Data Science & Neutrino Physics: Improving the Pandora Reconstruction Framework at the DUNE Far Detector

Ryan Joseph Cross

This thesis is submitted for the degree of
Doctor of Philosophy

Physics Department
Lancaster University

June 5, 2023

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 62,713 words.

Ryan Joseph Cross

Data Science & Neutrino Physics: Improving the Pandora Reconstruction Framework at the DUNE Far Detector

Ryan Joseph Cross
Physics Department, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*, June 5, 2023.

Abstract

This thesis outlines two new methods to drastically improve the reconstruction capabilities of neutrino events at Deep Underground Neutrino Experiment (DUNE), in the Pandora reconstruction framework. The liquid argon time projection chamber (LArTPC), the detector technology of choice at DUNE, provides high spatial and calorimetric resolutions, presenting a difficult but exciting reconstruction problem. One of the main reconstruction frameworks for event reconstruction in LArTPCs is Pandora, a software development kit using a multi-algorithm approach to pattern recognition, which is designed to target the complex pattern recognition problems that occur in particle physics. The work in this thesis includes an overhaul of the 3D event reconstruction for tracks, producing 3D hits from combinations of underlying 2D positions. This new method produces more coherent and truthful 3D representations of tracks, by intelligently selecting hits from a generated 3D point cloud through stages of fitting. Secondly, a graph neural network (GNN) is utilised for the complex problem of electromagnetic shower growing, taking an electron or photon shower that is clustered into hundreds of small groups and producing larger, more representative clusters per shower, whilst avoiding contamination from other interactions in the event. Deep learning is used to give a more global view of the event for growing, and to better use the topological features of the showers to help the growing process. All this work is verified on DUNE far detector simulated data, to give an understanding of what performance gains are made, and the failure modes they fix. Verification of the deep learning method is performed on real test beam data from ProtoDUNE Single-Phase (ProtoDUNE-SP) at CERN. This verification helps give confidence that work performed on simulated data can also be applied to real data, which is especially interesting for methods that utilise deep learning.

Acknowledgements

Whilst this work is my own, it would not have been possible without all the support I received, both in Lancaster and beyond.

First, I am immensely grateful for all the support I received from my supervisor, Dr Andrew Blake throughout my four years at Lancaster, getting me up to speed with Pandora, providing countless insights into the physics and for every comment provided on my thesis as I wrote it.

I'd like to thank all the members of Lancaster EPP, for their support and laughs that helped me get through my PhD, especially Katherine Rybacki, Neza Ribaric and Tristan Doyle. The rest of the Lancaster Liquid Argon team has provided plenty of help and questions over the years, which has helped my work get better and better. I'd like to especially thank Dominic Brailsford, as he has helped me an absurd amount over my years at Lancaster, and his guidance throughout my work has been invaluable.

I need to blame Alex Goldsack for getting me to swap from Computer Science into Physics at Lancaster University. He has helped me with Physics throughout, which I am thankful though, even if he probably still owes me for the technical help I've given him over the years.

My greatest thanks go to both my friends and family for all their support over the years. I can't thank my parents enough for their continued support and love over the years, and helping to push me forward. And to Cat, your support has meant the world to me over the years.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog-to-digital converter

ANN Artificial neural network

APA Anode plane assembly

ArgoNeuT Argon Neutrino Teststand

ATLAS A toroidal LHC apparatus

BDT Boosted decision tree

BSM Beyond Standard Model

CC Charged-current

CDF Collider detector at Fermilab

CERN European Organisation for Nuclear Research

CKM Cabibbo-Kobayashi-Maskawa

CLIC Compact Linear Collider

CMS Compact muon solenoid

CNN Convolutional neural network

CORSIKA COsmic Ray SImulations for KASCADE

CP Charge-parity

CPA Cathode plane assembly

CPV CP violation

CRP Charge readout plane

CVN Convolutional visual network

DESY Deutsches Elektronen-Synchrotron

DIS Deep inelastic scattering

DL Deep learning

DOE Department of Energy

DONUT Direct observation of the nu tau

DP Dual-phase

DUNE Deep Underground Neutrino Experiment

DUNE-PRISM DUNE Precision Reaction-Independent Spectrum Measurement

ECAL Electromagnetic calorimeter

EHN1 Experimental Hall North 1

ES Elastic scattering

ESPP European Strategy for Particle Physics

FC Field cage

FD Far detector

FHC Forward horn current

Fermilab Fermi National Accelerator Laboratory

GAN Generative adversarial network

GENIE Generates Events for Neutrino Interaction Experiments

GNN Graph neural network

GPU Graphics processing unit

GPUaaS GPU-as-a-Service

HD Horizontal drift

HPgTPC High-pressure gaseous TPC

ICARUS Imaging Cosmic And Rare Underground Signals

ILC International Linear Collider

IO Inverted ordering

KamLAND Kamioka Liquid Scintillator Antineutrino Detector

KASCADE KArlsruhe Shower Core and Array DEtector

KNN K-Nearest neighbour

LAr Liquid argon

LArSoft Liquid Argon Software

LArTPC Liquid argon time projection chamber

LBL Long-baseline

LBNF Long Baseline Neutrino Facility

LEP Large Electron–Positron Collider

LHC Large Hadron Collider

MAGAN Model-assisted GAN

MC Monte Carlo

MINERvA Main Injector Neutrino ExpeRiment to study v-A interactions

MINOS Main injector neutrino oscillation search

MIP Minimally ionising particles

ML Machine learning

MLP Multi-layer perceptron

MVA Multivariate analysis

NC Neutral-current

ND Near detector

ND-GAr Gaseous Argon Near Detector

ND-LAr Liquid Argon Near Detector

NN Neural network

NO Normal ordering

NOvA NuMI Off-axis ν_e Appearance

NSI Nonstandard interactions

OSG Open Science Grid

P5 U.S. Particle Physics Project Prioritization Panel

PCA Principle component analysis

PCB Printed circuit board

PD Photon detector

ProtoDUNE-DP ProtoDUNE Dual-Phase

PDS Photon detection system

ProtoDUNE-SP ProtoDUNE Single-Phase

PID Particle identification

PIP-II Proton Improvement Plan II

PMNS Pontecorvo-Maki-Nakagawa-Sakata

QE Quasi-elastic

RANSAC Random sample consensus

ReLU Rectified linear unit

RENO Reactor Experiment for Neutrino Oscillation

RES Resonance

RHC Reverse horn current

RNN Recurrent neural network

S/N Signal-to-noise

SAND System for on-Axis Neutrino Detection

SBN Short-Baseline Neutrino

SBND Short-Baseline Near Detector

SCE Space charge effect

SiPM Silicon photomultiplier

SLAC SLAC National Accelerator Laboratory

SM Standard Model

SNO Sudbury Neutrino Observatory

SURF Sanford Underground Research Facility

T2K Tokai to Kamioka

TDC Time-to-digital converter

TPC Time projection chamber

VD Vertical drift

1

Introduction

"One thing I've learned. You can know anything. It's all there. You just have to find it."

John Constantine - The Sandman

Neutrino physics has enjoyed a period of extensive development and investment in recent history, spurred on by the discovery of neutrino flavour oscillations. Because of this, the field as a whole has moved on from first-time measurements to an era of increased precision. Every parameter that controls neutrino oscillations is either well constrained, or work is in-progress to improve the current measurement of that parameter. Chasing higher precision measurements means there is an even greater need for higher and higher resolutions of detector, larger and larger masses for even more interactions, and maximising the efficiency of every link in the analysis chain.

This requirement means that computing is needed now more than ever, to deal with ever-increasing data sets and the increasing complexity of interactions in each event. Intelligently implemented software is a key requirement of any neutrino experiment, a core component to unlock the full potential of the detector hardware. Without it, the carefully designed hardware and all the work that went into it may be wasted, blurred away behind ineffective code that obscures the true power of the detector. Effective reconstruction software is required for multiple stages of the physics pipeline, producing performant selection algorithms to pick the target neutrino events, and also to reconstruct the target particle energy. Both of these are a key component to extract the physics parameters like δ_{CP} that we are chasing.

Deep Underground Neutrino Experiment (DUNE), as a next generation neutrino experiment, is at the forefront of neutrino experiments and also has to contend with these issues. The detector hardware chosen, the liquid argon time projection chamber (LArTPC), offers high spatial and calorimetric resolution, resulting in a difficult but very exciting reconstruction problem, to help untangle the detailed neutrino interaction hierarchies. The high resolution, combined with the rich physics program at DUNE means there is a significant number of different topologies and interaction types that must be understood and reconstructed well, to unlock new physics and aide physics analyses.

This thesis presents work to improve the reconstruction of events in the DUNE horizontal drift far detector. Computing work such as this is an important part of any experiment, and reconstruction must be in place when the DUNE detector starts taking initial data, to aid understanding of how the detector is performing, and remove any potential delay between data taking and using the collected data. As computing has evolved, so have the techniques used, taking advantage of increased available computing power, as well as a deeper understanding of efficient software production. For this reason, a modern neutrino experiment must also utilise modern data science techniques including deep learning and the wealth of advancements made there to help improve the reconstruction of events, enabling physics analyses to proceed with fewer issues.

First, Chapter 2 outlines the history of neutrino physics, alongside a brief overview of how computing and neutrino physics has evolved together over time. As software forms a core part of experiments now, it is interesting to look back on how it has evolved, and how paradigms in computing have changed, especially with respect to reconstruction and analysis. Due to neutrino physics having a rich history of experiments over the years, we can track the changes and broad uses of techniques from the sixties through to today. This chapter also outlines some common neutrino interaction types, to aid understanding of later chapters.

With an understanding of both neutrino physics and the ties to computing, Chapter 3 gives an overview of DUNE, including its physics goals, and an overview of the projects and hardware at the heart of the experiment, including the chosen detector technology, the LArTPC. Additionally, an outline of ProtoDUNE-SP, the prototype of one of the DUNE detector designs at CERN is given. ProtoDUNE-SP is exciting for many reasons, providing a realistic test bed for both hardware designs and also providing a realistic data input for software production and

validation.

As software is a key part of this thesis, Chapter 4 outlines the software stack in use at DUNE, explaining the high-level steps to go from nothing through to a full realistic simulated neutrino interaction in the chosen DUNE detector designs. Following from the more general overview of the software steps, an in-depth explanation is given of Pandora, the reconstruction framework within which the work from this thesis was implemented. This allows the design behind Pandora to be explained, as well as a more thorough explanation of the steps that Pandora takes to reconstruct interactions in LArTPCs.

Deep learning is becoming a larger and larger part of analysis across all of computing, not just particle physics. Chapter 5 covers the basics of deep learning, from basic neural networks and how they learn, through to image and graph based networks. Then, a brief overview of how deep learning is being used in particle physics experiments, to give an idea of the wide range of use cases it has, across many problems.

Chapter 6 is dedicated to work on improving the 3D reconstruction of hits in Pandora. First, an explanation of the basics of producing a 3D hit is given, as well as some approaches to match hits across views. This is followed by the limitations and why work is needed in this area. Next, an explanation of each of the core components that went into upgrading the 3D hit creation in Pandora, as well as an overview of the performance improvements seen, split into how each of the individual changes helps improve the production of sensible 3D hits. As part of this thesis, extended investigation was made into the ordering and impact of the existing algorithms, as well as development of the new tooling and algorithms to exploit the existing algorithms more effectively. This includes adding the ability to run all algorithms, not just one, and then all tooling to utilise this new, extend output. Additionally, existing infrastructure was extended to add new features like stronger detector geometry checks and hit interpolation. The existing initial 3D hit creation algorithms were not developed as part of this thesis.

Chapters 7 and 8 are dedicated to the improvement of shower growing in Pandora with deep learning. Shower growing is the process of building up electromagnetic showers from many hundreds of small groups of hits, into large groups that more accurately reflect an individual electromagnetic shower. Chapter 7 outlines the existing implementation for this growing step, as well as the work put in to understand its limitations and potential scope for improvement.

This is followed by extensive work on the building of a graph neural network to target this problem, including the input graph and considerations around building it, and the technical implementation of that graph and associated network in Pandora. This is followed by Chapter 8 which is an in-depth look at all the results of the new deep learning-powered growing, starting with the training process, and then details about the performance on both the DUNE far detector simulation and real ProtoDUNE-SP data. Metrics are given for performance, examples of the growing itself, and potential further work based on the final metrics to unlock even more performance, given in context against the existing shower growing. With the exception of the outlined 'existing' growing, all the work outlined in this chapter was performed as part of this thesis, including the cheating study and development of a cheating algorithm, and then all subsequent studies into graph structure and shower growing parameters.

Finally, Chapter 9 contains a summary of all the achieved results.

As part of this thesis, large amounts of technical work was needed to interface between software tooling, both in creating new interfaces and extending existing ones. As most of this work has no physics impact, and instead is only used to unlock the use of new ideas, it is not expanded on much when explaining the development of this thesis. However, to give an idea of the extent of this technical work, it is outlined briefly here.

For the 3D reconstruction work outlined in Chapter 6, work was needed to extend the LArSoft to Pandora interface to include additional 3D simulation information, which was required for the comparisons performed in that chapter. This took around a week to implement. A much more substantial amount of work was needed to extend the existing interface inside Pandora to include the required libraries needed for graph neural networks (GNNs) for the work in Chapters 7 and 8. The complexity here is mostly due to the relative obscurity in using deep learning (DL) tools outside of the Python programming language. To build and subsequently include the newly required libraries took around two weeks, with an additional week needed to decode and understand the undocumented software interface required to use the new libraries[1]. Another week or so was spent ensuring compliance with the generated model to ensure that it was actually able to be exported for use outside of Python, requiring slight adjustments to the model structure and interface to ensure it could be converted to a more portable format. Finally, a large amount of unseen technical work was needed to ensure the efficient

training of the developed GNN. This includes the selection of a suitable training framework, Ray Tune, as outlined in Chapter 7, as well as extensive work on the efficient creation and storing of graphs for later use in training and testing. This work took around 6 weeks total, spread across the full development of the DL shower growing.

[1]When used via Python, data structures are set up automatically for later use in training or inference. The layout of these data structures was instead reverse engineered to fit the expected layout.

2

Neutrino Physics & Computing

"And therefore, by process of elimination, the electron must taste like grapeade."

Futurama - S01E11 "Mars University"

Neutrinos are the most abundant massive known particle in the universe, however their existence was difficult to discover, due in part to how elusive they are. In fact, the neutrino was introduced as a "desperate remedy", an improbable solution to explain the results seen in the experiments of the time.

Since then, there has been an explosion of rich physics around neutrinos, with a suite of experiments designed to probe neutrinos and extend our understanding of them. Experiments to understand neutrinos have become more and more complex over time, requiring ever-increasing sensitivity, as well as statistics. Alongside the growing understanding of neutrinos, physicists have been able to take advantage of advances in computing power and techniques to be able to use the produced data to its full potential. Without improvements in computing techniques, improvements in the experiments we use to probe particle physics theories would be in vain, as the point where analyses can be performed by hand has long since passed.

This chapter will outline both the history of the neutrino, and the computing advances that were made during this period that helped utilise the experimental data to produce a physics understanding. Finally, this chapter ends with a brief explanation of the common neutrino interactions.

2.1 History of the Neutrino

In 1914, James Chadwick had shown that the electrons emitted during β decay had a continuous spectrum of energies [1], rather than the monochromatic energy distributions seen in the other forms of radiation, α and γ. This directly conflicted with the nuclear theories of the time, which were based on a two-body model of beta decay, requiring monochromatic beta spectra. The observed spectra appeared to violate the law of energy conservation.

It was Wolfgang Pauli who realised that if there was a third invisible particle that was part of the interaction [2], this particle could take with it a portion of the interaction's energy, which would result in the observed distributions in electron energies. Pauli named this new neutral particle the neutron, with Fermi later renaming it to the neutrino following Chadwick's discovery of the neutron in 1932, with the name neutrino chosen to reference the electrically neutral nature of the particle, as well as the very small rest mass the particle must have[1].

In 1934, Fermi formalised a neutrino-inclusive beta decay theory, which described the reactions:

$$(A, Z) \rightarrow (A, Z + 1) + e^- + \bar{\nu}_e \tag{2.1.1}$$

$$(A, Z) \rightarrow (A, Z + 1) + e^+ + \nu_e \tag{2.1.2}$$

where A and Z are the mass number and atomic number, e^- and e^+ are electrons and positrons, and ν_e and $\bar{\nu}_e$ are the neutrino and antineutrino.

Despite having a formalised theory, it would not be until the 1950s that evidence for the existence of neutrinos was confirmed experimentally by Reines and Cowan.

Reines and Cowan were able to find experimental evidence for the "free neutrino" [3, 4], using an experiment that was initially designed to use the process of inverse beta decay

$$\bar{\nu}_e + p \rightarrow n + e^+ \tag{2.1.3}$$

to detect antineutrinos from a nuclear explosion that would take place nearby.

[1] *-ino* is a diminutive suffix in Italian, to convey the ideal of the small rest mass of the particle, i.e. little neutral one.

This was later changed to use antineutrinos that were emitted from the radioactive decays of fission products occurring inside a nuclear reactor instead.

Later, it would be determined that the Reines and Cowan had discovered the electron neutrino, ν_e, (specifically the electron antineutrino, $\bar{\nu}_e$) and it would not be until 1962 that the muon neutrino, ν_μ would be discovered by Leon Lederman, Melvin Schwartz and Jack Steinberger [5]. At Brookhaven National Laboratory, they created a beam of muon neutrinos from decaying pions, and could then observe the leptons that were produced in the neutrino interactions. They found that in the interactions they observed, only muons were produced, and therefore the neutrinos they had observed were associated with a muon. Additionally, this also showed that neutrinos are produced with a specific flavour in weak interactions.

Given that both the muon neutrino and the electron neutrino were associated with charged leptons (the e and the μ), following the discovery of the tau lepton by Martin Perl in 1975 [6], it was hypothesised that a third flavour of neutrino, the tau neutrino, must exist. It would not be until 2000 that the DONUT collaboration would finally find evidence of the tau neutrino, ν_τ [7]. The DONUT experiment produced tau neutrinos from the decay of charmed mesons produced in collisions between protons and a stationary target, where they would be detected with a fine-grain emulsion detector.

The ν_τ was the last lepton of the Standard Model to be found, as well as being the second most recent particle to be found, with the Higgs Boson being experimentally discovered in 2012 [8]. Limits on the number of active light neutrinos were set in 1992 by the Large Electron–Positron Collider (LEP) [9], which restricted the number to three, based on data from measurements of the width of the Z boson line shape. An active light neutrino is any neutrino where $m_\nu < \frac{m_Z}{2}$, such that it can interact with the Z boson, so the decay $Z \to \nu\bar{\nu}$ is allowed.

Even as neutrinos went from a theoretical particle to ones with direct experimental evidence, there was mounting evidence that they behaved in odd ways, creating an unexplained difference between the theories of the time, and experimental results. One suggestion for this behaviour was suggested in 1957 by Bruno Pontecorvo [10, 11], who predicted that neutrinos could oscillate, inspired by a similar process in the neutral kaon system. This would not be confirmed for many more years, but did provide an explanation for experimental issues seen later,

most notably, the solar neutrino problem and atmospheric neutrino anomaly. It would take until 2015 for neutrino oscillations to be confirmed by SNO and Super-Kamiokande, unravelling the solar and atmospheric contradictions and providing a deeper insight into the behaviour of neutrinos.

The solar neutrino problem was first identified by the Homestake experiment [12], which was an experiment designed to collect and count neutrinos emitted by nuclear fusion in the Sun. The Homestake experiment utilised a tank of perchloroethylene, a dry cleaning fluid, deep underground in the Homestake Gold Mine. Neutrinos would interact with the chlorine ^{37}Cl atoms and turn into a radioactive isotope of ^{37}Ar, which could be extracted and counted.

$$\nu_e + {}^{37}\mathrm{Cl} \rightarrow {}^{37}\mathrm{Ar} + e^- \tag{2.1.4}$$

After John Bahcall calculated a predicted rate [13] at which the detector should capture neutrinos, the experiment only turned up a value one third this figure, at 2.56 ± 0.25 SNU[2]. At the time, this was mostly discounted as an issue with the experiment, either the low rate of interactions, or missing directional and energy information, meaning it was difficult to confirm that the interactions were explicitly due to solar neutrinos, and not another source.

A similar anomaly was identified in atmospheric neutrinos. The atmosphere is constantly being hit by cosmic rays, mostly composed of protons (95%), alpha particles (5%) and heavier nuclei and electrons ($< 1\%$). When these rays hit nuclei in the atmosphere, they shower, setting up a cascade of hadrons. These hadrons in turn decay and produce atmospheric neutrinos. The dominant part of this chain is due to charged pions decaying:

$$\begin{aligned} \pi^+ &\rightarrow \mu^+\nu_\mu \;\; \mu^+ \rightarrow e^+\nu_e\bar{\nu}_\mu \\ \pi^- &\rightarrow \mu^-\bar{\nu}_\mu \;\; \mu^- \rightarrow e^-\bar{\nu}_e\nu_\mu \end{aligned} \tag{2.1.5}$$

There is also a contribution from kaon decays at higher energies. This distribution should peak around 1 GeV and extend out to hundreds of GeV, with a general ratio of

[2] 1 SNU = 10^{-36} neutrino interactions per target atom per second.

$$R = \frac{(\nu_\mu + \bar{\nu}_\mu)}{(\nu_e + \bar{\nu}_e)} \tag{2.1.6}$$

should be equal to 2, with computer models indicating this should be equal to 2 with a 5% uncertainty. Experiments, most notably Kamiokande, measured this ratio, giving their results most commonly in a double ratio of experimental data vs theoretical methods.

$$R = \frac{(N_\mu/N_e)_{\text{DATA}}}{(N_\mu/N_e)_{\text{SIM}}} \tag{2.1.7}$$

with N_μ the number of muon neutrinos in the detector and N_e the number of electron neutrino events in the detector. If the observed flavour composition matches expectation, $R = 1$. However, all measured values of R were significantly less than 1, indicating either less ν_μ, more ν_e or both.

