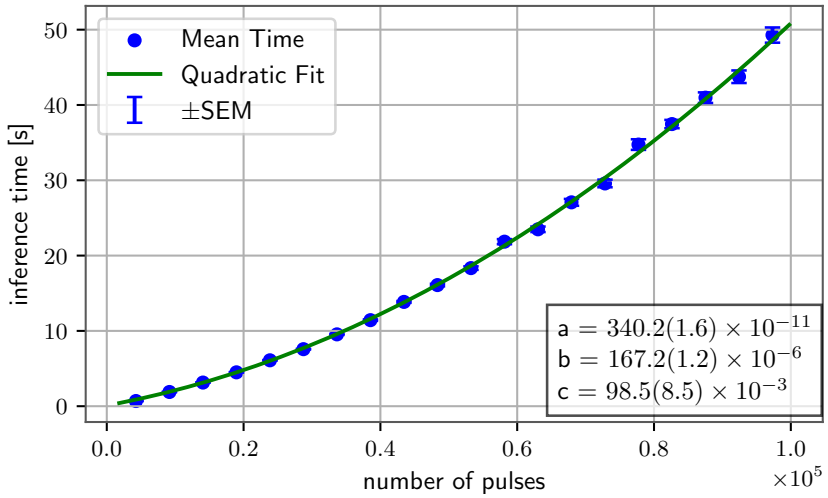

Figure 5.4.3: Inference time required by an H200 SXM GPU with 141 GB of VRAM as a function of the number of pulses per event. A quadratic fit $y = a \cdot x^2 + b \cdot x + c$ was applied to the data, where y is the inference time in s, and x is the number of pulses. The fit parameter a has units of s/pulse^2, while b is given in s/pulse and c is given in s. The values of the fit parameters are shown in the bottom right corner. SEM is the standard error of the mean.

The apparent trend of computation time scaling quadratically with the number of pulses per input event is supported by findings in other scientific publications. In [34], it is stated that for a graph with n nodes and the feature dimension is d, the computation of the matrix multiplication AX requires $O(n^2 d)$ time, where $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix, and $X \in \mathbb{R}^{n \times d}$ is the data matrix. Since each hit on a PMT is represented as a node n in this case, it is reasonable to assume a quadratic

dependence of inference time on the number of pulses.

SRT cleaning, on the other hand, shows a different behavior. As shown in Figure 5.4.4, the execution time scales linearly with the number of input pulses, and a linear fit was applied to the data. This behavior arises from the fixed algorithmic structure of SRT cleaning: for each hit, neighboring PMTs are always scanned for additional hits. Consequently, more pulses directly translate to more computations, resulting in an overall linear scaling.

Due to the different scaling of both algorithms, their time requirements are on a comparable scale for a small input size corresponding to small numbers of pulses, but already at 10^5 pulses, GNN cleaning requires more than double the time of SRT cleaning when run on a CPU, and roughly two thirds more than SRT when run on the used GPU. Since SRT computation time scales linearly with the input size, while GNN cleaning scales quadratically, it is expected that this difference will increase with even larger numbers of pulses. Extrapolation of the fit parameters shows that for 10^6 pulses, SRT requires $(343.79 \pm 219.99)\,\mathrm{s}$, whereas the GNN, when performed on a CPU, requires $(6441.02 \pm 308.53)\,\mathrm{s}$. One attempt to address this is discussed in the following section.