These two problems combined lead to further investigations, culminating in SNO and Super-Kamiokande receiving Nobel Prizes for the discovery of neutrino oscillations, explaining the observed problems.

SNO was able to explain the apparently missing neutrinos that Homestake had seen, as it can detect all neutrinos, regardless of their flavour. Solar neutrinos have an energy of around 30 MeV, which combined with Homestake only being able to measure neutrino captures, outlined in Equation 2.1.4, meant Homestake was only sensitive to ν_e and effectively blind to any ν_μ or ν_τ that arose due to ν_e oscillations. SNO, a heavy water-based neutrino detector, detects neutrinos with three processes, with the most important for the solar neutrino problem being that SNO can see the neutral-current (NC) channel

$$\nu + d \rightarrow n + p + \nu \tag{2.1.8}$$

which can be used to measure the total neutrino flux, $\phi(\nu_e) + \phi(\nu_\mu) + \phi(\nu_\tau)$, assuming that the final state neutron can be currently measured. The final result for this channel achieved a flux of $\phi_{\text{NC}} = 5.09 \pm 0.63$, which agrees with the Solar Standard Model estimate. The full flux results, across the three processes, NC, CC and ES, can be seen in Figure 2.1.

Super-Kamiokande, the next generation of the Kamiokande experiment, aimed to understand the atmospheric neutrino anomaly, by using a large water Cherenkov detector, capable of resolving the angular distribution of the atmospheric neutrino interactions. Like the Kamiokande experiment, it uses the Cherenkov radiation

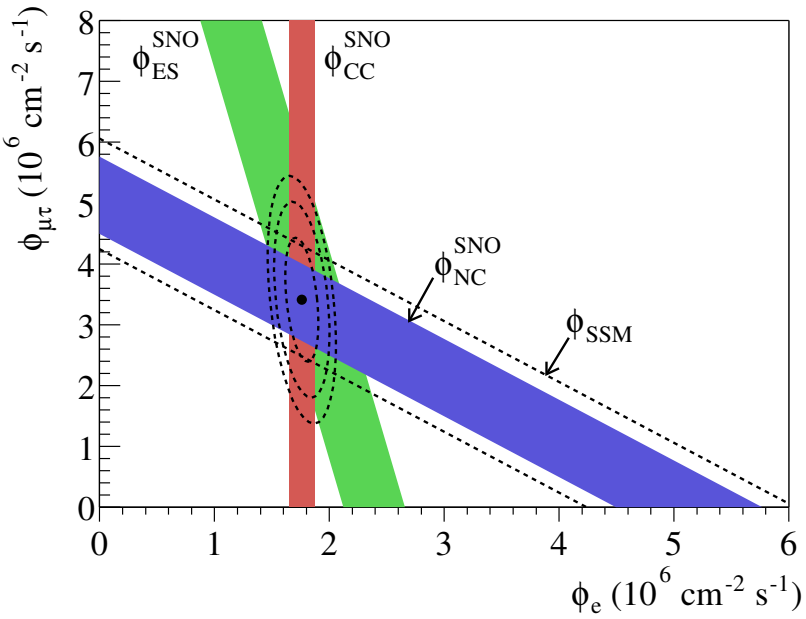


Figure 2.1: Solar neutrino flux composition as measured by the SNO experiment. Each of the coloured bands represents the measured flux of charged-current (CC), neutral-current (NC) and elastic scattering (ES) events, with a $\pm 1\sigma$ spread. The contours are used to represent the 68%, 95% and 99% probability contours for the joint fit between ϕ_e and $\phi_{\mu\tau}$. Finally, the dashed lines around the ϕ_{NC} represent the predicted flux of ^{8}B neutrinos based on the standard solar model. Figure is from [14].

produced by the charged leptons in water; however it is much larger in size, both in terms of target mass and the number of photomultiplier tubes to catch the Cherenkov light. The light produced can also be used to determine the charged lepton, with muons leaving clear Cherenkov rings due to their higher mass, and electrons leaving more diffuse rings, due to the tendency to scatter and shower. In 1998, Super-Kamiokande published their results, showing the atmospheric muon neutrino flux as a function of azimuthal angle [15], seen in Figure 2.2. The observed results can be easily explained under the hypothesis of neutrino oscillations. The neutrinos arriving from above have travelled around 15 km, which is used to calculate an oscillation probability, alongside measurements of Δm^2 and an understanding of the average energy of atmospheric neutrinos, the oscillation probability for neutrinos coming from above is closer to zero. However, for neutrinos coming up, travelling anywhere up to around 13000 km, the probability will be much higher. Furthermore, as the ν_μ are reduced but ν_e sees no enhancement, it suggests that the dominant oscillation mode for atmospheric neutrinos is $\nu_\mu \rightarrow \nu_\tau$. However, Super-Kamiokande was not able to easily detect ν_τ at the time, so was unable to check this option itself. More recently, Super-Kamiokande has been able to exclude no ν_τ appearance at the 4.6σ level [16].

2.2 Computing And Neutrino Experiments

Physics, especially particle physics, has played a key role in the history of computing. As experiments in particle physics moved past calculations that were too complex for a single physicist or a team of physicists to calculate, there has been a need for further computing power. As experiments get bigger, their requirements on sensitivity gets higher, and the data they produce gets even larger, more and more computing power is needed. However, this is not a one-way street, as many of the advances in computing have also helped unlock new scientific discoveries and provide a more profound insight into physics data. It is this influence that computing has had on particle physics, specifically neutrino experiments, that will be delved into here, giving an overview of the history of computing and neutrino experiments, to give some context to later work using modern data science techniques on a next generation neutrino experiment. The timeline outlined here is meant to outline the broad trends of analysis software, rather than the exact transition points between computing paradigms, and also

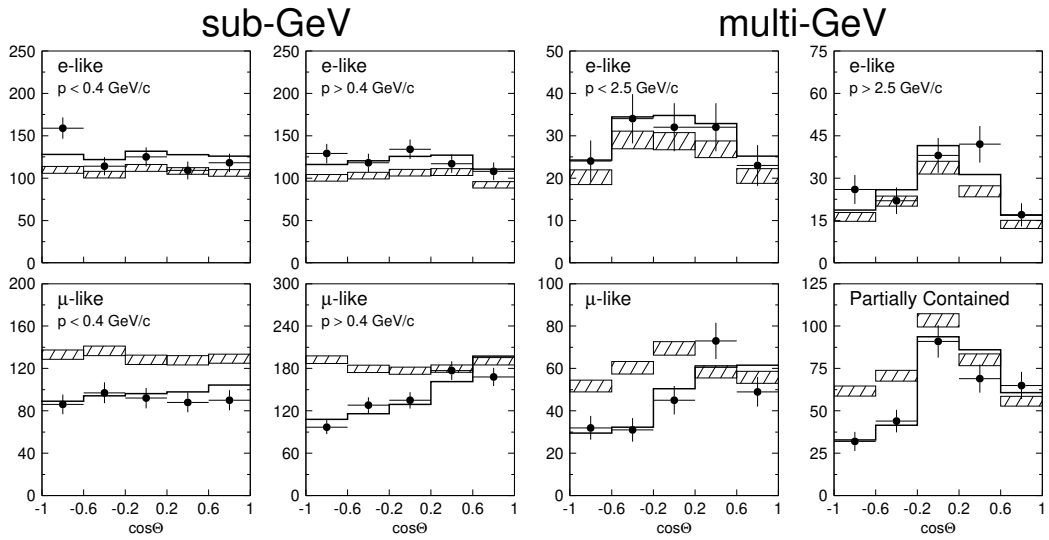


Figure 2.2: The zenith angle distributions of muon and electron like events at Super-Kamiokande. Here, upward going particles have $\cos\theta < 0$ and downward-going particles have $\cos\theta > 0$. The particle-like and the energy range for each subplot is given in the top left. The hatched region shows the MC expectation for no oscillations, normalised to the data live-time with statistical errors. The best-fit expectation for $\nu_\mu \to \nu_\tau$ oscillations is shown as the bold line, with the overall flux normalisation fitted as a free parameter. Figure is from [15].

focuses on analysis software only. Particle physics has many other problems that depend on computing heavily, such as the immense requirements on network and storage infrastructure that modern neutrino experiments need to deal with super novae, but that lies outside the scope of this work.

Early neutrino experiments are older than the widespread use of computing, with experiments of that time relying on manual calculations and technical drawings to show results. For results such as those in "Observation of High-Energy Neutrino Reactions and the Existence of Two Kinds of Neutrinos" [5], the amount of data is easily understandable without any help, for example the paper quotes a number of 113 events passing their geometric cuts. For data rates of that level, it is understandable that additional help is not required, but also that experiments of that time had to be designed with those constraints in mind. More modern experiments can target incredibly high data rates, as it is known that the computing infrastructure is there to support it.

Following from this, there is the start of a period of transition, moving from the purely analogue methods used previously, into more and more usage of online, computing-based analysis. For the results outlined in "High energy neutrino and

antineutrino interactions in a neon filled bubble chamber" [17], the bubble chamber camera films are scanned in, and processed with both a PDP-8 minicomputer, the first commercially successful mini-computer, and the PDP10 mainframe. Data from experiments is more and more likely to remain in a purely digital form, though often with some physical redundancy as a physical copy too, when available.

As we move from the 1970s to the 1980s, interactive data analysis is now becoming a common tool used in particle physics. The personal workstation, rather than a shared mainframe became more common towards the second half of the 1980s, meaning physicists were able to work on results interactively, with CERN, DESY, and SLAC each having their own frameworks for interactive data analysis in the mid-1980s, and then Physics Analysis Workstation [18] being released in 1986 at CERN as a more general framework for multiple experiments. Looking at Kamiokande-III in 1995, computing forms a key part of the experiment, as shown in "Measurement of solar neutrinos from 1000 days of data at Kamiokande-III" [19]. It is used extensively, for online monitoring, simulation of the detector, as well as manual reconstruction of events, calculating the interaction location and direction, its energy and more using bespoke handwritten algorithms. This outlines how key computing had already become in the early 90s, with datasets of 1000 days, containing almost five thousand events, being infeasible to process without the help of computing, especially if further data from other runs or experiments is also included.

If we move forward to 1999 and look at SNO, we can get an idea of how analyses changed due to advances in computing, both in terms of available computing power, but also techniques. Whilst Kamiokande utilises computing for infrastructure and simulation and reconstruction, SNO was able to utilise advances in computing even more. For example, looking at "Search for Neutron Anti-neutron Oscillation at the Sudbury Neutrino Observatory" [20], we can see some more basic improvements such as more comprehensive 3D visualisation and plots, as well as algorithmic improvements both to analysis and the underlying software used, for example the particle simulation libraries becoming more advanced, with more advanced techniques for fitting the ring-like structures found in the detector. It is also interesting to see how experiments evolve, with some early SNO techniques being similar to those outlined in Kamiokande-III, but later results starting to use more modern techniques, and the development of these techniques becoming a larger part of a physicists work, rather than simply a quick tool to

unlock a certain analysis variable.

This is especially visible at Super-Kamiokande, as it has been in operation since 1996, such that the computing methods used have changed fairly drastically over its lifetime. For example, early analysis at Super-Kamiokande utilised ring fitting tools for particle identification [21], calculating variables based on the ring pattern and angles relative to an interaction vertex to produce a score that can be selected on, with additional terms added later. If we move forward again, analyses at Super-Kamiokande [22], MiniBooNE [23] and more are now using advanced techniques, including early machine learning techniques such as shallow neural networks and decision trees and support vector machines, used for particle identification and more, to improve upon the performance achieved using handwritten scores. Again, this was only made possible by leaps in computing, enabling the fast processing of many tens of thousands of events to build more capable classifiers.

It is also interesting to look at the generational leaps in technology requirements, moving from a trigger rate of around 1 Hz to 12 Hz at Super-Kamiokande. Here, technology and the experiments grow together, with the bigger experiment being possible in part due to the advances in computing, netting a result of around 1 million events a day, pre-filtering. More computing power available means that more data can be safely produced and analysed.

Throughout all this time, papers on computing and analysis techniques at experiments are becoming more and more common, PhD theses are having a greater focus on the production and upgrading of software analysis tools, further highlighting the deep ties between experimental particle physics and computing.

Finally, we can move forward even further, looking at the NOvA experiment, representing the most recent jump in computing paradigm, with its deep learning-based classifier for neutrino event selection [24], based on earlier studies at the LHC [25] and Daya Bay [26]. Here, over 4.7 million events are used for the full training and testing process, utilising over a week of processing time to train the network on modern hardware for the time. This sort of deep learning work is only feasible due to the advancements in computing, with machine learning techniques themselves dating back much earlier in computing history, but only made widely and easily achievable by further computing hardware and software advances that meant the vast processing it requires was feasible in useful time frames. This is made even clearer comparing the sort of hardware quoted for Kamiokande-III [19], with the VPX210/10S used for analysis work, with a quoted speed of

285 Mflops, compared to the Nvidia K40 used for the NOvA CVN which tops out at 4290000 Mflops, and two of these was used for the training process for a week. This vast difference helps highlight just how infeasible certain styles of work are without computing and particle physics keeping pace with each other. A further example of this can be seen in Figure 2.3, showing how the total core hours used at the Open Science Grid (OSG) [27, 28] has changed over time since 2006. The largest users in 2022 use more CPU hours than every user combined earlier in the OSGs life.

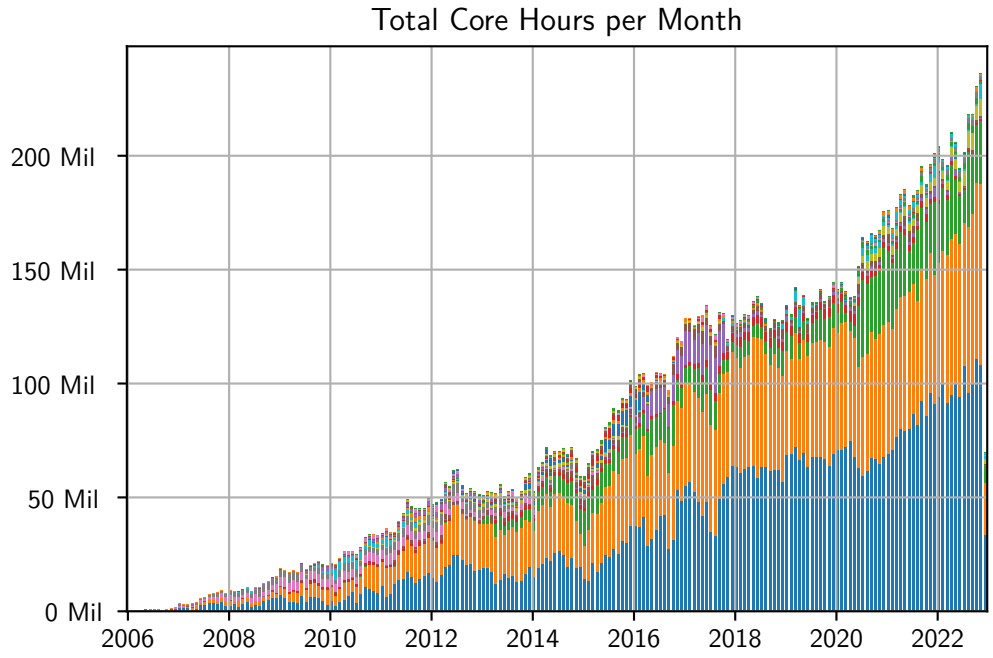


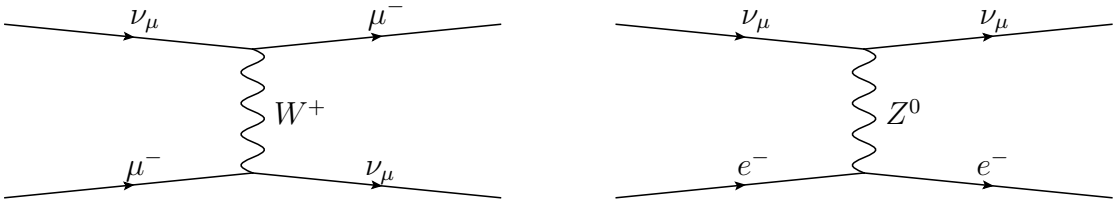
Figure 2.3: The total core hours used at the OSG since 2006. Colours are used to distinguish between different users of the OSG, mostly large particle physics experiments or labs. A clear trend can be seen, with more and more millions of CPU hours being used each month, both by existing experiments requirements growing, and new experiments starting with high computing requirements. Data taken from [29], plotted in Python.

This trend of more and more computing being required will only continue, with experiments needing higher resolutions, more interactions, faster software and pushing efficiencies as close to 100% as possible. Deep learning is looking to form an even larger portion of the required computing power at experiments now, bringing new complications to writing software that effectively utilises graphics cards compute power, as well as the logistical constraints of enabling access to

graphics cards, which drastically speed up machine learning, across many labs and universities. This issue of distributed computing will also only get more difficult and more necessary, as it is becoming quickly infeasible to have a single institute host all the storage and compute required for an experiment or experiments.

2.3 Neutrino Interactions

In the standard model, neutrinos can only interact via the weak force, as they do not have electric charge or colour charge. Neutrino interactions can proceed in two ways, via either a charged-current (CC) or neutral-current (NC) interaction, mediated through the W and Z boson respectively. In CC interactions, a neutrino interacts with either a quark or a lepton via the exchange of a W boson, producing a charged lepton the same flavour as the initial neutrino. In contrast, in a NC interaction, the neutrino scatters off a target via the exchange of a Z boson and remains intact.



(a) Charged-current scattering. **(b)** Neutral-current scattering.

Figure 2.4: Charged-current and neutral-current interaction Feynman diagrams.

Charged current interactions are of particular importance to experiments, as in this type of interaction the flavour of the neutrino can be determined. The charged current interactions can be broken down further into multiple interaction types, where three are particularly important.

Quasi-elastic (QE)

Quasi-elastic interactions are the dominant type of interaction for neutrinos below 1 GeV. In a quasi-elastic interaction, a neutrino scatters off an individual nucleon, and when energies are high enough the nucleon can be liberated from the nucleus. Most commonly, this produces a final-state hadronic system that contains a single proton.

An example of this interaction is shown in Figure 2.6a, which shows a ν_μ interacting with a neutron, which changes the flavour of one quark in the neutron from d to u, resulting in a μ^- and a proton in the final state of the interaction.

Resonance (RES)

Resonance interactions are the most common type of interaction in the 1 – 5 GeV energy range. In a resonance interaction, a neutrino excites the target nucleon into a resonance that subsequently decay. The final-state particles commonly include mesons, usually pions, produced by the decays of the resonant state. An example of a CC RES interaction can be seen in Figure 2.6b, where the resonant state results in a neutron and a charged pion being produced in the final state. There are also coherent processes, in which mesons are produced directly by interactions within the nucleon without the creation of a resonant state.

Deep inelastic scattering (DIS)

Deep inelastic scattering is the most common interaction type for neutrinos that are above 5 GeV. In a deep inelastic scattering interaction, the neutrino has sufficient momentum transfer to resolve the individual quarks within the target nucleon. The subsequent interaction may liberate a quark, which hadronises inside the nucleus to produce final-state systems containing multiple mesons. This interaction can be seen in Figure 2.6c, with the hadronic particle shower labelled as X.

The cross-sections for these different interaction types vary as a function of energy, with the preferred energy range stated above for each interaction type. An example of the relevant cross-sections per nucleon for a muon neutrino are shown in Figure 2.5.

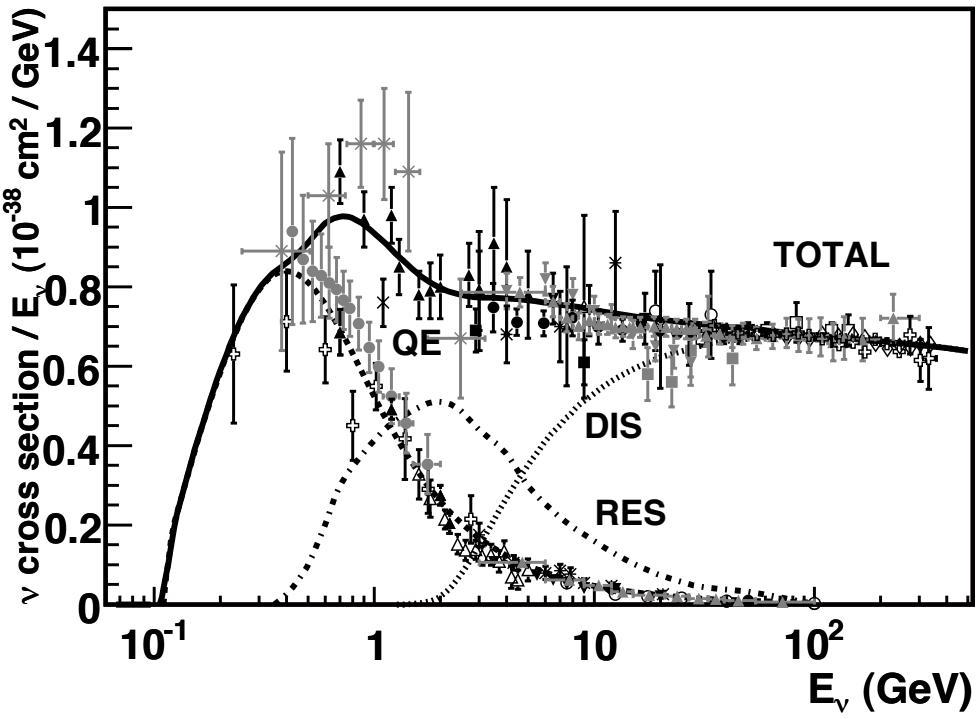
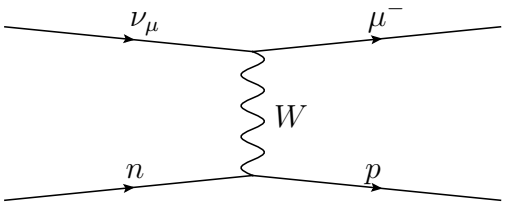
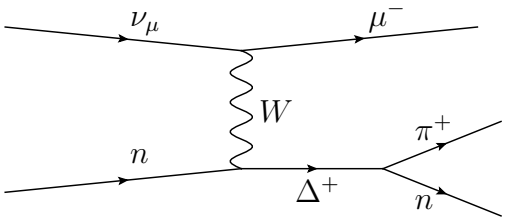


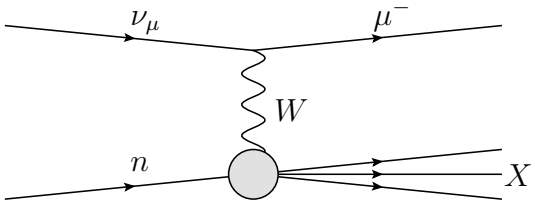
Figure 2.5: Predicted muon neutrino cross-sections as a function of energy for CC interactions, split into QE, RES and DIS. Figure taken from [30]. The region of interest for DUNE is between 1 and 10 GeV.



(a) ν_μ CC QE interaction.



(b) ν_μ CC RES interaction.



(c) ν_μ CC DIS interaction.

Figure 2.6: Feynman diagrams showing examples of ν_μ CC QE, RES and DIS interactions.

3

The Deep Underground Neutrino Experiment

"Words can be like X-rays if you use them properly - they'll go through anything. You read and you're pierced."

Helmholtz Watson - Brave New World

Deep Underground Neutrino Experiment (DUNE) is a next generation long-baseline neutrino oscillation experiment, that is currently in the construction phase. The primary goal of DUNE is to provide a definitive measurement of δ_{CP}, though there is also a rich physics program including nucleon decay, supernovae neutrinos and more. DUNE is an international experiment, hosted by the U.S. Department of Energy at the Fermi National Accelerator Laboratory (Fermilab), Illinois. It will be composed of three main components, a far detector (FD) located around 1.5 km underground at the Sanford Underground Research Facility (SURF) in South Dakota, USA, at a baseline of 1300 km from Fermilab, and both a near detector (ND) complex and the facilities for the neutrino beam which will be hosted at Fermilab. DUNE will utilise multiple large-scale liquid argon time projection chambers (LArTPCs), with a total mass of over 70 kt of liquid argon (LAr), with at least 40 kt being instrumented, and the total mass being split over multiple modular detectors.

A key part of DUNE is the accelerator neutrino beam, which will be manufactured by the Long Baseline Neutrino Facility (LBNF) at Fermilab, which will also manage the infrastructure and sites for both the ND and FD. The beam

will be the world's most intense neutrino beam, passing through a high-precision near detector suite 574 m away from the primary target. The ND will measure the energy spectrum and flavour composition of the wide-band neutrino beam. Accurate comparisons of the ND and FD spectrum and composition will be crucial to the discovery of new phenomenon. A diagram showing a high-level overview of these components can be seen in Figure 3.1.

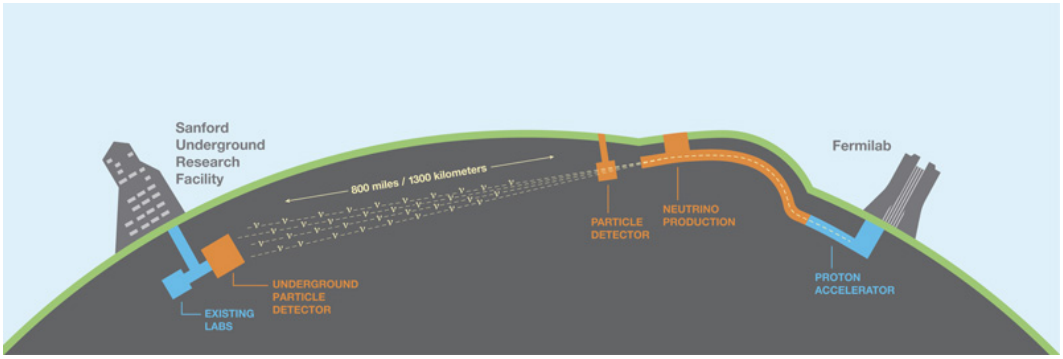


Figure 3.1: The LBNF-DUNE baseline, showing both the near site at Fermilab and far site at SURF [31].

3.1 Physics Goals

As outlined in Chapter 2, neutrino physics has advanced significantly in recent years. The discovery that neutrinos oscillate, which clearly showed that the Standard Model was incomplete, has opened a new window on particle physics, and raised many new questions that will be addressed by DUNE. DUNE, once fully built, will attempt to answer some remaining questions around neutrino oscillations.

The DUNE physics programme is split into a series of primary and secondary objectives. The primary physics goals for the DUNE FD are [32]:

- World-leading measurements of neutrino oscillation parameters through precise observations of muon neutrino disappearance and electron neutrino appearance. In particular, DUNE aims to determine the neutrino mass hierarchy and observe CP violation in the neutrino sector. Figure 3.2 shows the predicted impact of δ_{CP} on neutrino oscillations at DUNE.
- Evidence of baryon number violation, specifically via proton decay and other baryon number violating processes.

- Measurement of the ν_e flux from a core-collapse supernova within our galaxy, subject to one occurring during the operational lifetime of DUNE.

DUNE will be able to carry out this comprehensive study using both ν_μ and $\bar{\nu}_\mu$ beams from Fermilab. There is also a range of secondary physics goals, as well as physics goals that are specific to the DUNE ND, due to its increased rate of neutrino interactions. These goals are:

- Various Beyond Standard Model (BSM) physics searches, including a search for sterile neutrinos, searches for non-unitarity of the PMNS matrix, and searches for nonstandard interactions (NSI).
- Neutrino interaction physics at the near detector (ND), including neutrino interaction cross-sections and nuclear effects.
- Tau neutrino appearance measurements.
- Atmospheric neutrinos as a tool for measuring neutrino oscillation phenomena.

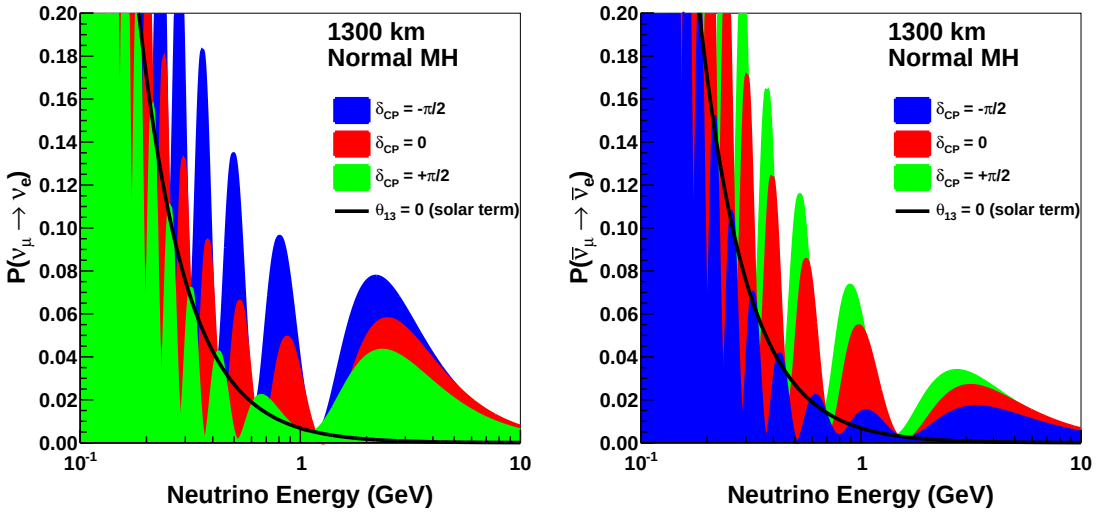


Figure 3.2: The effect that varying the value of δ_{CP} has on the oscillation probability of $\nu_\mu \rightarrow \nu_e$ and $\bar{\nu}_\mu \rightarrow \bar{\nu}_e$, using the DUNE baseline of 1300 km and assuming the normal mass ordering [31].

A next generation neutrino oscillation experiment, DUNE is in a position to measure CP violation to better than three standard deviations (3σ), over more

Physics Milestone	**Exposure** (staged years, $\sin^2(\theta_{23}) = 0.580$)
5σ Mass Ordering δ_{CP} = -$\pi/2$	1
5σ Mass Ordering 100% of δ_{CP} values	2
3σ CP Violation δ_{CP} = -$\pi/2$	3
3σ CP Violation 50% of δ_{CP} values	5
5σ CP Violation δ_{CP} = -$\pi/2$	7
5σ CP Violation 50% of δ_{CP} values	10
3σ CP Violation 75% of δ_{CP} values	13
δ_{CP} Resolution of 10 degrees $\delta_{CP} = 0$	8
δ_{CP} Resolution of 20 degrees δ_{CP} = -$\pi/2$	12
$\sin^2 2\theta_{13}$ Resolution of 0.004	15

Table 3.1: The projected DUNE oscillation physics milestones. Exposure is given in years, assuming the true normal ordering and equal running in both neutrino and antineutrino mode, which is required to reach selected physics milestones in the nominal analysis, based on the best-fit oscillation values from NuFit 4.0 [33]. Using a value of $\sin^2\theta_{23} = 0.580$, with the reasoning for this explained further in Section 5.9.4 of [31], as well as any assumptions around staging. Exposures rounded to the nearest year. These milestones are taken from [31].

than 75% of the range of the possible values for δ_{CP}. This goal is outlined in the U.S. Particle Physics Project Prioritization Panel (P5) report [34], as well as in the recommendations of the European Strategy for Particle Physics (ESPP) which were adopted by the CERN Council in 2013, classifying the long-baseline (LBL) neutrino program as one of the four scientific objectives that required significant resources, commitment and sizeable collaborations. The search for CPV is paramount, with it potentially offering a better insight into the origin of matter-antimatter asymmetry in the universe [35]. Figure 3.3 shows an example of the significance of the DUNE determination of CPV sensitivity as a function of true δ_{CP} for different experimental run times.

3.2 LBNF

The Long Baseline Neutrino Facility (LBNF) project is the facility that will house and provide infrastructure for both the DUNE FD modules in SURF, South Dakota, and the ND at Fermilab in Illinois. The organisation and management of LBNF is separate to that of DUNE, though both are hosted by Fermilab, with the design and construction organised by both the US Department of Energy (DOE) and Fermilab, whilst also including international partners.

LBNF will provide DUNE with the range of facilities that it requires, as well as the required civil construction needed to excavate the complex cavern systems outlined in Figure 3.4. Specifically, LBNF will provide

- Both technical and conventional facilities required for the 1.2 MW neutrino beam that DUNE requires, utilising the PIP-II upgrade [36] to the existing Fermilab accelerator complex. PIP-II will provide the 1.0 to 1.2 MW beam of proton power from the Fermilab main injector at the start of DUNE operations, as well as providing a platform for a later upgrade to greater than 2 MW in the future. Currently, it is planned to upgrade the beam to provide 2.4 MW of beam power by 2030.

- All the conventional facilities to house and support the DUNE ND systems at Fermilab, as outlined in Figure 3.5.

- The required facilities and upgrades to infrastructure to support the DUNE FD, including the excavation of three underground caverns at SURF, as well

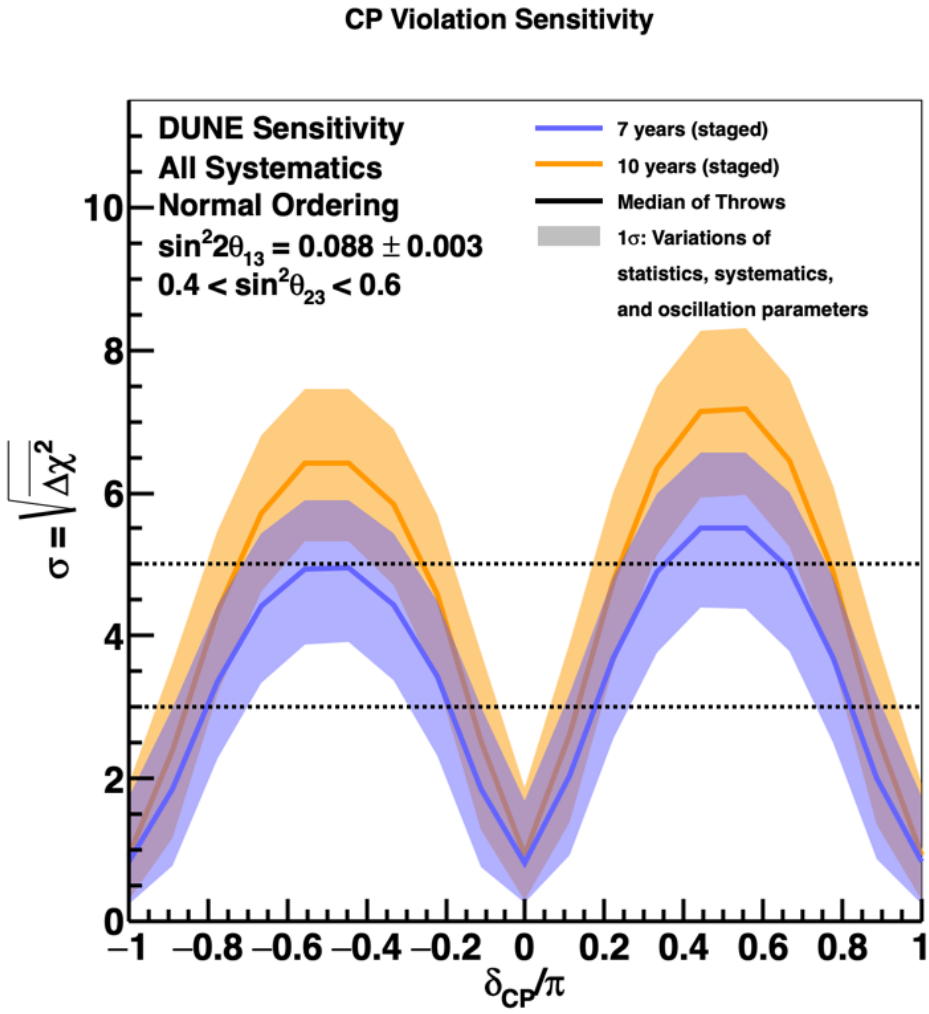


Figure 3.3: The significance of the DUNE determination of CP-violation as a function of the true value of δ_{CP} for both seven and ten years of exposure, in blue and orange, respectively. This plot assumes normal ordering, and the width of the bands cover 68% of fits, using random throws to simulate statistical variance, as well as to select true values of the oscillation and systematic uncertainty parameters, whilst constrained by pre-fit uncertainties. The solid lines here show the median sensitivity to CP-violation. Figure is from [31].

as surface, shaft, and underground infrastructure to support the installation of hardware into the two larger caverns. The third cavern will be used to house both cryogenics and data acquisition facilities, to support the four FD modules.

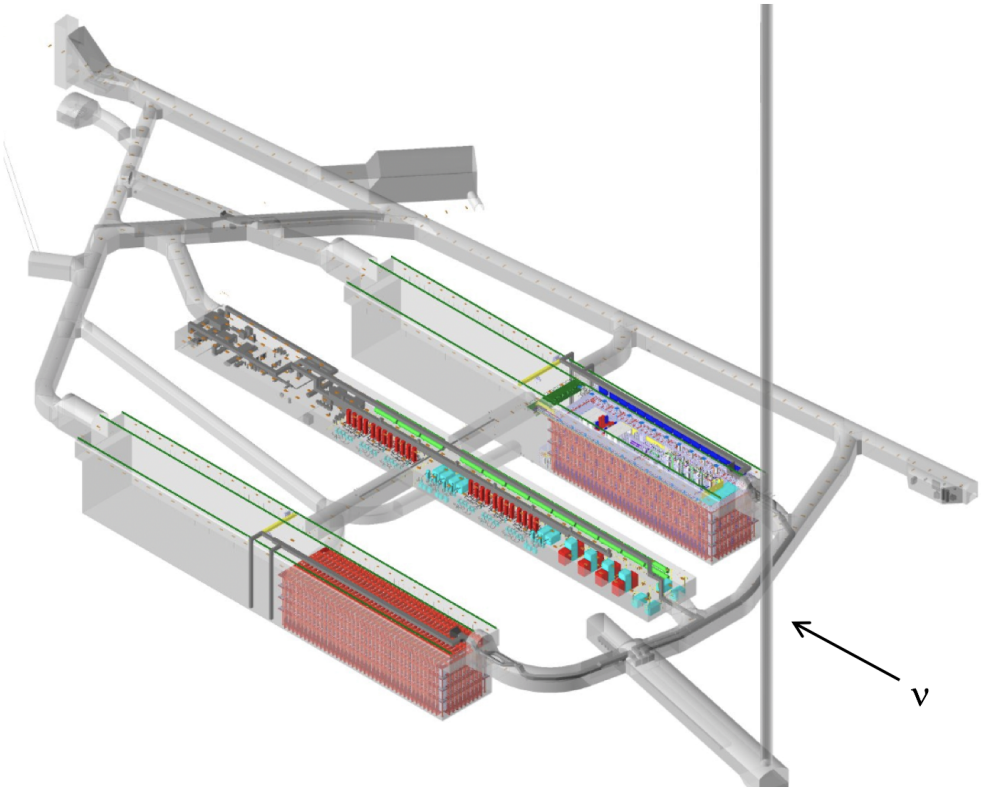


Figure 3.4: The cavern design for DUNE at SURF. There will be two caverns to store detectors, with the red boxes illustrating cryostats. In the middle, there is a central cavern where support system such as the DAQ and cryogenics systems will live. The direction of the beam can be seen in the bottom right. Figure from [31].

3.2.1 The LBNF Beamline and Target

The LBNF neutrino beam will be the world's most intense neutrino beam, and will deliver an estimated 1.1×10^{21} protons-on-target per year [32], once in operation [36]. This 1.0 – 1.2 MW proton beam will be directed on to a target, creating a wide-band on-axis neutrino beam in the direction of the DUNE FD modules at SURF.