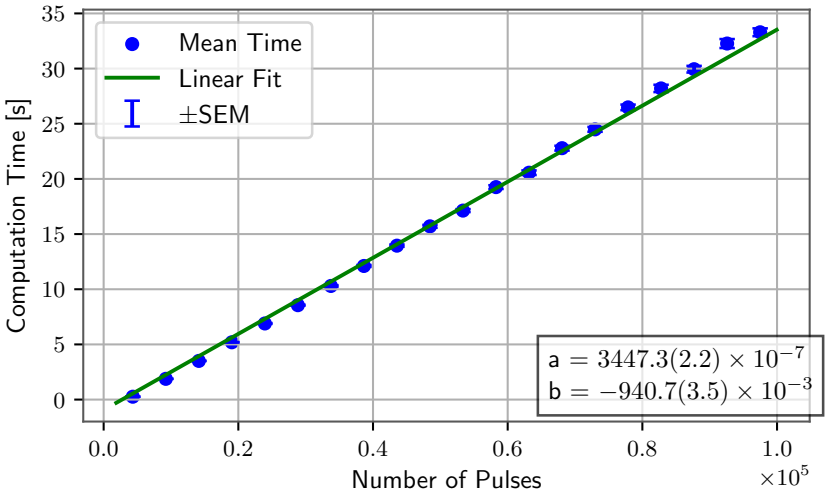

Figure 5.4.4: Computation time required to perform SRT cleaning as a function of the number of input pulses per event. A linear fit $y = a \cdot x + b$ was applied to the data, where y is the inference time in s, a has units of s/pulse, and b of s. The values of the fit parameters are shown in the lower right corner.

5.4.3 Multithreading

To reduce the inference time of the GNN model, the use of **multithreading** was investigated. Multithreading allows multiple threads of a program to be executed concurrently on different CPU cores. In general, this parallelization can significantly speed up inference when properly configured. For instance, if the code is manually set to use a multithreading factor of five threads, the job submission must explicitly request five CPU cores to make sure that the available resources are used effectively.

The number of threads was varied between 2 and 10. All computations were performed on CPUs of the PALMA cluster, equipped with AMD EPYC 7343 processors based on the Zen 3 microarchitecture. Similar as above, we fitted the inference time as a function of the number of pulses for each thread configuration. The parameter fits can be found in the Appendix in Figure A.0.1.

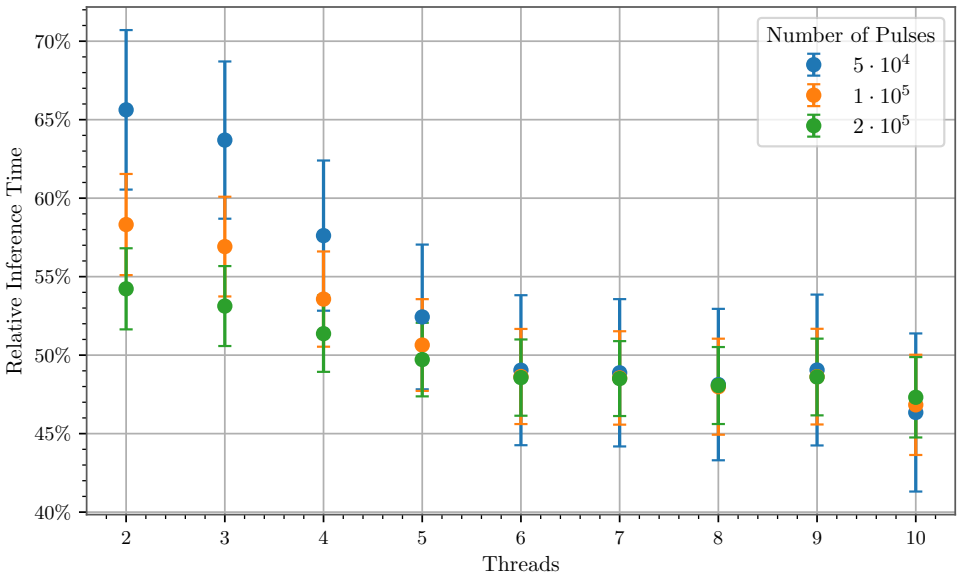

Figure 5.4.5: Relative inference time for different input pulse counts, expressed in units of the inference time without multithreading (Threads = 1), as a function of the number of threads used. The data points and uncertainties are derived from the fits shown in Figure A.0.1, and the corresponding fit parameters are listed in Table A.0.1.