Figure 3.5 shows the proposed LBNF beam site and ND hall. The 60 GeV to 120 GeV proton beam will be extracted from the Fermilab Main Injector, before

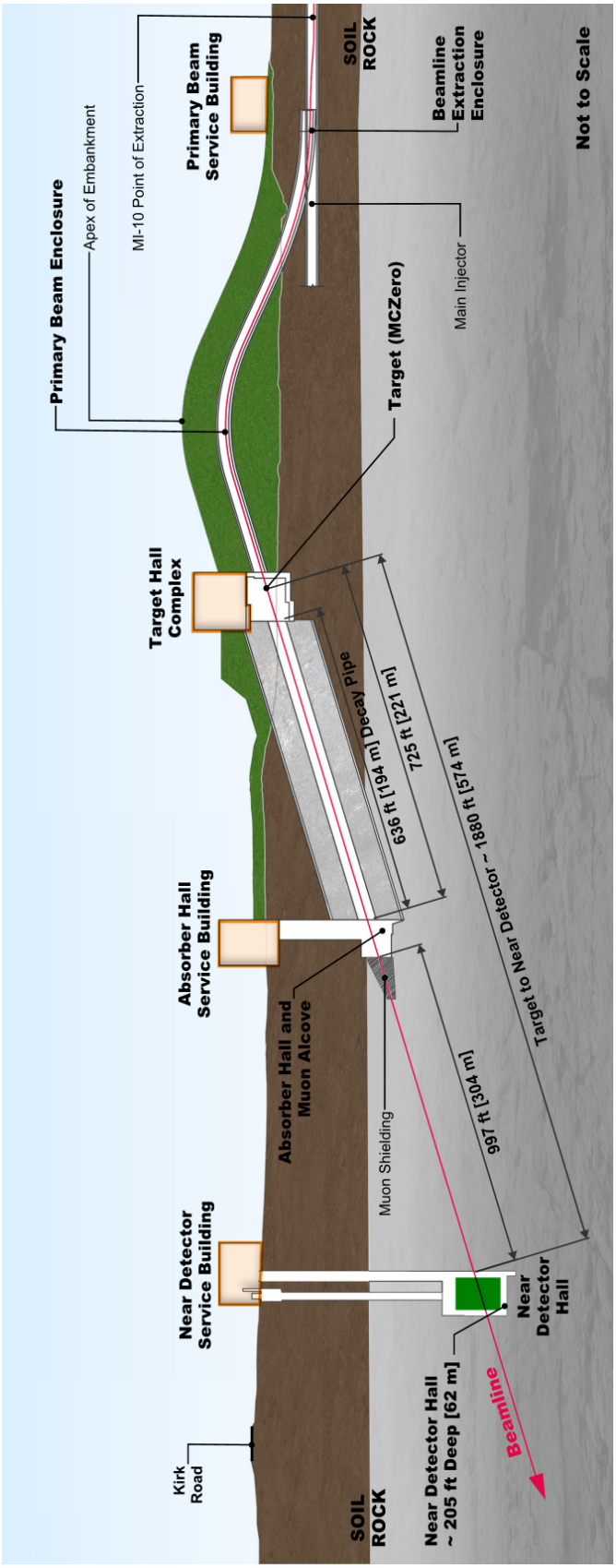


Figure 3.5: A side-on view of the LBNF beamline, target and ND hall. The proton beam can be seen entering from the right-hand side of the image. The beam comes from the Fermilab Main Injector, before being raised and then directed through the target. After the protons hit the target, the products from that interaction travel down a decay pipe, where they decay into neutrinos which can continue on to both the ND and FD facilities. Particles which have not decayed are instead blocked and absorbed. Figure taken from [32].

being carried over an embankment and directed towards the target hall and near detector facilities. The embankment is simply to allow the target hall to remain above ground, to help with costs, both in initial installation but also ongoing target maintenance going forwards. At the target hall, the proton beam collides with the target, producing a beam of secondary charged particles that are directed down a decay pipe, before finally decaying into the neutrinos that both the ND and FD modules will measure.

The secondary particle beam will mostly consist of both kaons and pions, which when they decay mostly produce ν_μ, but with a small contamination of ν_e, $\bar{\nu}_\mu$ and $\bar{\nu}_e$, as shown in Figure 3.6. This includes running in both forward horn current (FHC) and reverse horn current (RHC) modes, which focus either positively or negatively charged mesons, respectively, resulting in a neutrino-dominated and antineutrino-enhanced beam.

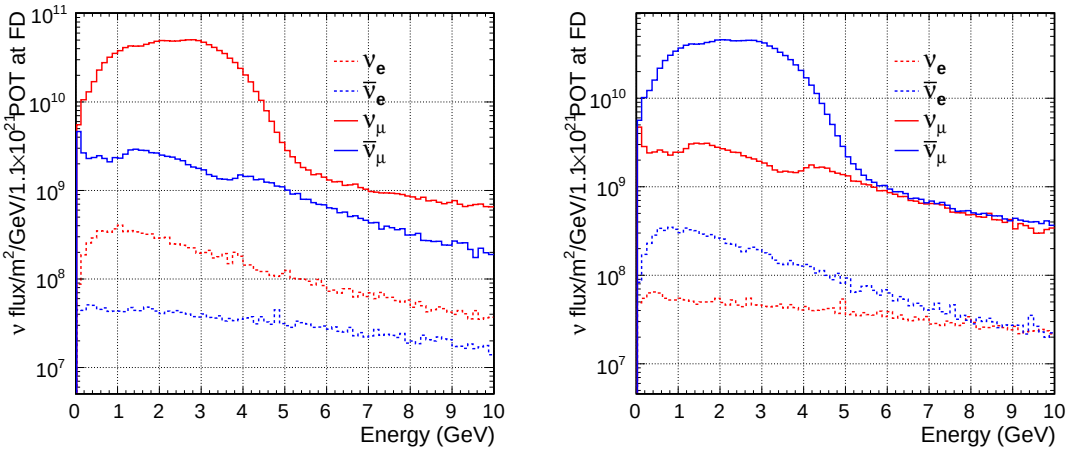


Figure 3.6: Neutrino fluxes at the FD for both neutrino mode (left) and antineutrino mode (right). The dominant component of the beam is ν_μ ($\bar{\nu}_\mu$), although there is more contamination of $\bar{\nu}_\mu$ (ν_μ) at lower energies. Figure is from [31].

3.3 Far Detector

The DUNE far detector will consist of four LArTPC modules of varying technologies, each containing approximately 17.5 kt of liquid argon and installed 1.5 km underground. Each module will have a fiducial mass of at least 10 kt, giving a total of in excess of 40 kt across the four modules. Each module will be situated in a cryostat with dimensions of 18.9 m (W) × 17.8 m (H) × 65.8 m (L).

The use of four modules will allow DUNE to retain flexibility in its construction and operations, in particular the phasing of its construction and the development of advanced LArTPC technology for the fourth module.

DUNE is currently planning and prototyping two LArTPC technologies, with the opportunity for further technologies to be used for the fourth module, which will be installed last. The two designs currently in prototype at CERN differ in how the LArTPC is orientated. The first prototype utilises a horizontal orientation, where ionisation charge is drifted horizontally to wires. The second prototype instead uses a vertical drift design, which has advantages such as being easier to construct and maintain, but at the cost of being a less developed technology. These two designs will be covered in more detail in Sections 3.3.2 and 3.3.3

As there will be multiple detector modules, DUNE has a staging plan for the installation of its detectors. The current plan is to phase the installation of modules as follows:

- Start of beam running: Two far detector modules and the 1.2 MW beam, with one horizontal drift and one vertical drift module.
- After one year: The addition of the third FD module, for a total fiducial mass of 30 kt. The choice of this module, between vertical or horizontal drift, is not fully confirmed.
- After three years: The addition of the fourth and final FD module, to reach the target of a 40 kt fiducial mass far detector. There is scope for the final module to utilise a different technology, rather than the vertical or horizontal LArTPC technology, as long as that technology allows DUNE to reach its physics goals.
- After six years: Upgrade the beam to 2.4 MW.

The rate of neutrino events that DUNE will observe in its far detectors will vary over time, as the fiducial mass increases to in excess of 40 kt. The expected event rates, including the staging plan, is outlined in Table 3.2, breaking down the backgrounds for each mode of running.

Before giving an extended description of the two main detector technologies in use at DUNE, it is necessary to outline what a LArTPC is, and the principles behind them. Following this, subsections 3.3.2 and 3.3.3 will give an explanation

	Expected Events (3.5 years staged)
ν mode	
ν_e Signal NO (IO)	1092 (497)
$\bar{\nu}_e$ Signal NO (IO)	18 (31)
Total Signal NO (IO)	1110 (528)
Beam $\nu_e + \bar{\nu}_e$ CC background	190
NC Background	81
$\nu_\tau + \bar{\nu}_\tau$ CC background	32
$\nu_\mu + \bar{\nu}_\mu$ CC background	14
Total background	317
$\bar{\nu}$ mode	
ν_e Signal NO (IO)	76 (36)
$\bar{\nu}_e$ Signal NO (IO)	224 (470)
Total Signal NO (IO)	300 (506)
Beam $\nu_e + \bar{\nu}_e$ CC background	117
NC Background	38
$\nu_\tau + \bar{\nu}_\tau$ CC background	20
$\nu_\mu + \bar{\nu}_\mu$ CC background	5
Total background	180

Table 3.2: Integrated rate of selected ν_e CC-like events between 0.5 and 8.0 GeV assuming a 3.5 year staged exposure in neutrino-beam mode and antineutrino-beam mode. Signal rates are shown for both normal ordering (NO) and inverted ordering (IO), whereas all backgrounds assume NO. δ_{CP} is assumed as 0. Table taken from [31].

of the current FD module designs, to give a better understanding of the specific LArTPC technology they will use, and how the technologies compare between the two detectors.

3.3.1 Liquid Argon Time Projection Chambers

The time projection chamber (TPC) was first proposed by David Nygren in 1974 [37, 38], where it was initially conceived as a gas-based drift chamber, combined with a multiwire proportional chamber which was designed to sit around the beam pipe of particle colliders. This design was then iterated on later by Carlo Rubbia in 1977, with the suggestion of liquid argon being used as the medium instead of a gaseous medium, meaning it could function as both target and detection medium at once [39].

A LArTPC contains a large volume of liquid argon that has an electric field across it. When a charged particle transverses the liquid argon, it produces two different types of energy deposition: a trail of ionisation electrons along its path, liberated from the liquid argon through the process of ionisation; and prompt ultraviolet scintillation photons. Due to the electric field, the ionisation electrons drift towards the anode plane of the TPC, where they can induce electrical signals. These electrical signals can be read out and stored for later analysis if the anode contains instrumented readout planes. The scintillation photons can propagate through the argon, as liquid argon is transparent to its own scintillation light. These prompt photons are collected by a photon detection system (PDS), where they provide a tagged start time for the event.

Figure 3.7 outlines the LArTPC detection principle. The anode in a modern LArTPCs will commonly have multiple readout planes, with each plane at differing relative angles. For this to work, it is necessary to have only a single collection plane, that is a single readout plane that absorbs the ionisation electrons, receiving a uni-polar signal. Any other readout planes instead receive an induced bi–polar signal due to the ionisation electrons passing the induction wires. Having multiple readout planes allows the reading out of a single event in multiple different orientations, which can be used to reconstruct the missing coordinates for a full 3D reconstruction of the event. The exact angle between the planes can differ per experiment, as can the relative resolution of the detector, which is set by the spacing between each readout channel in the plane. The readout channels themselves can also vary, with most LArTPCs using closely spaced wires as readout

channels, but other designs use thin metal strips or even pixel-based readouts. As an example, in the MicroBooNE experiment [40], the APA consists of 3 readout planes, with each plane having wires every 3 mm, and the orientation of the planes relative to the vertical set to $+60°, -60°, 0°$ for the U, V and Y planes respectively, with the Y plane corresponding to the final collection plane[1]. To ensure that the collection plane is reached, and the ionisation electrons drift past the first two induction planes unimpeded, a bias voltage is applied over the three planes, ensuring that the ionisation electrons drift past the first two planes before being collected.

Property Name	**Alternative Names**
Induction Plane	U and V Planes
Collection Plane	Y, W, X or Z Plane
	Real-World Representation
X Coordinate	Drift Direction
Y Coordinate	Detector Height (Not measured by readout)
Z Coordinate	Beam Direction & Readout Plane Number

Table 3.3: A reference for LArTPC naming conventions, covering both alternative wire plane names, and the most common coordinate system for a horizontal LArTPC.

3.3.2 Horizontal Drift

One of the first modules to be installed at SURF will be a horizontal drift LArTPC module, due to its more mature design and the prototyping of the HD technology at CERN being further developed. As the ionisation charge is read directly from the liquid, very low-noise electronics are required to achieve a high signal-to-noise (S/N) rate, which ProtoDUNE-SP can verify. This technology was originally pioneered by the ICARUS experiment, and has since had many years of international development. It is currently used in MicroBooNE experiment [40] at Fermilab, as well as the upcoming SBND detector which is part of the Fermilab Short-Baseline Neutrino (SBN) program [42].

A DUNE horizontal drift LArTPC is situated inside a large cryostat, as shown in Figure 3.8, with three module-length instrumented anode planes, constructed

[1]Naming conventions for the planes can differ across experiments, with some using U, V, Y others U, V, W or U, V, X and U, V, Z.

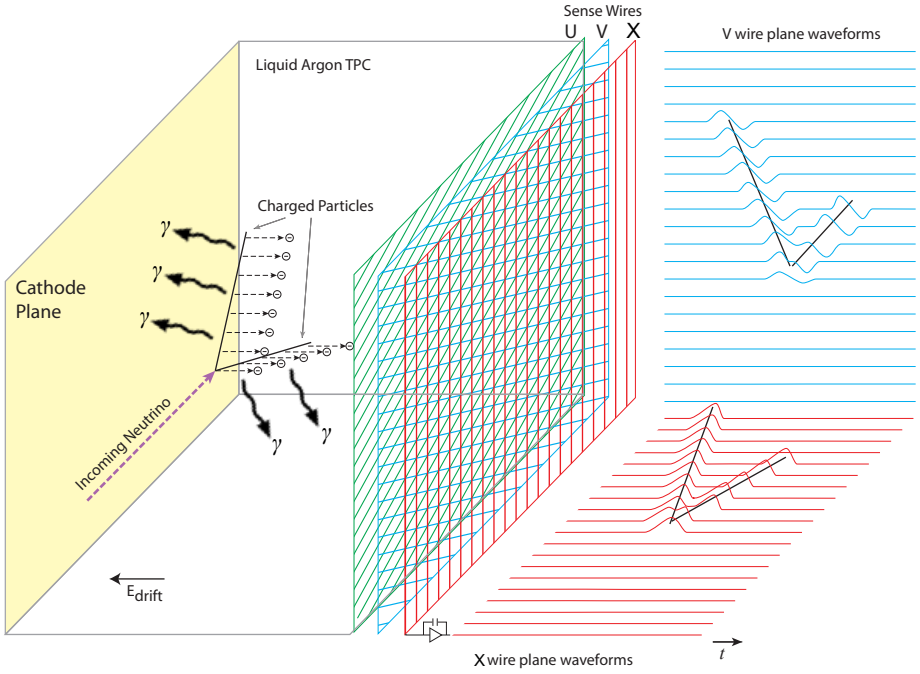


Figure 3.7: An illustration that shows the basic workings of a horizontal drift LArTPC. The dashed line shows an incoming neutrino, which interacts, and the resulting charged particles liberate ionisation electrons from the argon, which are drifted under an electric field to a set of wire planes, which can be read out. The photons produced as part of the neutrino interaction can also be seen. Figure taken from [41].

from 6 m high by 2.3 m wide anode plane assemblies (APAs), stacked two APAs high and 25 wide, for a total of 50 APAs per plane and 150 total, to account for the multiple drift volumes that the HD LArTPC will have. Figure 3.9 shows a DUNE APA. Each APA will have four total wire planes, two induction, one collection and a final grid plane. The two induction views are at $\pm 35.7°$ to the vertical, with the collection and grid plane being vertically orientated, and the wire spacing on the layers being around 5 mm. The grid plane is the first of the wire planes, offering shielding for the remaining active planes, significantly improving the signal shapes seen on the U induction plane by shielding it from the incoming drift electrons. This totals 2560 active wires to be read out per APA (with an additional 960 grid plane wires that are not read out). Alongside the APAs, cathode plane assemblies (CPAs) are used to complete the electric field. Each CPA will be held at -180 kV, with the APAs being held near ground, resulting in a uniform 500 V/cm E field across each of the 3.5 m drift volumes. A field cage (FC) is used to ensure the E field remains uniform at the open sides of the TPC. Figure 3.8 shows how this all looks in a completed module, indicating the APAs, CPAs, and part of the FC, as well as the multiple drift volumes that the HD modules will have. As can be seen in Figure 3.9, a DUNE APA can receive signal from either side, which is dealt with by having the wires wrap fully around the structure, which is shown at the top and bottom of Figure 3.9. There it can be seen there are darker regions of purple and green, representing wires that physically wrap around the APA, rather than stopping. This wrapping does lead to additional complexity, both in construction and reconstruction, though the trade-off is deemed worth it to help simplify the construction of the APAs compared to having additional sets of wires on each side of the collection plane.

The produced scintillation photons will be collected in novel photon detector (PD) modules, based on the light-trap concept, known as ARAPUCA [43–45], which utilises dichroic filters, wavelength-shifting plates and a silicon photomultiplier (SiPM) read-out. For the HD modules, the PD modules are placed in the inactive space between the innermost wire planes of the APAs. There will be 10 of these modules per APA, for a total of 1500 per HD module. The PD modules mounted in the innermost APA must collect light from both directions, whereas the PD modules mounted in APA frames near the cryostat walls only need to collect light from one direction.

Table 3.4 indicates the key specifications for the HD module, including its total

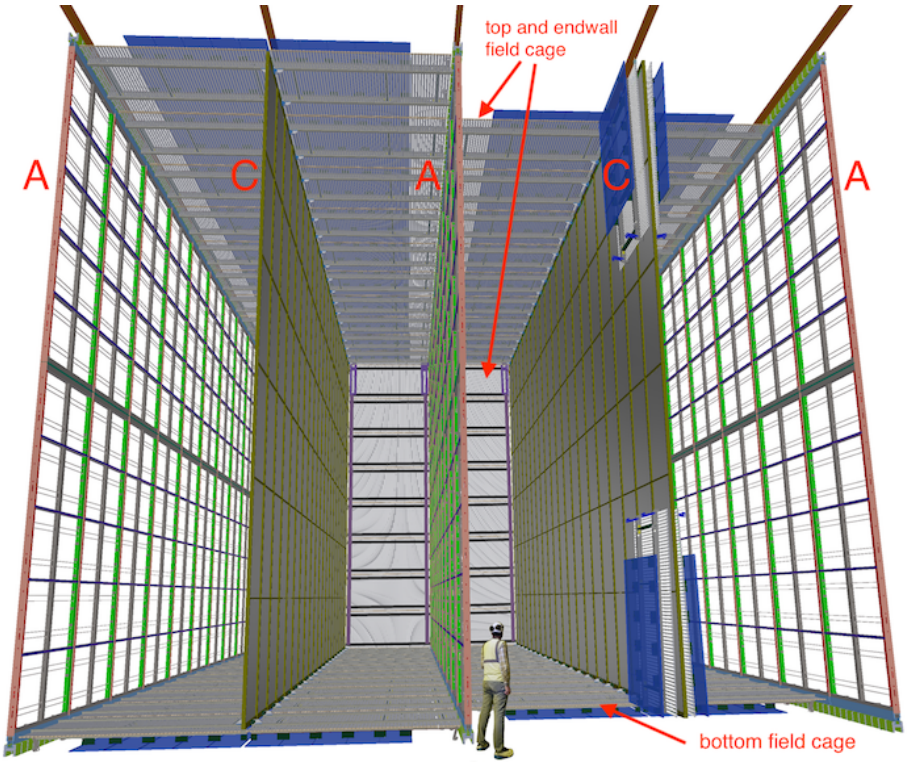


Figure 3.8: The internal layout of a horizontal drift LArTPC with multiple drift volumes. For DUNE, the anodes (labelled A) are built of two vertically stacked anode plane assemblies (APAs), whereas the cathode (labelled C) are three cathode plane assemblies (CPAs) high. Additionally, the field cage can be seen around the edges of the detector [41].

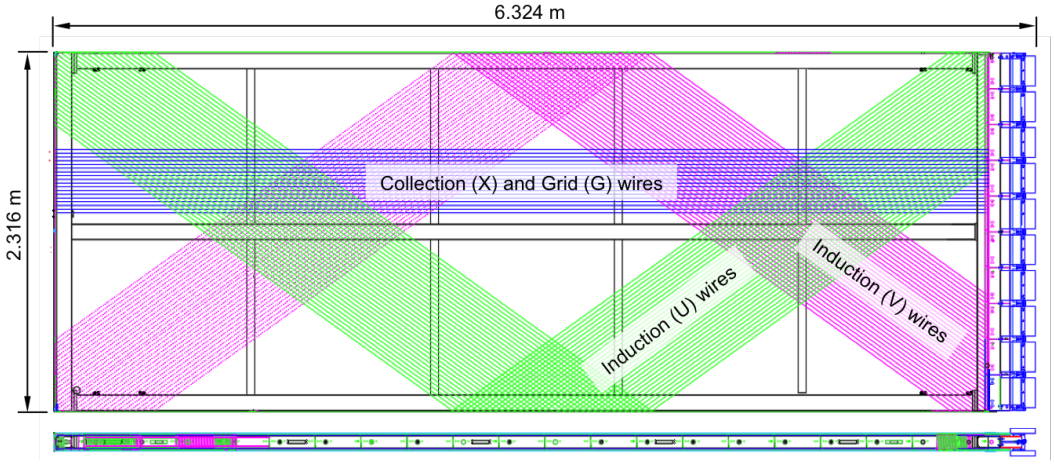


Figure 3.9: Schematic for a DUNE APA. Here, the collection and grid plane wires can be seen in blue, whereas purple and green represent the two induction planes. Readout electronics can be seen on the right-hand edge. Figure is taken from [41].

Item	Quantity
TPC size	12.0 m × 14.0 m × 58.2 m
Nominal fiducial mass	10 kt
APA size	6 m × 2.3 m
CPA size	1.2 m × 4 m
Number of APAs	150
Number of CPAs	300
Number of X-ARAPUCA PD bars	1500
X-ARAPUCA PD bar size	209 cm × 12 cm × 2 cm
Design voltage	-180 kV
Design drift field	500 Vcm^{-1}
Drift length	3.5 m
Drift speed	1.6 mm $\mu\mathrm{s}^{-1}$

Table 3.4: Key parameters for a 10 kt FD HD module. Table taken from [31].

size, mass, and requirements on the electron drift length and speed.

3.3.3 Vertical Drift

Alongside the DUNE HD detector design, the second detector technology to be deployed at SURF will be the vertical drift (VD) detector. This design evolved from an earlier DUNE dual-phase (DP) design, which also included a secondary gaseous phase that could be used to increase the signal by developing avalanches in the gas phase. However, a DP design also required the cathode to operate at much higher voltages and high voltages to be present in the gas phase. A purely vertical drift detector with no secondary phase avoids this issue, but keeps benefits like simpler detector construction and installation, at reduced costs, whilst also allowing the drift distance to be shorter compared to a DP design, due to top and bottom mounted readouts.

The VD detector has a few key differences to the HD design, most obviously the drift direction of the ionisation electrons, which lengthens the drift distance. However, both ProtoDUNE detectors have proven that electron lifetimes well above a few milliseconds are possible, with ProtoDUNE-SP achieving lifetimes in the tens of milliseconds. The ProtoDUNE-DP design had additional issues that

meant it was sensitive to detector environmental effects, such as disturbances on the LAr surface either due to movement in the LAr, or from contaminants.

One key part of the vertical drift design was the need for a different sort of read out electronics, rather than using a wire-based APA as in the HD module design. ProtoDUNE-DP instead uses a charge readout plane (CRP), a different anode design that does not use wires, but rather two double-sided printed circuit boards (PCBs). These PCBs are attached to a frame to form the CRP, which will be attached to the roof and floor of the VD cryostat. The innermost face of the PCB, that is the side that directly faces the cathode, has a guard plane to absorb unexpected discharges, whereas the reverse side is etched with strips that form the first of three readout planes, the first induction plane. The second PCB has the second induction plane on one side and the final collection plane on the reverse. These three sets of electrode strips are segmented to around a 5 mm pitch, set at different angles relative to each other to provide charge readout from different projections, similar to the HD APA design, though with 3200 readout channels per CRP, rather than the 2560 per APA. Figure 3.10 shows a CRP and the super structure that holds multiple of them together, whilst Figure 3.11 shows how all these elements come together to produce a full VD FD module.

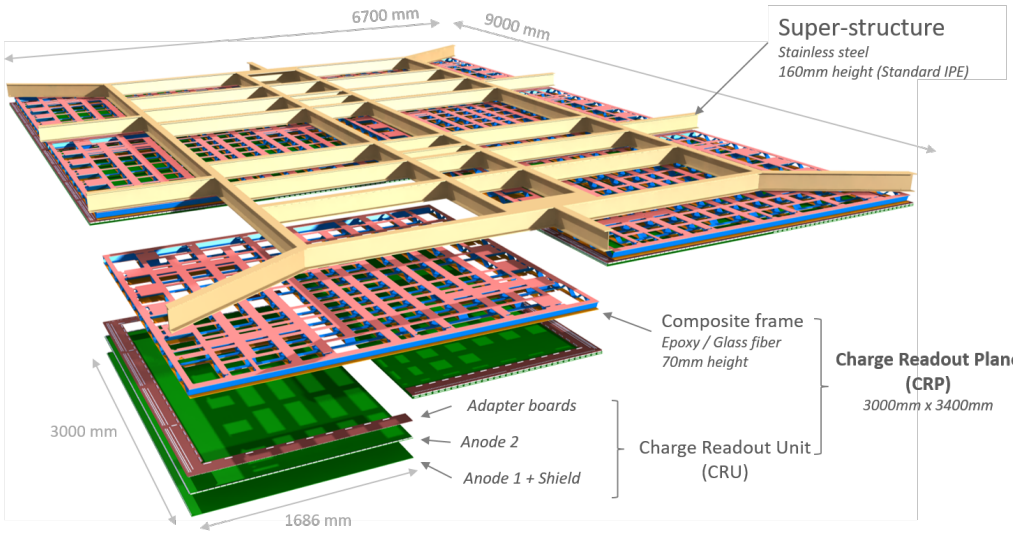


Figure 3.10: An exploded view of a CRP and its associated top superstructure. The multiple anode PCBs can be seen on the left, and how they are tiled together to produce a full CRP. Figure is internal to DUNE.

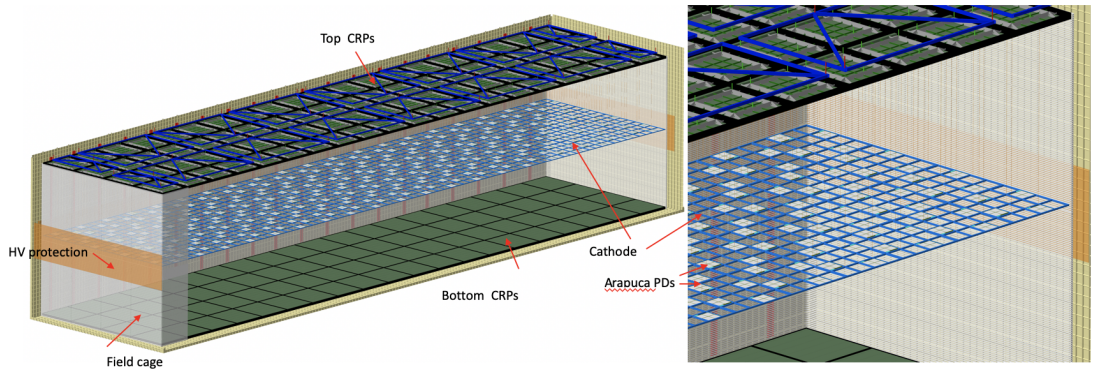


Figure 3.11: The DUNE VD detector design, showing both the anode and cathode planes, as well as the location of the field cage. Figure is internal to DUNE.

3.4 Near Detector

For a long-baseline neutrino experiment to be successful, the neutrino beam needs to be understood very well. This is achieved with a near detector (ND), which for DUNE will be located 574 m downstream from the end of the decay pipe. The DUNE ND will consist of a suite of three different detectors, each utilising a different detector technology. This suite of detectors will be arranged linearly downstream of the neutrino beam, as shown in Figure 3.12.

The most upstream detector in the ND complex is the Liquid Argon Near Detector (ND-LAr), which compromises a number of LArTPC modules, rather like the FD, but with a modified design that allows the detector to operate efficiently in the high-intensity environment close to the source of the neutrino beam. The chosen technology for ND-LAr, called ArgonCube [46], consists of 35 optically separated LArTPC modules, each allowing for independent identification of neutrino interactions on argon. Each TPC will have its own FC, cathode, PDS and a pixel-based charge readout. This allows ND-LAr to cope with the large number of neutrino interactions, which is expected to be $\mathcal{O}(50)$ ν interactions per spill. It is very important that both the near and far detector share the same detector technology, in this case the LArTPC, such that there can be a cancellation of some detector systematics when utilising both the near and far detector data together in an oscillation measurement.

As ND-LAr uses both the same target (LAr) and the same detector technology, this will be key in reducing systematic effects in the oscillation analysis, such

as uncertainties in the neutrino flux, neutrino interaction physics and detector properties. The size of ND-LAr has been chosen to ensure the detector will be able to provide high statistics with good hadron containment, though muons with momenta greater than 0.7 GeV/c will not be contained in the LArTPC volume.

Located directly downstream of ND-LAr is the Gaseous Argon Near Detector (ND-GAr). The principal component in the ND-GAr detector is a high-pressure gaseous TPC (HPgTPC) that is surrounded by an electromagnetic calorimeter (ECAL) all enclosed in a 0.5 T magnetic field. The primary benefit of ND-GAr is that its lower density and high-pressure system provide benefits to both tracking resolution and lowering the momentum acceptances, as well as helping to identify particles that are produced in the interactions of ND-LAr, by measuring the momentum and sign of charged particles that exit the ND-LAr detector volume.

To improve the constraints on the neutrino energy spectrum and flavour composition, both ND-LAr and the ND-GAr will be able to move off-axis, through a system called DUNE Precision Reaction-Independent Spectrum Measurement (DUNE-PRISM). This allows them both to take off-axis measurements, allowing the ND to create a neutrino energy distribution that more closely matches that of the FD, by combining different flux measurements taken at different degrees off-axis. Figure 3.13 shows how the predicted DUNE muon neutrino flux changes at the ND as a function of off-axis angle.

The final detector will be the System for on-Axis Neutrino Detection (SAND). SAND will function as an on-axis, magnetised beam monitor at a fixed position. This fixed on-axis position should allow it to be more sensitive to variations in the neutrino beam. SAND will consist of an inner tracker surrounded by an ECAL inside a large solenoid magnet, with two possible tracker designs currently being considered. The first option consists of plastic scintillator cubes with TPCs, whilst the second option is based on straw-tubes. SAND is most important as a dedicated neutrino spectrum monitor, staying in an on-axis position whilst ND-LAr and ND-GAr move off-axis. The different mass numbers of the hydrocarbon target at SAND, relative to the argon at the other two ND modules means that SAND may also prove useful for developing models of nuclear effects, building confidence in the interaction models used and the sizes of the numerous systematic uncertainties, whilst also offering an interesting point of comparison to other experiments such as Hyper-K and MINERvA [47], which could also prove useful for understanding systematic effects.

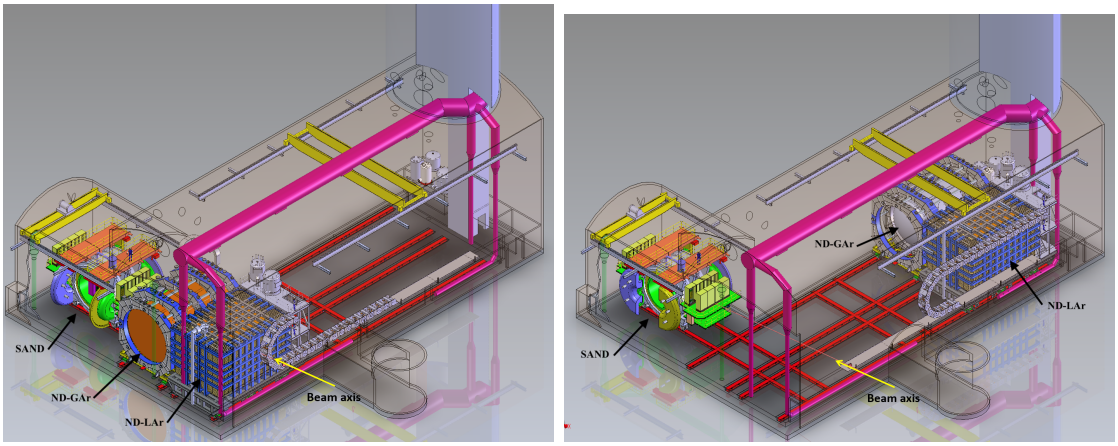


Figure 3.12: Schematic of the DUNE ND hall, shown in both the on-axis and off-axis configurations (left and right, respectively). The beam location is shown in each figure as a yellow arrow. Figure is taken from [48].

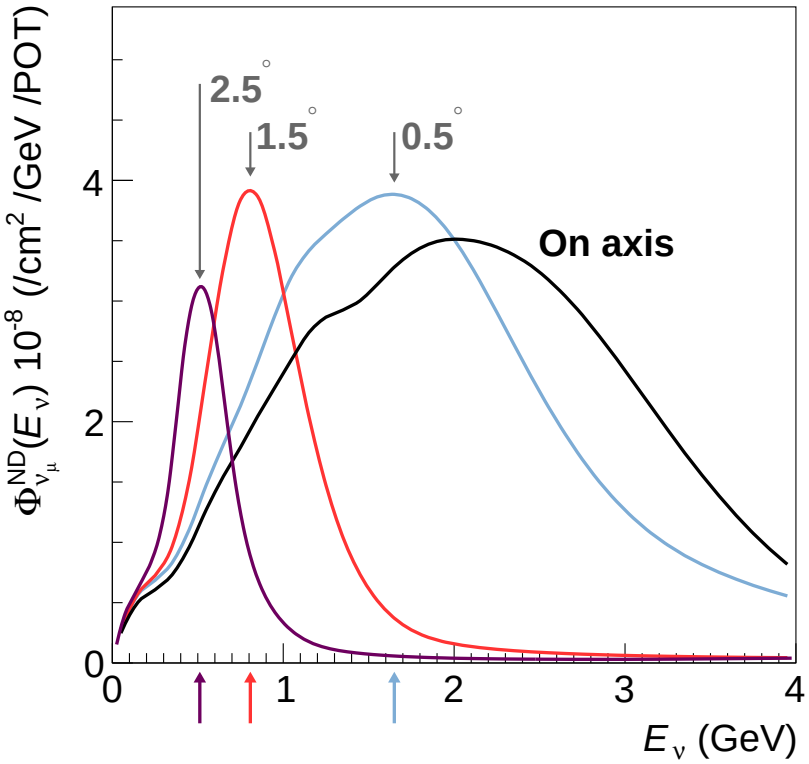


Figure 3.13: The predicted DUNE muon neutrino flux at the ND, as a function of off-axis angle. The arrows in the figure indicate the peak neutrino energy for three different off-axis angles, due to DUNE-PRISM. Figure is from [48].

The primary driver for the design of the DUNE ND complex is the long-baseline oscillation physics program. Figure 3.14 shows the impact of ND detectors on the measurement of CP violation at DUNE. However, this does not mean that the DUNE ND is only capable of producing results for this physics program, as it can be a powerful tool for investigating both Standard Model (SM) and Beyond Standard Model (BSM) physics topics. For example, the DUNE ND has an extensive cross-section physics program, as well as the opportunity to search for light dark matter, a search for heavy neutral leptons, enhanced background searched for proton decay and much more. Most of the work around this is at an early stage, and is subject to change as the experimental and theoretical landscape changes before the ND is fully operational, but DUNE will look to take advantage of any competitive and novel measurements in this area.

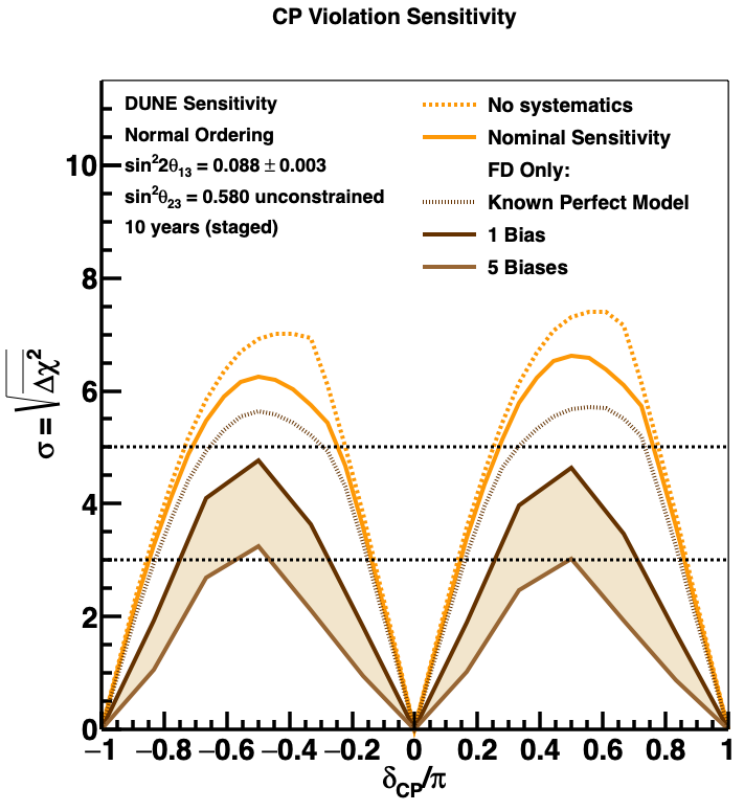


Figure 3.14: The effect on the sensitivity to CPV at DUNE if an incorrect cross-section model is used in the reconstruction. This danger can be alleviated by improving or tuning the model using data from the ND. Figure taken from [48].

3.5 ProtoDUNE

As DUNE will be the first large-scale neutrino experiment to exploit LArTPC technology, an extensive programme of research and development is underway to demonstrate and up-scale the systems that will be deployed in the ND and FD modules. As part of this research and development programme, a pair of large-scale prototypes has been constructed and operated at CERN: ProtoDUNE Single-Phase (ProtoDUNE-SP) and ProtoDUNE Dual-Phase (ProtoDUNE-DP). Both of these detectors live in the EHN1 building at CERN's Prevessin site, and both exist to test and prototype all aspects of the technology that will eventually be deployed to the DUNE FD. Figure 3.15 shows a photo of both ProtoDUNE-SP and ProtoDUNE-DP.

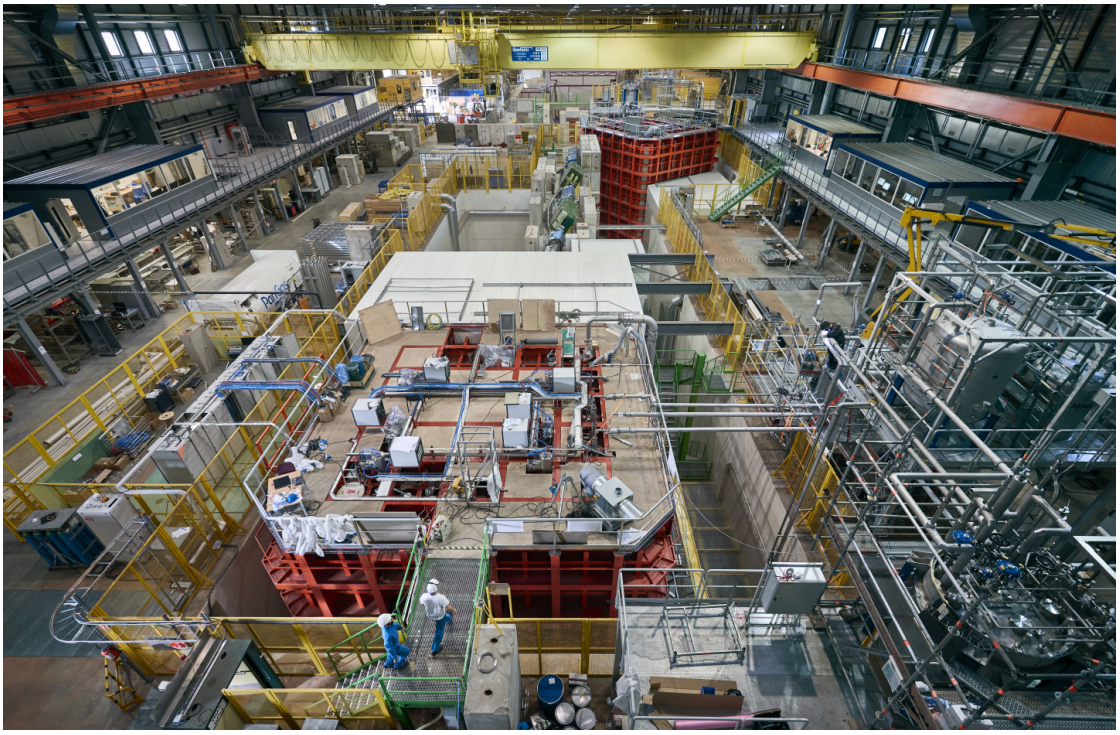


Figure 3.15: An image of the EHN1 building at CERN, which houses the two ProtoDUNEs. In the foreground, ProtoDUNE-SP can be seen, with the secondary red cryostat in the background housing ProtoDUNE-DP. Between the two is the H4 beamline. Figure from [49].

Full-sized components are used following current FD design specifications, but in a detector that represents only a small portion of a full FD module. ProtoDUNE-SP has only two drift volumes, with one cathode wall in the middle of

the detector, with the anode walls only being one APA tall (rather than two) and six APAs deep (rather than twenty-five). This allows the full-sized components to be tested in a setup that is smaller than a full size FD module, as well as meaning the manufacturing process for many components can be tested, to aid understanding for later mass fabrication. ProtoDUNE-DP represents a similar portion of a full-size FD VD module.

Both ProtoDUNE detectors were built between October 2015 and July 2018. ProtoDUNE-SP was completed, filled with LAr and commissioned in August 2018, allowing it to receive test beam data between September and November in 2018. ProtoDUNE-DP completed slightly after ProtoDUNE-SP, meaning it missed the CERN test beam before the planned long shutdown, but was able to take cosmic ray interaction data along with ProtoDUNE-SP. An example test beam interaction is shown in Figure 3.16 for ProtoDUNE-SP, showing a 6 GeV/c electron candidate event.