For comparability, the inference time required to process $5 \cdot 10^4$, 10^5 and $2 \cdot 10^5$ pulses was calculated from each quadratic fit and plotted as a function of the number of threads in Figure 5.4.5, as a fraction of the inference time required without multithreading. While the overall trend shows decreasing inference time with an increasing number of threads, the performance gain diminishes rapidly. The plot shows that if two threads are used, this reduces the time required for inference significantly, and the advantage gained through this is relatively higher for a larger number of pulses, e.g. in this case, for $2 \cdot 10^5$ pulses, the inference time when using two threads is roughly 55% that of using only one thread, whereas for $5 \cdot 10^4$ pulses, the advantage gained is smaller, requiring roughly 65% of the inference time without threading. For all numbers of pulses, the general trend that the inference time can be reduced even further by adding more threads, but with diminishing return. From 6 threads onward, a plateau is reached, suggesting that beyond this point—where inference time across all pulse ranges is reduced to slightly less than half of that without multithreading—no significant additional speedup is achieved by adding more threads.

Using 10 threads instead of 2 reduces the inference time for 10^5 pulses by about one fourth but increases CPU usage by a factor of five. Due to this diminishing return, arbitrarily increasing the number of threads is not recommended. However, using just two threads already yields a significant reduction in computation time—for example, to $(65.6 \pm 5.1)\%$ of the time required without multithreading at 50,000 pulses. This is a substantial improvement, especially for relatively high-energy events with a large number of pulses.

With this optimization, using only two threads allows GNN noise cleaning in the examined regime up to 10^5 pulses to operate on a comparable timescale to SRT cleaning. Extrapolation of the fits suggests that even with multiple threads, GNN noise cleaning will still require more time at very high pulse counts due to its quadratic scaling with input size, in contrast to the linear scaling of SRT cleaning: For example, at 10^6 pulses, the fit parameters suggest that GNN inference with two threads will require $(3266.3 \pm 152.8)\,\mathrm{s}$, whereas SRT cleaning, at $(343.79 \pm 219.99)\,\mathrm{s}$, requires roughly a tenth of this time.

6 Summary and Outlook

The goal of this thesis was to provide a proof of concept for applying GNN-based noise cleaning to simulations for the IceCube-Gen2 optical array. The noise expected in the IceCube-Gen2 optical array will originate mainly from radioactive decays inside the glass vessels of the optical modules, currently simulated using Geant4, considering Vitrovex glass in this work. Removing this noise is crucial, as it can complicate the directional reconstruction, especially for low-energy events where an average of roughly 3000 noise pulses per event can have a significant impact.

Noise cleaning was performed using the DynEdge GNN architecture, which had previously shown strong performance in binary classification for IceCube Upgrade [3]. For comparison, the results were evaluated against the traditional SRT cleaning used in IceCube. Training data were preprocessed with IceTray and GraphNeT, equipping `i3` files with truth flags and converting them into SQLite format. These were then used to train the model to make predictions on an unseen dataset.

The performance comparison showed that both GNN and SRT cleaning achieve a comparable true positive rate (TPR) across different pulse counts per event. Both struggle more with events containing few pulses but reach very high true positive rates of roughly 0.999 at about $5{,}000$ pulses. Although SRT cleaning appears to perform slightly better than the GNN in low-pulse events, this becomes less significant when viewing the TPR as a function of physics pulses. At 11 physics pulses per event, SRT cleaning achieves a median TPR of 87.5%, compared to 66.7% for the GNN; at 30 physics pulses per event, their median TPR is almost identical, with 97.3% for SRT and 96.3% for GNN, and both continue to rise beyond this point.

The false positive rate (FPR), however, is where the GNN clearly outperforms SRT cleaning: across all pulse ranges, the fraction of noise misclassified as physics is significantly lower for the GNN. SRT cleaning maintains a median FPR of approximately 30% for events with fewer than $10{,}000$ pulses, which increases to roughly 35% at $100{,}000$ pulses. In contrast, GNN noise cleaning produces a median FPR below $0.5\,\%$ at 10,000 pulses, which also rises with the number of pulses, but reaches a peak median of 1.5% at 100,000 pulses.