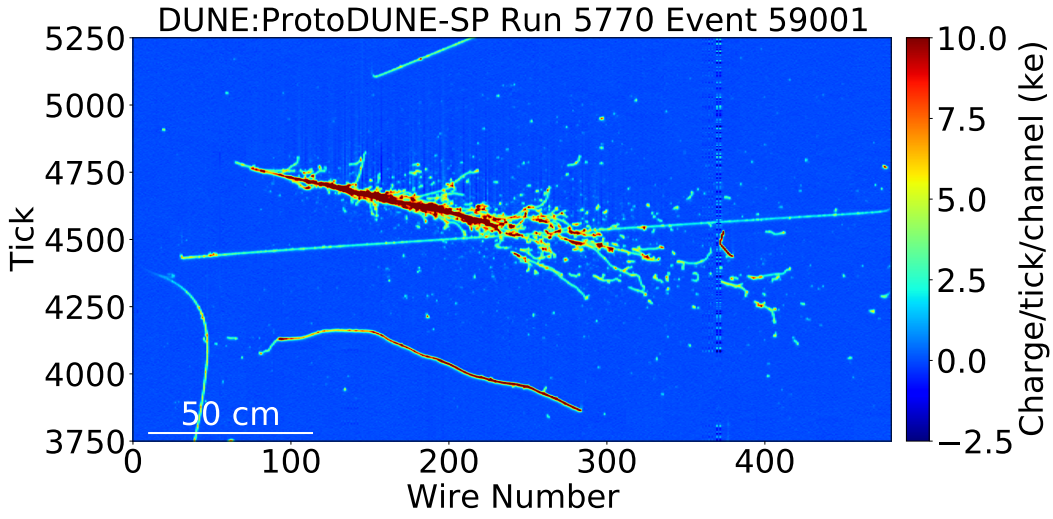


Figure 3.16: An example ProtoDUNE-SP event display, showing a 6 GeV/c electron candidate. The colour scale is used to show deposited charge in the LAr, and other charged particles can be seen around the central electron candidate. Figure is from [50].

3.5.1 ProtoDUNE-SP

As ProtoDUNE-SP has test beam data it is a useful test bed for a variety of DUNE FD work, allowing software to be tested on a large, multi drift-volume

detector, to see how data and simulation results compare. This includes the ability to both develop and demonstrate the performance of algorithms for simulation, calibration, and reconstruction; the measurement and subsequent understanding of detector properties; and physics studies including pion absorption analyses. The total amount of data taken for test beam events recorded is outlined in Table 3.5, split up by momentum.

Momentum (GeV/c)	**Total Recorded Beam Triggers**
0.3	269 000
0.5	340 000
1.0	1 089 000
2.0	728 000
3.0	568 000
6.0	702 000
7.0	477 000
Total	4 173 000

Table 3.5: The number of beam trigger events at ProtoDUNE-SP, split by momentum, provided by the CERN test beam. Table taken from [51].

Using this test beam data, ProtoDUNE-SP can provide useful insight into how components and software will perform in the full scale FD, as well as being able to contribute its own physics results. The first performed analyses at ProtoDUNE-SP focus on understanding both the TPC and PDS in more detail, and how they perform in a large scaled LArTPC. These initial studies are outlined below, taken from [50]:

- **Understanding the space charge effect (SCE)** in ProtoDUNE-SP. As ProtoDUNE-SP is a surface-based detector, it experiences a large flux of cosmic ray interactions, which in turn ionise the LAr, producing both ionisation electrons and argon ions. The argon ions have drift velocities slower than that of the ionisation electrons (around $2 - 4 \times 10^5$ times slower), which results in the slow build-up of a considerable amount of positive charge with a gradient that opposes the applied electric field, as these ions build up over the scale of ten minutes or so. This positive charge density is also known as space charge. This effect is also impacted by the flow of the

argon, and asymmetries in the flow patterns can cause asymmetric space charge in the detector, necessitating a data-driven study to understand the effect. Understanding this effect is key, as it must be incorporated in both the simulation and reconstruction, due to its impact on dE/dx[2], energies and particle trajectories. Sophisticated space charge maps have been produced, using reconstructed cosmic rays, allowing its effect to be better understood, and have it be included more accurately in the simulation, as well as producing "inverted spatial distortion maps" that can be used to calibrate out spatial distortions in reconstructed or MC data.

- **The lifetime of drift electrons**, that is, the rate of attenuation of electron in the liquid argon, in ProtoDUNE-SP is very important and directly relates to how pure the LAr is kept, with impurities like oxygen or water in the detector capturing ionisation electrons as they drift, reducing the signal measured. This then means the charge measured is lowered, biasing the amount of charge measured. Purity monitors and fits to cosmic ray dQ/dx are used to measure the drift electron lifetime, to check that the strict limits in lifetime are met. Measuring electron lifetime also allows it to be more accurately reflected in the simulation. Electron lifetimes of around 10.4 ± 1.5 ms were measured at the start of data taking, rising to closer to 100 ms near the end of data taking, as the purity of the LAr increased.

- **Energy reconstruction** was performed for beam muons and pions, beam protons and beam electrons. This is useful both to understand how the detector performs, but also provides useful data for the DUNE FD, increasing datasets for interactions on argon. Reconstructing final state pions and their produced secondary particles is an important part of energy reconstruction for neutrinos, and beam pion data can be used to verify this. Particle identification and calorimetric energy reconstruction has been performed for each beam particle, allowing the simulation to be verified and reconstruction algorithms to be tuned. A fit performed on reconstructed beam pions and stopping muons dE/dx distributions at ProtoDUNE-SP compared to MC agreed to better than 1%.

- **A robust particle identification (PID)** is critical to the DUNE FD and

[2] dE/dx (and dQ/dx) are used to refer to energy and charge deposited per unit track length. The dx refers not to the detector coordinate x, rather a differential step along the track path.

ProtoDUNE-SP. A calorimetric-based PID method for LArTPCs has been tested at ProtoDUNE-SP. It uses the reconstructed energy deposits as a function of residual range for any stopping particles. This allows for the identification of minimally ionising particles (MIP) (muons) against non-MIP particles (protons), which can be difficult otherwise. This is required to meet the physics goals of DUNE, so verification at ProtoDUNE-SP is key. Early results at ProtoDUNE-SP show that when the dE/dx of muons and protons are plotted against their residual range, a high level of separation is observed.

Further studies have also been performed, as well as upcoming physics analyses that focus on neutrino cross-section measurements on argon. Each analysis also allows the full software chain to be checked, verifying how the software chain runs on data, compared to simulation. This includes verifying the energy reconstruction as mentioned, as well as the steps that are needed to reach that point, such as the waveform deconvolution, hit construction and the pattern reconstruction. The software chain itself is described in detail in Chapter 4.

Whilst ProtoDUNE-DP does not have specific test beam data, software efforts to properly simulate and reconstruct CRP data is useful, as well as informing design decisions for the DUNE VD module. The ProtoDUNE detectors are both providing useful insight into the largest scale LArTPCs yet, allowing both hardware and software to be tested, which is key in making a full scale, 10 kt DUNE FD a reality. They prove that a LArTPC can scale up from the pioneering T600 detector built by the ICARUS collaboration, to 1 kt of LAr and a 3.6 m drift distance, whilst maintaining a high S/N ratio and the ability to accurately reconstruct charged particles from interactions in the LAr.

4

Software for LArTPCs

"Estás usando este software de traducción de forma incorrecta. Por favor, consulta el manual."

Wheatley - Portal 2

Software is a key part of any modern physics experiment. Experiments produce so much data (DUNE is expected to have a data rate to tape of up to 30 PB/year), that automated processing is a requirement. Fast, efficient software for taking the fine-grain images of a LArTPC and producing outputs that can be used for physics, such as clusters of energy deposits, particle identification and their energy and more, is needed to harness the full power of DUNE. Equally, sophisticated simulation is required, to ensure that the detector is fully understood, to develop and benchmark reconstruction software whilst the detector is being built, and optimise the physics capabilities of the detector, using the current understanding of how the detector should perform and the interactions that will take place. Finally, data like that from ProtoDUNE-SP and ProtoDUNE-DP mean that the simulation and reconstruction chain can be tested, to ensure it runs as expected on real data, benchmarking the current software approach, and the underlying physics of the simulations.

This chapter will outline the key software components used by DUNE, and how each part interacts, starting from the simulation of neutrino interactions on argon nuclei in the DUNE FD module, and ending with their reconstruction and the subsequent physics objects that are used for analysis. The first section will outline the overall software framework used in DUNE. Following this section, a

description of the neutrino interaction simulation is given for each of the individual steps that are needed to simulate a neutrino interaction on argon nuclei inside a DUNE FD module. After simulation, there is a section outlining the various steps of reconstruction that are applied to data to produce physics outputs. Finally, the last section outlines Pandora, the reconstruction framework that was used and further developed as part of this work.

art and LArSoft

The DUNE software is built on the common *art* [52], and Liquid Argon Software (LArSoft) [53, 54] framework that is used across the Liquid Argon neutrino program.

art is an event-processing framework that is used to process particle physics data in an event-by-event loop. This includes standards for the organisation of data (into runs, sub-runs and events), as well as defining the format of output data, containing so-called *data products*, which are experiment (or user) defined classes. *art* was built to avoid previous issues where experiments would produce their own framework, making the sharing of code between experiments much harder, as well as duplicating work which could otherwise be shared.

Built on top of *art*, there is LArSoft, a shared suite of reconstruction and simulation software, used across multiple LArTPC neutrino experiments. It provides many of the standard C++ classes that a neutrino experiment will need to encapsulate the various simulation, reconstruction, and analysis workflows that are common in LArTPC experiments.

The existence of a shared software framework creates an environment for cross-experiment collaboration on software. It is common for software enhancements that are achieved at ProtoDUNE-SP (utilising beam data) to feed back into the software for the DUNE FD modules, as well as SBND and more. This also means that many of the collaborations and software groups that work on software for LArTPC experiments are associated with multiple experiments, rather than just one. Some are general enough that they are used ubiquitously across most of particle physics, such as GEANT4 [55], whilst others like Wire-Cell [56, 57] are used across multiple LArTPC experiments such as MicroBooNE, SBND and DUNE. It is common for there to be interfaces between LArSoft and other software packages, allowing more broad packages to be used, making code reuse even easier. Overall, this means that the entirety of the DUNE FD simulation and reconstruction chain

can run in a single framework, with a consistent interface, making the creation and subsequent analysis of events much easier and consistent.

All the sections that follow are embedded into the *art* / LArSoft framework, except where explicitly stated otherwise. Many of these packages are also available in a standalone capacity, but through the use of a LArSoft interface, they can be embedded into LArSoft, enabling a single framework to simulate and reconstruct neutrino interactions on argon in a DUNE FD module. This is very useful for onboarding new users, as understanding LArSoft enables them to produce neutrino events, rather than requiring them to learn multiple tools.

4.1 Event Simulation

Event simulation covers two broad parts, the generation, and interaction of particles, and then simulation of how the detector hardware will respond to those interactions. These steps are usually split into three parts, with the first being the initial generation of the interaction, which produces the neutrinos that interact at a given energy (based on simulations of the beam) with argon and the resulting final-state particles from that interaction. Subsequently, with the particles and their energies known, as well as an interaction position, there is a simulation of the particle transport, through the liquid argon of the detector, as well as simulation of the secondary interactions and any particle decays. Finally, there is a detector simulation steps, which simulates the detector response to the particles travelling through the matter of the detector, to produce realistic outputs, adding in detector noise and other physical effects, as well as simulation of the collection, amplification, and digitisation of the signal.

A key part of event generation is the use of Monte Carlo (MC) generators, which use pseudo random numbers to simulate the expected interaction kinematic distributions and the final state particles, by sampling from the probability distributions associated with the cross-section, particle transport and detector response models. This technique plays a part in each stage of the simulation process, allowing the estimation of the various physical properties that occur throughout the simulation chain.

4.1.1 Event Generation

Before any neutrino interaction can be simulated, a simulation of the incident neutrino flux is required. At DUNE, this is achieved with G4LBNF, a GEANT4-based simulation of the LBNF neutrino beam, utilising a detailed description of the LBNF optimised beam design [58]. This provides a prediction of the number of $\nu_\mu, \nu_e, \bar{\nu}_\mu, \bar{\nu}_e$ per proton on the target per unit area, that will come from the beam as a function of energy. The neutrinos from the beam flux are consumed by GENIE [59], which is the primary event generator used by DUNE. GENIE is used across many neutrino experiments, LArTPC and otherwise, and uses data from existing experiments to tune its physics models. It can provide modelling of neutrino-nucleus interactions, but also simulation of many non-neutrino processes useful for non-beam data simulation, such as nucleon decay, boosted dark matter interactions, and more. There is also additional tooling built into GENIE to facilitate the propagation of modelling uncertainties, interfacing with detector geometries. In addition to GENIE, DUNE employs a number of other specialised event generators. For example, the CORSIKA [60] generator is used for the flux of cosmic-ray particles, and a dedicated event generator is used for the simulation of supernova neutrino bursts [61, 62]. The NuWro generator [63] is also used throughout DUNE, mostly to offer a point of comparison against GENIE. For example, a mock study into interaction model deficiencies at the DUNE near detector used NuWro as a fake data sample with GENIE as the simulation, to test the sensitivity of a near detector module to model differences [48].

Regardless of the chosen generator, the output of this stage is a list of final-state particles from the initial neutrino interaction, each described by a particle type and four-momentum, that provides the input to the next stage of the simulation. Further decays of these particles are left to the next stage.

4.1.2 Particle Transport and Detector Simulation

In this phase of the simulation chain, the final-state particles are propagated through the detector, and interact and decay according to a set of detailed physics models. The simulation must describe all features of the detector geometry, such as the LAr, the cryostat, or even the experimental hall that contains the detector module. At DUNE, the transport of particles is simulated using GEANT4 [55], a general purpose toolkit for the simulation of particles through materials. GEANT4

is responsible for tracking each particle through the detector and simulating the interactions of decay products with the detector volumes. As particles are transported through the detector geometry, energy deposits are simulated and then converted into ionisation electrons and scintillation photons. The ionisation electrons drift, due to the applied electric field, towards the anode plane assemblies. As these electrons drift, they undergo diffusion and recombination effects. The associated ions move in the opposite direction and generation distortions in the electric field, which must also be simulated. Additional simulation is needed of the effects that can reduce the number of ionisation electrons that reach the wire planes, such as recombination with positive ions in the LAr or impurities in the LAr.

The final stage of the simulation models the detector readout and response. A detector simulation step is needed to simulate the waveforms that are produced when simulated ionisation electrons and scintillation photons reach the active detector components and induce a signal. The drift electrons induce a signal on the induction wires in the APA, before being collected onto the collection plane wires. These signals are amplified and shaped by the detector electronics, with the resulting waveforms being digitised. At DUNE this is achieved with the Wire-Cell toolkit software package [56, 57]. This simulation takes the GEANT4-produced energy depositions from the particle traversing the detector, and outputs the digitised waveforms that the front-end electronics would produce.

The waveforms should accurately reflect realistic noise rates and electronics characteristics, giving a useful input to the reconstruction stage. Once built, the simulation can be tuned and modelled using real data, leading to better understanding of the detector and data. The tuning of the simulation could reflect a better understanding of the interaction models of physical processes that form the basis of the underlying particle transport or detector response, or more mundane things such as a more realistic simulation of noise using real hardware.

4.2 Event Reconstruction

Event reconstruction, the translation of raw 2D LArTPC outputs into 3D physics objects for analysis, is a complex, but necessary task. What may be an easy problem for a human to solve, such as the identification of an interaction vertex, can be a challenging problem to encode into software. However, physics analyses

are performed at a much higher level than raw data, requiring selections of particle types and energies, interaction locations, counts of the number of particles in an event and much more, rather than raw data from ionisation electrons and scintillation photons. This means sophisticated reconstruction is required, to allow analysis to be performed without being obstructed with needing to parse the raw data first.

Reconstruction can be split into three main stages. There is the initial, early, reconstruction that reduces and cleans up the images. At this stage, waveforms are processed to filter out noise and deconvolved to correct for detector response. The resulting waveforms are formed into discrete peaks called hits. The reconstructed hits are fed into pattern recognition algorithms, that fit into more traditional computer vision style problems, such as clustering hits to make a single group that represents a particle or finding features in the event like interaction vertices. Finally, high-level reconstruction determines the specific properties of particles, such as the type of particle or the energy. Particles are also formed into hierarchies, mapping the cascades of interactions or decays.

4.2.1 Raw data processing

Reconstruction starts with the loading and processing of raw data. LArTPC raw data is in the form of analog-to-digital converter (ADC) and time-to-digital converter (TDC) counts, with waveforms encoding ADC vs TDC, with a full output image consisting of many of these waveforms in parallel. The first step is to reconstruct the charge distribution detected on each readout wire[1]. This is achieved through the use of deconvolution algorithms, alongside noise filters to reduce noise.

Noise filtering is used to reduce the various sources of electrical and thermal noise that can occur in the electronics of a LArTPC. Noise sources are usually due to interactions between various parts of the hardware. For example, one source of "noise" is the appearance of "sticky codes" in real data, where certain ADC channels will prefer to stick to certain values, independent of the input voltage. This can be dealt with in some cases by filtering out the known stuck channels and replacing the value with one approximated from the neighbouring channels. Another source of noise is coherent noise, a form of noise which occurs across

[1]Wire is used interchangeably with the words strip or pixel, to refer to the readout channel in use, regardless of the actual detector technology.

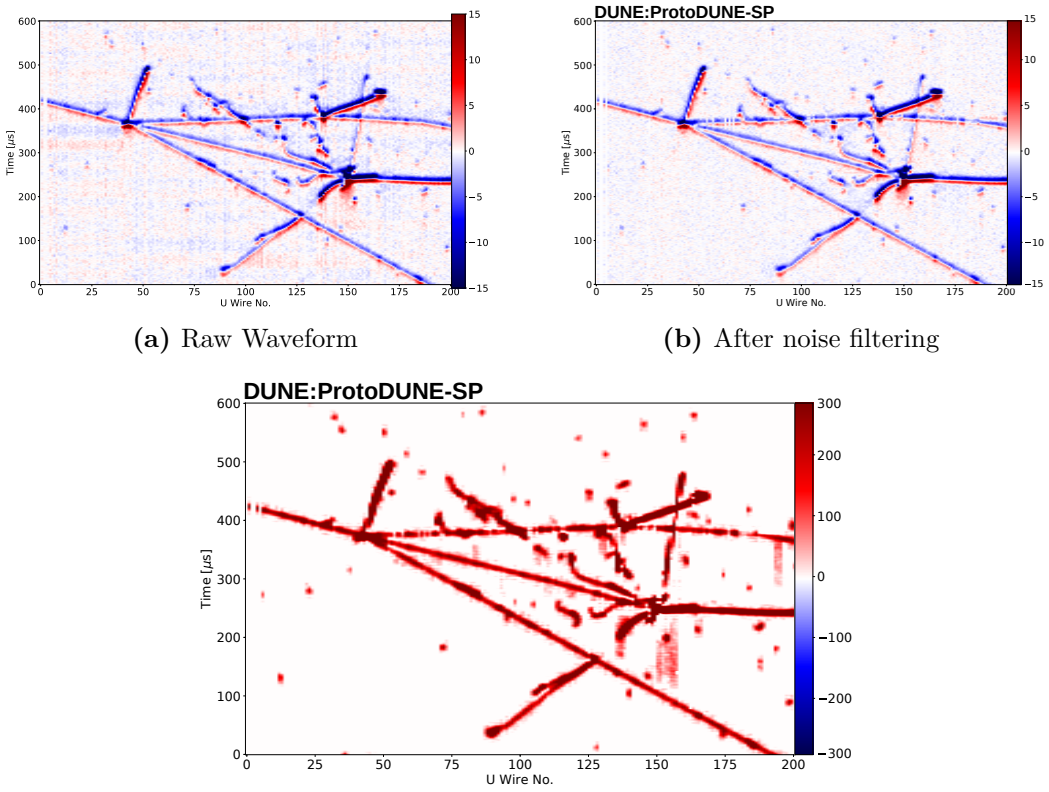
multiple connected channels across the detector. For example, ProtoDUNE-SP observed a source of noise with a peak around 45 kHz [50], which was found to be shared across channels that utilised the same low-voltage regulator in their front-end motherboard. The impact of this noise can be reduced by constructing a correction waveform, based on the calculated median over the groups of wires, and subtracting it from the initial waveforms, taking care not to inadvertently reduce the signal too by avoiding areas that include substantial signal from ionisation electrons.

After the noise-filtering algorithms have been applied to the data, a deconvolution procedure is used to filter out residual high-frequency noise, and to unfold the signal from the detector response. Deconvolution was first introduced to LArTPC signal processing, with analyses performed at ArgoNeuT [64]. Deconvolution aims to extract the true signal from the measured signal, by removing the impact of field and electronic responses from the measured signal, such that the number of ionisation electrons can be reconstructed. This technique is fast and robust, and an essential part in the reconstruction of LArTPC data. Whilst initially implemented in 1D (TDC) by ArgoNeuT, the MicroBooNE collaboration improved to a 2D process (wire number vs TDC), considering the long-range induction effects in the spatial dimension, such that both time and the wires are considered [65, 66]. At DUNE this is achieved using Wire-Cell. The combination of both noise removal and deconvolution can be seen in Figure 4.1, applied to a 7 GeV particle event at ProtoDUNE-SP.

Following noise removal and deconvolution, there are hit finding algorithms that allow follow-up algorithms to stop using raw waveforms and instead use discrete hits, with a hit representing a charge deposition on a certain wire at a given time. This is achieved through the use of algorithms that scan the input deconvolved waveforms and fits a Gaussian shape to the peaks, producing hits. In situations where a peak cannot be fitted with a simple Gaussian shape, for example where a particle trajectory is close to the current plane, the peak can bypass the hit-fitting, and instead be divided into a number of evenly spaced hits. An example of this hit finding at ProtoDUNE-SP can be seen in Figure 4.2.