Considering the small difference in retained physics hits compared to the much larger difference in false positive rate—where SRT retains about one third of noise pulses while the GNN removes nearly all—it is likely that the overall sensitivity, even in the low-pulse regime, is significantly higher for the GNN.
This improvement comes at a cost, as GNN inference requires more computational resources than the classical SRT cleaning. In the examined range (up to 10^5 pulses), inference time scales quadratically with pulse count, while RAM usage scales linearly. For very high-energy events with 10^6 pulses or more, this scaling challenges the practicality of GNN cleaning. However, for such events, noise removal is considered less critical since the average noise level of roughly $3{,}000$ pulses per event becomes less relevant compared to the total signal.

The increasing inference time was addressed with multithreading: it was shown that using just two threads already reduced runtime significantly compared to one thread, with further gains possible when more resources are available. At $50{,}000$ pulses, two threads require only $(65.6 \pm 5.1)\,\%$ of

the inference time compared to no multithreading, which drops to $(46.3 \pm 5.0)\,\%$ for ten threads. At 2,000,000 pulses, the times for two and ten threads are $(54.2 \pm 2.6)\,\%$ and $(47.3 \pm 2.6)\,\%$, respectively, of the single-thread runtime. It was also shown that running inference on a GPU offers faster computation. However the time difference could likely be improved by using batching, which is why this should be investigated further.

Given the success of the GNN in cleaning events with low pulse counts, it offers a promising way to improve IceCube-Gen2 reconstruction at lower energies, where noise cleaning is most important. Several aspects remain for future work. The model will need retraining once a more detailed PMT electronics simulation becomes available and must handle potential overlaps between noise and physics waveforms. Moreover, the current training relied on pulse-level features, which are known to yield limited data–Monte Carlo agreement in IceCube. Improving this agreement will be crucial for future deployment. The upcoming IceCube Upgrade, which is expected to use similar algorithms [3], will provide valuable insights. Alternatively, future models could explore using summary statistics of the PMT pulse lists, a method widely applied in neural-network-based algorithms in IceCube [35]. This reduces dimensionality, improves data–Monte Carlo agreement, and can help address the resource limitations observed at the highest energies.

Bibliography

[1] *IceCube — icecube.wisc.edu*. https://icecube.wisc.edu/science/icecube/. [Accessed 04-06-2025].

[2] *TDR – IceCube-Gen2 — icecube-gen2.wisc.edu*. https://icecube-gen2.wisc.edu/science/publications/tdr/. [Accessed 02-07-2025].

[3] R. Abbasi et al. "Graph Neural Networks for low-energy event classification amp; reconstruction in IceCube". In: *Journal of Instrumentation* 17.11 (Nov. 2022), P11003. DOI: 10.1088/1748-0221/17/11/P11003. URL: https://dx.doi.org/10.1088/1748-0221/17/11/P11003.

[4] Fermi National Accelerator Laboratory. *All Things Neutrino*. Accessed: 2025-06-27. URL: https://neutrinos.fnal.gov/history/.

[5] Ian Laird. *Frederick Reines won a Nobel Prize for detecting the neutrino. | LANL — lanl.gov*. https://www.lanl.gov/media/publications/national-security-science/0325-project-poltergeist. [Accessed 25-07-2025].

[6] *The Nobel Prize in Physics 2015 - Scientific Background: Neutrino Oscillations - NobelPrize.org — nobelprize.org*. https://www.nobelprize.org/prizes/physics/2015/advanced-information/. [Accessed 10-07-2025].

[7] M. Aker et al. "The design, construction, and commissioning of the KATRIN experiment". In: *Journal of Instrumentation* 16.08 (Aug. 2021), T08015. ISSN: 1748-0221. DOI: 10.1088/1748-0221/16/08/t08015. URL: http://dx.doi.org/10.1088/1748-0221/16/08/T08015.

[8] *Direct neutrino-mass measurement based on 259 days of KATRIN data — arxiv.org*. https://arxiv.org/abs/2406.13516. [Accessed 01-07-2025].

[9] *Victor Hess discovers cosmic rays | timeline.web.cern.ch — timeline.web.cern.ch*. https://timeline.web.cern.ch/victor-hess-discovers-cosmic-rays-0. [Accessed 21-07-2025].

[10] U.F. Katz and Ch. Spiering. "High-energy neutrino astrophysics: Status and perspectives". In: *Progress in Particle and Nuclear Physics* 67.3 (2012), pp. 651–704. ISSN: 0146-6410. DOI: https://doi.org/10.1016/j.ppnp.2011.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0146641011001189.

[11] Martin Antonio Unland Elorrieta. *Development, simulation, and characterisation of a novel multi-PMT optical module for IceCube Upgrade with emphasis on detailed understanding of photomultiplier performance parameters*. July 2023. DOI: 10.5281/zenodo.8121321. URL: https://doi.org/10.5281/zenodo.8121321.

[12] R. E. JENNINGS. "ČERENKOV RADIATION". In: *Science Progress (1933-)* 50.199 (1962), pp. 364–375. ISSN: 00368504, 20477163. URL: http://www.jstor.org/stable/43425324 (visited on 06/16/2025).

[13] Aya Ishihara. *The IceCube Upgrade – Design and Science Goals*. 2019. arXiv: 1908.09441 [astro-ph.HE]. URL: https://arxiv.org/abs/1908.09441.

[14] M G Aartsen et al. “IceCube-Gen2: the window to the extreme Universe”. In: *Journal of Physics G: Nuclear and Particle Physics* 48.6 (Apr. 2021), p. 060501. ISSN: 1361-6471. DOI: 10.1088/1361-6471/abbd48. URL: http://dx.doi.org/10.1088/1361-6471/abbd48.

[15] Alexander Kappes. *The Optical Sensor for IceCube-Gen2*. 2025. arXiv: 2507.08415. URL: https://arxiv.org/abs/2507.08415.

[16] IceCube Collaboration. *OMSim: Optical Module Simulator*. https://github.com/icecube/OMSim. Accessed: 2025-07-29. 2025.

[17] IceCube Collaboration. *SLC Hit Cleaning*. Access restricted; login required. [Accessed 17-07-2025]. URL: https://wiki.icecube.wisc.edu/index.php/SLC_hit_cleaning.

[18] *What is a Perceptron: Components, Characteristics, and Types — simplilearn.com*. https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron. [Accessed 01-07-2025].

[19] *ReLU &x2014; PyTorch 2.7 documentation — docs.pytorch.org*. https://docs.pytorch.org/docs/stable/generated/torch.nn.ReLU.html. [Accessed 20-07-2025].

[20] *Sigmoid &x2014; PyTorch 2.7 documentation — docs.pytorch.org*. https://docs.pytorch.org/docs/stable/generated/torch.nn.Sigmoid.html. [Accessed 26-07-2025].

[21] *Deep Neural Networks: Concepts and History Overview — botpenguin.com*. https://botpenguin.com/glossary/deep-neural-network. [Accessed 15-07-2025].

[22] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. Accessed: 2025-06-25. 2021. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a/.

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

[24] *Intro to optimization in deep learning: Momentum, RMSProp and Adam | DigitalOcean — digitalocean.com*. https://www.digitalocean.com/community/tutorials/intro-to-optimization-momentum-rmsprop-adam. [Accessed 15-07-2025].

[25] Farhana Sultana, Abu Sufian, and Paramartha Dutta. “Advancements in Image Classification using Convolutional Neural Network”. In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*. IEEE, Nov. 2018, pp. 122–129. DOI: 10.1109/icrcicn.2018.8718718. URL: http://dx.doi.org/10.1109/ICRCICN.2018.8718718.

[26] Rasmus F. Ørsøe, Aske Rosted, and GraphNeT Team. *GraphNeT 2.0 – A Deep Learning Library for Neutrino Telescopes*. 2025. arXiv: 2501.03817 [hep-ex]. URL: https://arxiv.org/abs/2501.03817.

[27] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019, pp. 8024–8035.