4.2.2 Pattern Recognition

After the low-level reconstruction is performed, each event has had its raw waveforms cleaned and converted to hits. The second stage of the reconstruction



(a) Raw Waveform

(b) After noise filtering

(c) After Deconvolution

Figure 4.1: Example of noise filtering and 2D deconvolution applied to a 7 GeV particle event at ProtoDUNE-SP. Figure **a** shows the original raw waveforms, which becomes Figure **b** after noise filtering is applied; Figure **c** shows the ionisation charge in number of electrons (scaled by 200), which has been extracted by the 2D deconvolution process. The blue and red components in **a** and **b** refer to the positive and negative components of the bipolar signal, respectively. This signal becomes unipolar after the deconvolution step is performed. Figures **b** and **c** are taken from [50], with **a** being an internal plot from DUNE for the same event.

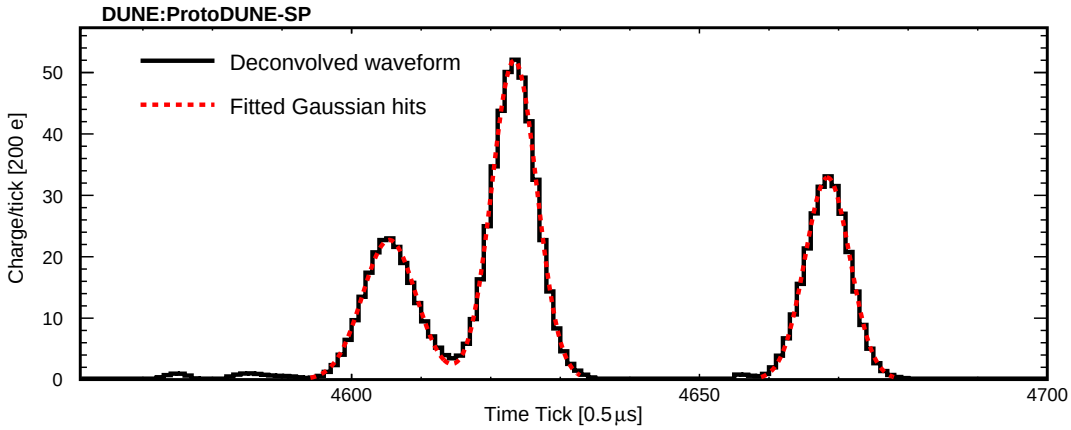


Figure 4.2: An example of a reconstructed waveform on a single wire, taken from ProtoDUNE-SP data. The dashed red lines show the fitted Gaussian shapes, highlighting the three reconstructed hits. Figure is from [50].

is pattern recognition, which clusters together the individual hits that relate back to the same final-state particles. This problem can be approached many different ways, but there are some key tasks that are common across all reconstruction paradigms:

- **Clustering**: The process of grouping together hits that belong to the same initial particle. For example, if a muon travels through the detector leaving a trail of ionisation electrons, clustering would group together the hits, tagging them as all coming from one particle. Clustering must be careful to not also include hits from other particles, which would bias energy calculations later, and makes other reconstruction tasks harder.

- **Vertexing**: Vertexing refers to the identification of the start location of interactions or decays in the event. This stage of the pattern recognition is important, as the knowledge of this key feature in the event helps to drive the clustering of hits and the reconstruction of particles. Vertexing is usually split into several steps, starting with the initial neutrino interactions, and then moving on to secondary interactions or decays.

- **3D Reconstruction**: As mentioned briefly in Section 3.3.1, it is necessary to reconstruct events in 3D by combining the multiple 2D images produced in modern LArTPCs. This involves matching the hits across multiple views, and then combining them to determine the full 3D coordinate. A range of

approaches have been developed for reconstructing 3D events from the 2D images. A deeper explanation of this is given in Chapter 6.

- **Interaction hierarchies**: Building on the clustering and interaction vertices, a full interaction hierarchy can be built. This allows the chain of interactions to be followed, from the neutrino, to its initial daughter particles and then their daughter particles and so on. This is useful for later analysis exercises that wish to target specific interactions.

These results are all achieved using the input hits from the previous hit finding step. The actual specific programmatic approach taken by each reconstruction framework can differ, though, as well as the order of the above steps. For example, a vertexing step could be taken immediately, as the result of knowing where an interaction took place impacts the process of clustering, as the nearby particles will start from that vertex. However, it is also a valid approach to this problem to look for the vertex after the clustering step, and look for a common interaction location based on properties of the identified clusters, such as calculating a direction vector from the cluster hits. This difference in approach is even more drastic when looking at more modern approaches to reconstruction using deep learning. Deep learning techniques and how they function will be explained in further detail in Chapter 5, although some explanation of how it can be used for reconstruction will be outlined here. A deep learning-based approach to vertexing may not use clustering at all, instead being based on images or graphs generated from the input hits, and using relationships between the hits and the properties associated with them, as well as topographical features.

There are three main reconstruction paradigms to talk about in the context of LArTPC reconstruction, with two of them in use at DUNE.

- **Pandora**: The first is Pandora, which will be covered in more detail in Section 4.3, as it is the reconstruction framework that the work of this thesis was developed inside. Pandora approaches the reconstruction problem initially from a 2D perspective, using the 2D readouts from each wire plane to reconstruct events, before moving to a full 3D reconstruction once there is sufficient reconstructed information to perform the step. This 3D information is then used to verify the initial 2D reconstruction by heavily utilising coordinate mappings to project the 3D information back into 2D.

Pandora is used for both FD modules, ProtoDUNE and has in-progress work to apply it to the DUNE ND.

- **Wire-Cell**: Wire-Cell is another reconstruction framework that differs to the Pandora approach in that it attempts to reconstruct the 3D coordinates of the event up front, before performing pattern recognition steps, under the assumption that performing pattern recognition in 3D gives additional separation power over purely 2D reconstruction. Wire-Cell is currently not used for the pattern recognition step at DUNE, though it is used for the signal processing in the previous step, as well filtering and deconvolving the raw waveforms. It has, however, been used successfully for full reconstruction at MicroBooNE [67].

- **MLReco**: The final reconstruction paradigm is one based on deep learning [68]. MLReco differs most drastically from the other two reconstruction frameworks in that it uses the raw hits for many more tasks, such as segmenting the hits of the event, clustering, and vertexing all with just the hits, rather than building up an image of the event sequentially building on the previous step. The broad tasks, however, fit with the list given previously, with targeted networks used for specific tasks such as clustering or vertexing, with outputs from multiple networks feeding into follow-up networks that build up higher level information about the event, such as interaction hierarchies. MLReco is specifically being used at the DUNE ND at the moment.

One thing to note is that both Pandora and Wire-Cell utilise deep learning techniques as part of their reconstruction chain, but they are used alongside other non-deep learning-based algorithms, whereas the deep learning framework does the majority of its reconstruction using deep learning.

4.2.3 High-level reconstruction

Finally, there is high-level reconstruction. Whilst pattern recognition identifies the important features of an event, such as clusters, vertices, charged-particle trajectories, the high-level reconstruction involves the determination of global event properties for physics analyses. The key high-level reconstruction tasks include particle identification and energy reconstruction. However, as mentioned

previously, the line between each stage can be blurred depending on the approach taken, with the MLReco paradigm in particular blurring the line between pattern recognition and high-level reconstruction.

The high-level reconstruction for DUNE is under active development, but some examples of the present algorithms are descried in Section 3.5.1 in the context of early studies with ProtoDUNE-SP data. Figure 4.3 shows the result of dE/dx versus residual range for stopping protons and muons, as an example of how a PID could operate to distinguish between protons and muons, a task that is not usually performed at the pattern recognition stage. A similar study has been performed at ProtoDUNE-SP utilising deep learning to reconstruct the energy of particles. Studies are also performed on simulated data of the full DUNE detectors, rather than the real ProtoDUNE-SP data, to get software tooling in place, as well as give more realistic bounds on the physics potential of the experiment using actual software, as compared to earlier estimates that may make assumptions about how well events can be reconstructed or similar. The convolutional visual network (CVN) is an example of this that runs on the DUNE FD MC to tag the neutrino interaction flavour [69], allowing for a more accurate understanding of the physics potential of DUNE. This software can be tested on both the DUNE FD MC, as well as real data from ProtoDUNE-SP. The process of comparing simulated data to real data is an important exercise, as it allows limitations of the current simulation to be explored, as well as highlighting detector effects that are currently not well understood or known about.

The outputs of the high-level reconstruction are event-level objects, that allow physics analyses to be performed at a high level, without the need to consider low-level details in the data. For example, a common first step in a physics analysis is a cut on the neutrino interaction vertex, to ensure interactions are fully contained within the detector. Similarly, it is common to have requirements on the particles in an interaction or the event type. Additionally, a combination of the pattern recognition stage and PID can be used to target specific interaction hierarchies more generally, such as “one track and one shower”, then, getting more specific, for example that the track should be a muon, not a proton, or the interaction should have particles that have an energy that is more than a given amount. This allows analyses to be started much quicker, as well as meaning that every analysis benefits from improvements made to the reconstruction chain, thanks to the vast reuse of reconstruction code, both across analysis groups, detectors, and

experiments.

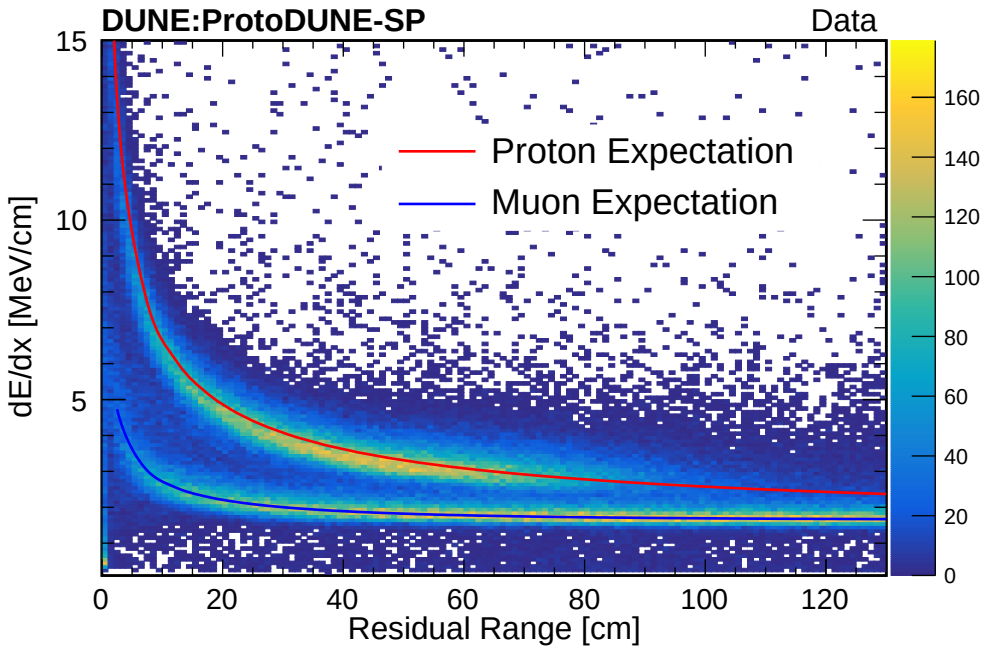


Figure 4.3: The dE/dx versus residual range for both stopping protons (the upper band) and muons (the lower band) at ProtoDUNE-SP. This is after space charge corrections have been applied. Figure is taken from [50].

4.3 Pandora

Pandora is a software development kit [70], designed to provide a solution to the complex pattern recognition problems that occur in particle physics, by utilising a multi-algorithm approach, that slowly builds up a full representation of the input event. This approach allows the complex problem of reconstructing a full event to be approached iteratively, with problems being broken down until they are more easily achieved with small targeted algorithms, as well as allowing tasks that benefit from a more complete event model to wait until the event has been sufficiently reconstructed. Initially, Pandora was developed in the context of the International Linear Collider (ILC), before moving to be focused on the reconstruction of the fine grain images produced in a LArTPC, whilst also showcasing how the framework is flexible enough to swap detector technologies. Pandora is currently in use at MicroBooNE [71], ProtoDUNE-SP [72] and ProtoDUNE-DP, ICARUS, SBND, the ILC [73], CLIC [74], and the DUNE

FD and DUNE ND. This covers a vast range of detection technologies and detector environments. Within the LArTPC neutrino programme alone, Pandora is able to reconstruct events for surface-based LArTPCs with many cosmic rays, or with multiple drift volumes, differing APA layouts, test beam interactions rather than neutrino interactions and much more, by utilising its multi-algorithm approach.

4.3.1 Multi-algorithm approach

The multi-algorithm approach to pattern recognition that Pandora employs is designed such that each of the over 100 algorithms tackles one small problem, by taking a larger problem and breaking it down into simpler and simpler steps, before finally the problems are small and well-defined enough that it is simple to complete without introducing mistakes, which in turn means that follow-up algorithms do not need to correct mistakes from the early stages. One high-level concept such as clustering may be split into many algorithms, with each algorithm in turn being powered by many tools to target the specific issue that the algorithm was designed to address. The algorithms can then be chained together, building up the full pattern recognition pipeline. A secondary advantage of this approach is the ease in which the chain can be altered or adapted, by inserting new algorithms between the existing, and allowing parts to be removed or added if a specific experiment requires a different approach to pattern recognition. Similarly, algorithm settings can be defined in the algorithm chain configuration, allowing the tools and algorithms to be optimised per experiment or use case. A simple example of this may be to tune an algorithm to deal with the varying sizes of detector that it may run on, such that it can work for both the full scale of a DUNE FD module, and a smaller detector like MicroBooNE.

Another benefit of the multi-algorithm approach is the ease with which algorithm chains can be modified to handle different event types and topologies. There are three main chains in use at DUNE and its prototypes: Pandora Neutrino, Pandora Cosmic and Pandora Test Beam. These each refer to specific chains of algorithms that are optimised for different event topologies. The differences between these chains varies algorithm to algorithm, with some parts being identical, but in others there will be specific targeted algorithms or assumptions that are changed. For example, the beam neutrino and test beam chains can make assumptions that the primary interaction vertex is more likely to be closer to the beam entry, whereas for cosmic or atmospheric neutrino interactions this is not

true. Similarly, different algorithms may be better suited at dealing with downward facing muons versus the muons produced in test beam or neutrino interactions, which will travel forward through the detector. An example of this approach is shown in Figure 4.4, showing the decision-making between Pandora Cosmic and Pandora Test Beam / Pandora Neutrino at surface-based detectors. Here it can be seen that multiple chains can be run simultaneously, allowing a particle to be reconstructed under multiple hypotheses and the best result used.

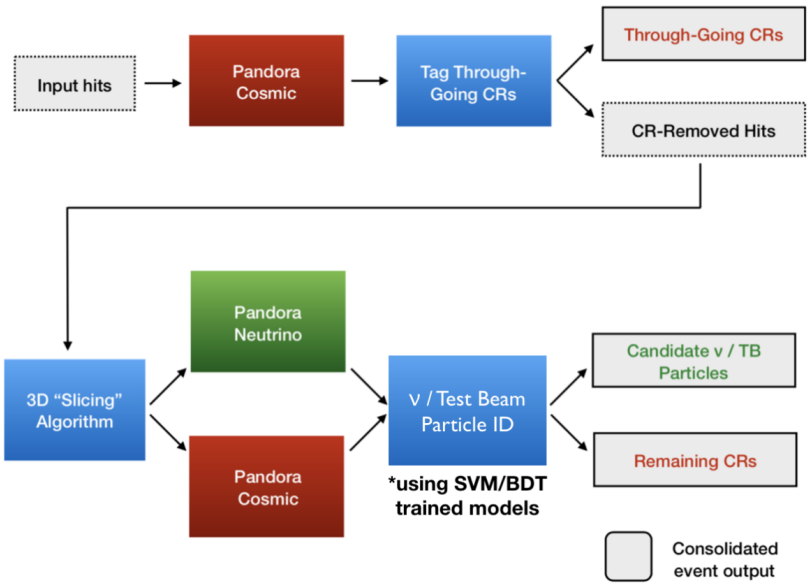
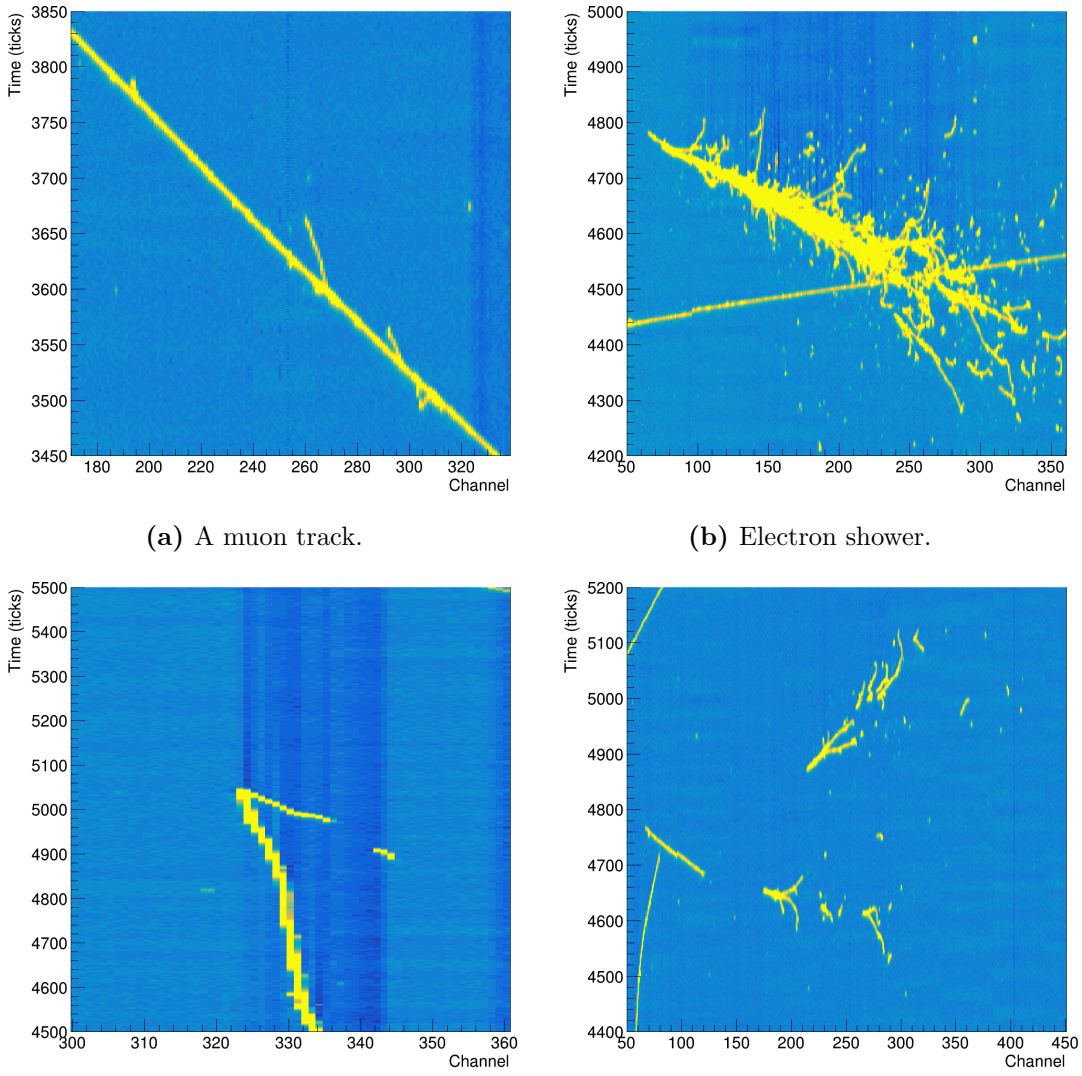


Figure 4.4: Schema showing the Pandora consolidated outputs and reconstruction strategy for surface-based LArTPCs such as ProtoDUNE and MicroBooNE. This shows the broad chains that are used at different points and where decisions are made to pick between them dynamically. Figure is taken from [31].

The multi-algorithm approach employed by Pandora enables dedicated algorithm chains to be developed for specific event topologies. In liquid argon, particles can broadly be split into two categories: track-like and shower-like particles. Track-like particles, such as muons, pions, and protons, travel in mostly straight, continuous lines through the detector, losing their energy primarily via ionisation. Shower-like particles on the other hand, such as the electron and photons have much less well-defined paths through the detector, due to losing energy via radiative emissions, creating electromagnetic cascades. These electromagnetic showers are much wider, with hits being created in a much less defined way than the straight lines of track-like particles. Figure 4.5 shows some examples of track-



(a) A muon track.

(b) Electron shower.

(c) Michel electron.

(d) Photon showers.

Figure 4.5: Particle signatures in ProtoDUNE-SP data for candidate particles from different particle species. These figures show some of the complexity / ambiguity in a strict track-like and shower-like distinction. The Michel electron candidate in Figure 4.5c (the decay electron starting around channel 324, at a time of 5020), is defined as an electron which is a shower-like particle, but at this low energy it exhibits track-like behaviour. Similarly, in Figure 4.5d there are track-like components at the start of each shower. All the figures shows raw detector readout from the collection plane, before any noise filtering or deconvolution is applied. Figure is taken from [75].

like and shower-like particles, as well as examples of cases where the distinction is far less clear. A consequence of these contrasting particle topologies means that different algorithms work best for different particle types. Whilst a straight line-based fit may accurately reflect a track-like particle, a better fit for a shower would be a cone-based fit, to reflect the spread of the shower not just the direction. Additional complexity arises due to the imperfect nature of this distinction, as shower-like particles also exhibit track-like properties before they begin showering. All of this means that Pandora treats tracks and showers differently for much of its reconstruction, using algorithms optimised to tackle their specific topology, rather than general algorithms that attempt to do both. In practice, this concept is combined with the algorithm chains to provide very specific algorithms that can target specific event topologies. For example, the cosmic ray algorithm chain utilises this concept to more heavily focus on the prominent tracks, with shower-like reconstruction only needed for the delta rays from the muons.