[28] William Falcon and the PyTorch Lightning team. *PyTorch Lightning*. Version 2.0.2. 2023. DOI: 10.5281/zenodo.3530844.

[29] IceCube Collaboration. *IceTray Public*. GitHub repository, accessed July 27, 2025. 2025. URL: https://github.com/icecube/icetray-public.

[30] *Home | Vitrovex — vitrovex.com*. https://vitrovex.com/. [Accessed 25-07-2025].

[31] *htcondor.org*. https://htcondor.org/. [Accessed 27-07-2025].

[32] Competence for Computing in Science Universität Münster. *PALMA — uni-muenster.de*. https://www.uni-muenster.de/CoCoS/Systeme/PALMA.html. [Accessed 15-07-2025].

[33] *GPUs - HPC Documentation - University of Münster — palma.uni-muenster.de*. https://palma.uni-muenster.de/documentation/hardware/gpus/. [Accessed 03-08-2025].

[34] Seiyun Shin, Ilan Shomorony, and Han Zhao. "Efficient Learning of Linear Graph Neural Networks via Node Subsampling". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 55479–55501. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/ada418ae9b6677dcda32d9dca0f7441f-Paper-Conference.pdf.

[35] The IceCube collaboration et al. "A convolutional neural network based cascade reconstruction for the IceCube Neutrino Observatory". In: *Journal of Instrumentation* 16.07 (July 2021), P07041. DOI: 10.1088/1748-0221/16/07/P07041. URL: https://dx.doi.org/10.1088/1748-0221/16/07/P07041.

A Additional Plots to Chapter 5

Table A.0.1: Fit parameters of the fits presented in Figure A.0.1. A quadratic fit $y = a \cdot x^2 + b \cdot x + c$ was applied to the data, where y is the inference time in s, and x is the number of pulses. The fit parameter a has units of s/pulse^2, while b is given in s/pulse and c is given in s.

Threads	$a\,[10^{-10}\,\mathrm{s/pulse^2}]$	$b\,[10^{-5}\,\mathrm{s/pulse}]$	$c\,[10^{-1}\,\mathrm{s}]$
1	63.9(3.1)	5.1(3.2)	1.4(7.1)
2	31.8(1.5)	8.4(1.6)	0.9(3.5)
3	31.3(1.5)	7.9(1.6)	1.0(3.5)
4	31.3(1.4)	5.6(1.5)	1.3(3.3)
5	31.2(1.4)	3.7(1.4)	1.4(3.2)
6	31.1(1.4)	2.4(1.5)	2.0(3.3)
7	31.0(1.4)	2.3(1.5)	1.9(3.2)
8	30.8(1.4)	2.2(1.5)	2.0(3.3)
9	31.1(1.4)	2.2(1.5)	2.5(3.3)
10	30.7(1.5)	1.5(1.6)	2.5(3.5)

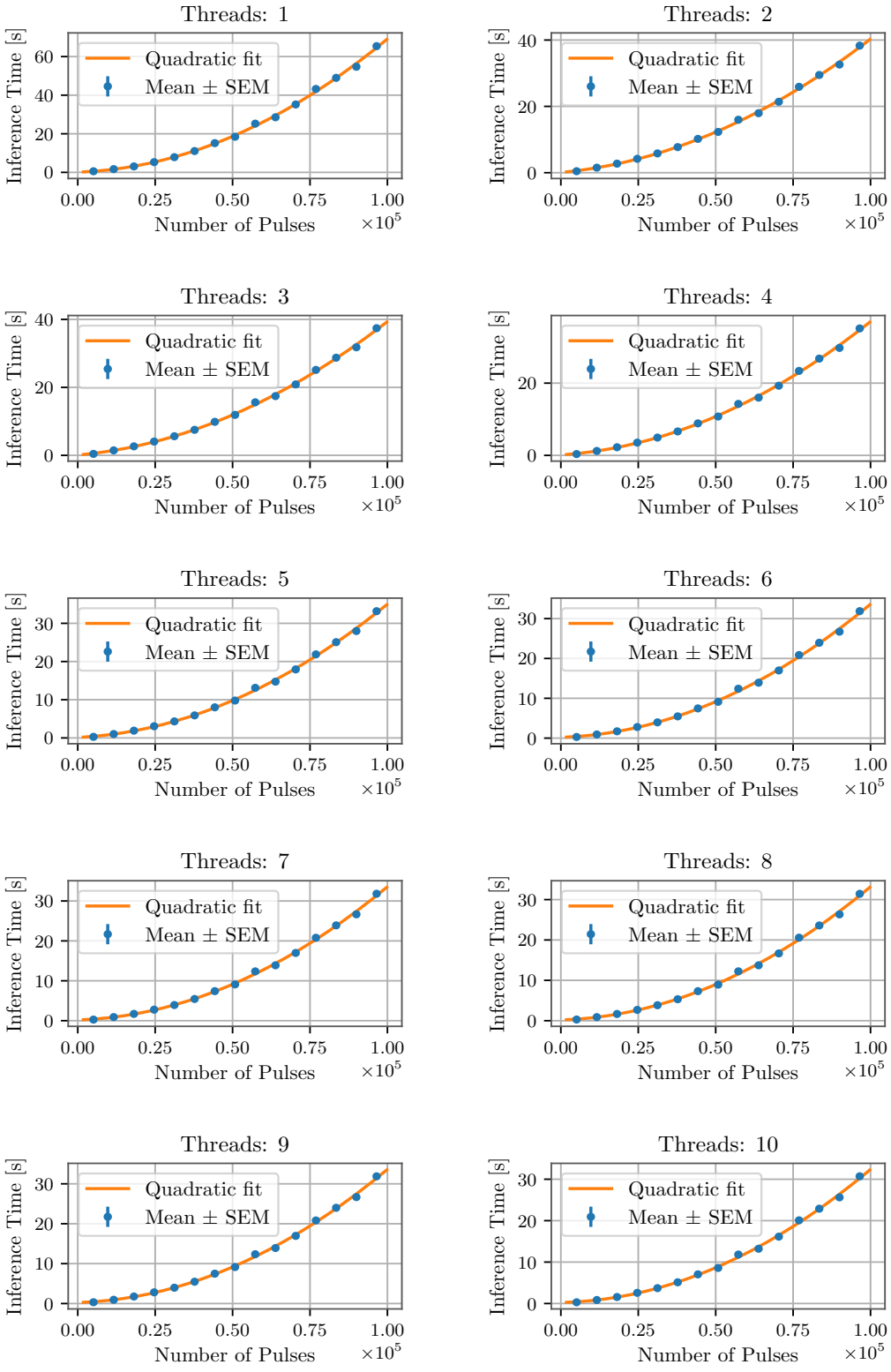

Figure A.0.1: Inference time for different thread counts as a function of input size, arranged from 1 thread (top left) to 10 threads (bottom right). The plots should be read row-wise, from left to right and top to bottom. A quadratic fit $y = a \cdot x^2 + b \cdot x + c$ was applied to the data, where y is the inference time in s, and x is the number of pulses. The fit parameter a has units of s/pulse^2, while b is given in s/pulse and c is given in s. The values of the fit parameters are shown in the bottom right corner. SEM is the standard error of the mean.

Acknowledgement

First of all, I would like to express my gratitude to Prof. Kappes for giving me the opportunity to write my thesis within this amazing group.

I am also thankful to Prof. Andronic for agreeing to serve as my second examiner.

My deepest thanks go to Javier Vara Carbonell, whose patience and guidance exceeded anything I could have hoped for in a supervisor. In addition, his encouragement to apply for the Double Master in Sevilla has had a big impact on me.

I am grateful to the entire working group for the support and the fun; in particular, I want to thank Fia Tenbruck, Berit Schlüter and Dr. Waleed Esmail for their invaluable help with proofreading.

Heartfelt thanks also go to all my friends and flatmates for their unwavering emotional support during the past months.

Last but certainly not least, I owe endless gratitude to my parents, Maria and Dirk, as well as to my siblings for their constant encouragement and unfailing support.