A complete example of a neutrino event being reconstructed in Pandora is shown in Figure 4.6, followed by an extended explanation of some of the steps that Pandora uses to reconstruct a beam neutrino event without cosmic ray interactions.

4.3.2 Initial Hit Clustering

The earliest stage in Pandora does the basic clustering that will be used to make up the full event eventually, creating initial 'proto-clusters' that are used to represent continuous lines of 2D hits. It is crucial that these initial clusters are of high purity, that is, contain only hits from a single MC particle, even at the cost of low completeness, where low completeness refers to containing a low percentage of the total MC hits for a given true particle. This approach works well for the tracks, since they are mostly continuous sections of hits, such that a single cluster can cover a large portion of the true track. However, for showers and their more segmented appearance in the detector, this results in hundreds or thousands of high purity, low completeness clusters that will be later examined and merged, using topological and more developed event information to merge them more intelligently than would be possible initially. This is because the more developed the event model is, the larger the amount of usable information. For example, a topological feature such as the shape and size of the hits produced by a particle, alongside a direction vector calculated from its hits, makes the selection of a vertex much easier than just raw hits. Figures 4.6.1 and 4.6.2 show the input hits and then

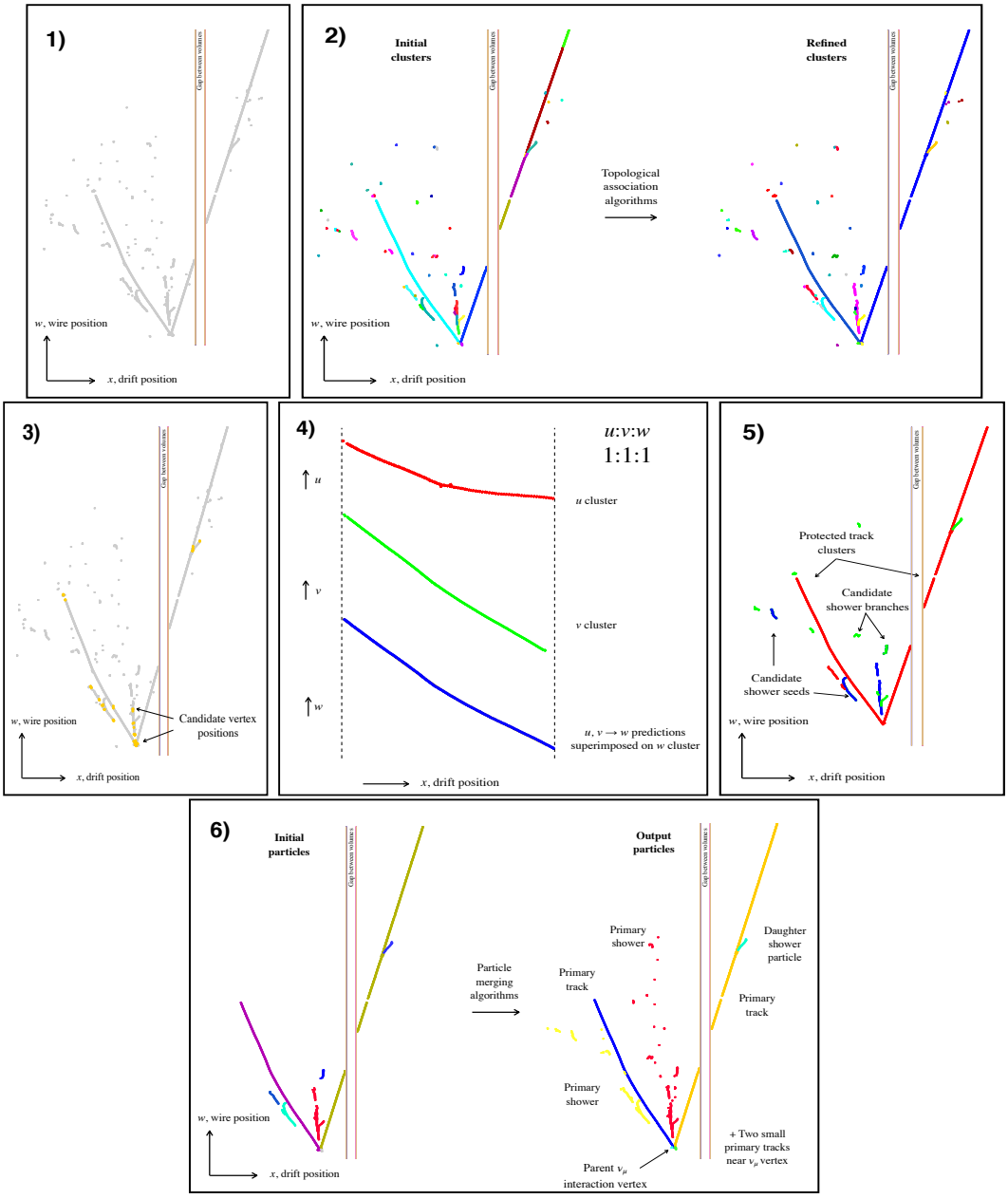


Figure 4.6: Illustration of the main stages of the Pandora pattern recognition chain: **(1)** Input hits; **(2)** 2D track-like cluster creation and association; **(3)** 3D vertex reconstruction; **(4)** 3D track reconstruction; **(5)** Track vs Shower separation; **(6)** 2D and 3D particle refinement and event building. Each of the figures includes an indicator for its coordinate system, as well as any defining features. Figures all taken from [31].

the initial clustering for an example event, showing how once clusters are created, topological features can start to be used to join together clusters that are close together with consistent directions.

4.3.3 3D Vertex Reconstruction

Once the initial clustering has been performed, sufficient topological information is available to enable the reconstruction of the neutrino interaction position. This is a key part of the reconstruction, such that additional care needs to be taken around the vertex to ensure that primary particles that emerge from the vertex are protected and that complex events with many final state particles are reconstructed correctly, with each particle being reconstructed fully back to the initial vertex. In Pandora, vertexing is achieved by utilising pairs of 2D clusters from multiple views, creating a list of potential 3D vertex positions. These candidate vertexes can then be scored, and the best performing vertex is picked. This scoring of the vertex utilises a boosted decision tree (BDT) [76] to score each candidate and picks the best of the total list, using a combination of calorimetric features, topological features such as the shape and size of the event, and simple counts of features in the event such as the number of hits and clusters. There has also been work to improve this BDT further to aid the vertexing step with deep learning. Figure 4.6.3 shows a number of the candidate vertices that have been found for the example event, before a single best vertex is picked.

4.3.4 3D Track Reconstruction

With well-defined 2D tracks in each of the three views, Pandora can begin to reconstruct 3D tracks. The same track cluster needs to be identified across each view, which can then be combined to calculate the final 3D track particle. This step can also be used to refine the existing 2D clusters by solving ambiguities in one view by using information from a different view. Pandora has developed an iterative approach to this problem that considers all the possible combinations of the 2D track-like clusters and builds up a comprehensive set of cluster-consistency information. This set can then be queried to help understand any ambiguities in one view that are not present in other views. Similarly, features in 2D that correlate in 3D can be used to make an even stronger association between distinct 2D clusters. This step first targets the most clear combinations of 2D clusters, before

performing the cluster-matched ambiguity corrections iteratively to enable full 3D particle creation. If required, the 2D clustering can be refined to help improve the level of agreement in 3D. Figure 4.6.4 shows an example of a comparison being made between the views, by projecting from the u and v view onto the w view to calculate their correlation.

4.3.5 2D & 3D Shower Reconstruction

As mentioned previously, tracks and showers are treated differently throughout the reconstruction chain, due to their differing appearance in the detector. For showers, the largest shower-like clusters are used as "seed" clusters for both 2D and 3D shower reconstruction. These central, long clusters are recursively built up with shower branches, before the branches are extended with additional branches and so on. Once this is complete, a similar process to the 3D track reconstruction is applied to the 2D shower clusters in each view to build up the 3D shower particles. Figure 4.6.5 shows the improvements made to the initial shower-like clusters, as well as the tagging of the clusters and branches, which can then be used to refine the showers further.

4.3.6 2D & 3D Particle Refinement

The final stage of the Pandora algorithm chain refines the 2D and 3D clustering. The refinement tools aim to improve the completeness of the reconstructed particles by merging nearby particles that look to be fragments of a single larger particle. This problem is approached with both 2D and 3D algorithms, with the most powerful algorithms using a combination of both. Features in the 2D views can be combined to look for features in 3D, or 3D features can be projected back to the individual 2D views, taking advantage of the coordinate transformation system in Pandora. This stage finishes with the final creation of 3D space points for every 2D input hit, before using this full 3D representation of the event to build up an interaction hierarchy, going from the initial neutrino, to each daughter particle and then any subsequent daughter particles they may have. This allows analysers to not only target specific particles in an event, but specific interaction hierarchies, such as a pion but only if it is associated with the primary interaction, rather than any secondary interaction. Figure 4.6.6 shows the input and output of this stage, with additional hits from the original input being included now, as well

as the two primary showers being fully merged with the shower branches nearby.

4.3.7 Performance Metrics

As mentioned briefly in the previous sections, a few key metrics are used to assess reconstruction performance, which are useful to discuss, to outline their impact and quirks.

These metrics broadly are as followed for simulated data:

- **Efficiency**: Efficiency here refers to a measure of how well the entire event has been reconstructed. It is defined as the number of MC particles that were matched to at least one reconstructed particle, over the total number of reconstructed particles. A match from a reconstructed particle to a MC particle is determined by the particle whose hits contribute most to the reconstructed particle, whilst also requiring a purity over 50% and a completeness above 10%. This gives a more global, event level reconstruction metric, versus completeness and purity which are particle level.

$$\text{Efficiency} = \frac{\text{Number of MC matched to at least 1 reco particle}}{\text{Total number of MC particles}} \tag{4.3.1}$$

- **Completeness**: Completeness in this context refers to how "complete" the current reconstructed particle is, that is, how many hits does it have out of the total hits associated with the main MC particle for that reconstructed particles. For example, if a reconstructed particle has 100 hits, with 95 of them coming from a muon and 5 hits from a nearby electron, the main MC particle for that reconstructed particle would be the muon. The completeness of that particle would then be 95 over the number of true hits associated with the muon in the MC. Completeness can also be given in terms of energy, to calculate how complete the energy of the particle is out of its total energy. This is achieved in the same way, with the truth matched energy over the total MC energy. Completeness is a useful metric to tell if the clustering and particle refinement has created sufficiently large particles. Low completeness refers to a reconstructed particle that is missing a large number of its hits, which in turn will impact energy reconstruction and more.

$$\text{Completeness} = \frac{\text{Number of hits shared between MC and reco particle}}{\text{Total number of MC Hits for main MC}} \tag{4.3.2}$$

- **Purity**: Purity is complementary to completeness. Where completeness tells you how much of the particle was put together, purity tells you how much contamination there is from other particles. Using the same example as for completeness, that is a particle built with 95 hits from a muon and 5 hits from an electron, the purity is the number of truth matched hits (95) over the total number of reconstructed hits for that particle (100). Similar to completeness, this can also be performed with the reconstructed energy, rather than the hits. Purity is useful to give the complementary information about the clustering to completeness, such that it can be seen if the clustering is too aggressive and is leading to merging multiple particles that result in low purities.

$$\text{Purity} = \frac{\text{Number of hits shared between MC and reco particle}}{\text{Total number of reco hits}} \tag{4.3.3}$$

These metrics, especially completeness and purity, are most useful when taken together. Achieving high completeness in an event is easy if you combine the whole event into one particle, but that would then achieve a low purity. Similarly, having clusters of size one achieves a high purity, but is of no actual use. Instead, particles with high completeness and high purity are desired. Efficiency as a metric is useful to understand how changes have impacted the event as a whole, but is only useful if the actual purity and completeness are expected to change drastically, so its use is more limited. Pandora by default favours using the hit-based version of these metrics, rather than energy-based ones, as the primary input to Pandora is hits, making hit-based metrics more intuitive. These metrics can then be split up further if desired, such as by energy, number of hits or particle type to understand how performance varies depending on the size of the event or the interaction type.

One key choice for performance metrics is the initial object used to build them. Metric generation can start from either a reconstructed object, or a truth level object. Starting from a reconstructed object, a 'reco-first' approach, tells you what the reconstructed objects look like, such as how complete or pure they are.

However, what it does not tell you is the result for any MC particle that was not reconstructed, as there is no object to start from. Starting from the MC particle, a 'MC-first' approach, and then analysing the completeness and purity of the reconstructed objects that make up that particle is a second way of building up metrics. Each way provides different information, with 'reco-first' outlining how the reconstructed objects look and perform, but the 'MC-first' metrics give a more complete overall event understanding. For this reason, the reconstruction first metrics are broadly used first, but with an eye on MC-based metrics such that more objects are not being lost in an event.

Examples of Pandora's achieved reconstruction performance at ProtoDUNE-SP can be seen in Figure 4.7, showing the accuracy of reconstructing the particle's end point. Reconstructing the end point of the test-beam particles is of particular interest to cross-section analyses, where it is critical to know if a particle interacted or stopped. Figure 4.7 shows that for 68% of beam particles, Pandora can reconstruct the end point within 2 cm of the true value. We can also see the different reconstruction efficiencies for each of the coordinates, with the X coordinate being the best, due to that being common across all views of the detector, whereas the Y and Z are more difficult to reconstruct. The performance numbers shown represent a snapshot of Pandora's performance, as continual development and tuning to Pandora mean that its performance is improving over time.

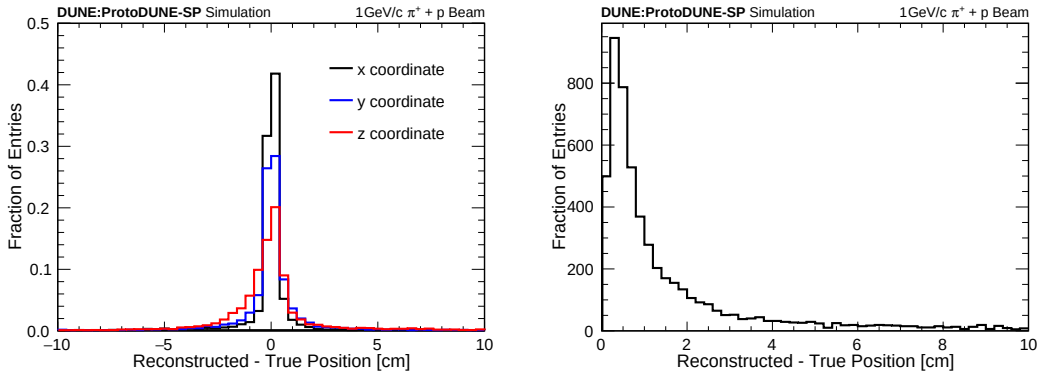


Figure 4.7: Pandora's reconstruction performance on ProtoDUNE-SP simulation, showing the difference between the reconstructed and true position of the particle end. The left shows this split into the x, y and z components, whereas the right shows the combined three-dimensional distance. Both plots are for 1 GeV/c primary proton and charged pion test-beam particles. Plots taken from [72].

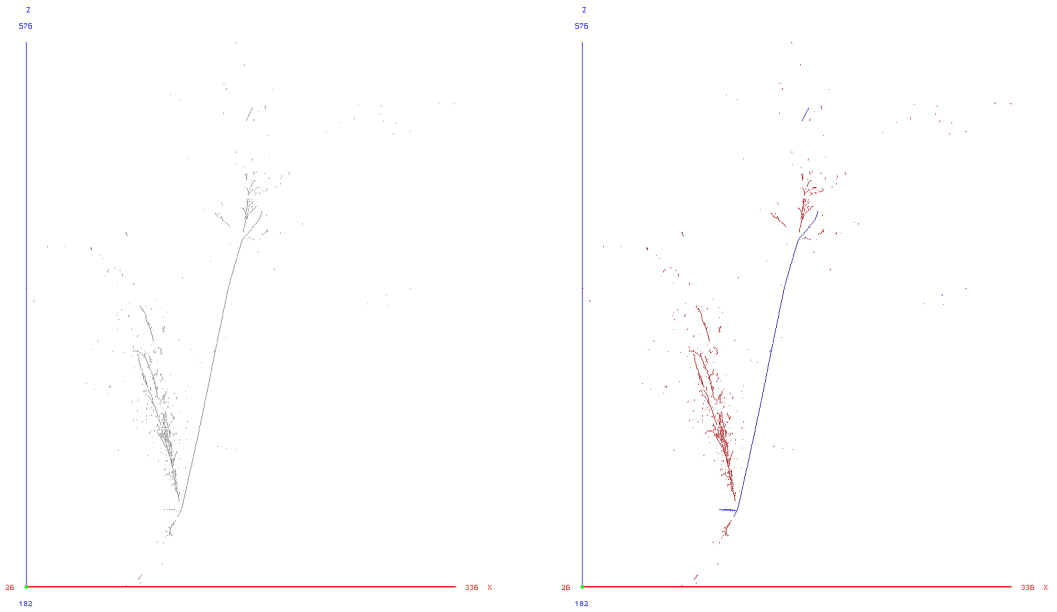
4.3.8 Deep Learning

One of the more recent developments in Pandora is a more thorough integration of deep learning to steer the direction of the Pandora algorithm chains. Pandora already extensively uses machine learning techniques such as BDTs and other multivariate analysis (MVA) based techniques to aid decisions in certain cases, such as assessing vertex options or particle classification. This is illustrative of how machine learning is mostly used in Pandora, to aid decision-making and enhance the power of existing algorithms in Pandora. Deep learning allows this to be extended, with more powerful algorithms that target specific problems, as well as making better use of existing algorithm outputs.

In the past few years, Pandora has added an optional integration with PyTorch [77], to allow the development of deep learning powered algorithms, to either replace or work alongside the existing algorithm chains. PyTorch is an open-source machine learning framework, based on the earlier torch library [78] and used across a wide range of machine learning tasks, currently maintained and written by Facebook's AI research lab. Deep learning-based algorithms are an interesting extension of the Pandora multi-algorithm approach, and as Pandora does not specify how an algorithm is implemented, there is no difference technically between a deep learning or non-deep learning-based algorithm, outside including additional PyTorch headers. Both deep learning-based and non-deep learning-based algorithms can be run at the same time, allowing the most useful algorithm to be used depending on the current problem.

Deep learning-based algorithms are currently being tested in a variety of forms, to both augment existing algorithms, and provide new ways to approach problems. A key example of this is the use of deep learning early in the reconstruction chain to classify every hit in the event as having come from either a shower-like or track-like particle. This algorithm takes the raw hits of the event as input, and outputs an identical sized output, with a score for every hit indicating how track or shower-like it is [79]. An example of how the algorithm performs is shown in Figure 4.8. This score can then be used for a variety of algorithms, with the most powerful being the ability to split the reconstruction chain. Pandora already targets track and shower-like topologies differently, but the presence of an accurate track or shower score for every hit means that algorithms designed for one topology can be run only on hits or clusters that have a very high track or shower score, excluding those which would not benefit from the current algorithm. This process,

called streaming, referring to the two different algorithm streams that the track and shower-like clusters now follow, allows impressive performance improvements using existing algorithm chains.



(a) Raw Input Hits **(b)** Tagged Output Hits

Figure 4.8: An example DUNE FD event with the results of the Pandora hit tagging algorithm. The raw hits are used as input to the deep learning-based algorithm, with the output being seen on the left. The output shows track tagged hits in blue, and shower tagged hits in red. The actual score is a scale with how track or shower-like the hits are, but the visualisation here only shows which classification fits better, not the strength of the classification. This is for a simulated electron neutrino event. Further details on this network can be found in [79].

Going forward, the streaming work allows for the introduction of new algorithms, both deep learning-based and otherwise, that target the two particle topologies more directly, with a much higher confidence that the input hits are compatible. There is also the potential for other similar early classifications, such as a CVN-like network [69], outlined in Section 4.2.3, to classify the neutrino flavour in beam events, such that ν_μ and ν_e events could be treated differently to target their more track-like or shower-like topologies more directly, or allow for cosmic rays or atmospheric neutrinos to be targeted more directly in a dynamic way, with specific tagging of the hits belonging to certain interaction types. Similarly, extension or replacement of existing algorithms for complex problems such as vertexing with deep learning-powered algorithms could yield

either significant improvements, or extensions of these algorithms to provide extra information, such as secondary interaction vertices. Deep learning-powered algorithms will become a key part of the Pandora algorithm chain going forward, with an extensive suite of handwritten algorithms for simple problems, or problems where physics restrictions that are not easily encoded in a network make a deep learning approach unfeasible. This approach allows Pandora to maintain the advantages of deep learning-powered algorithms where they are most powerful, but also use simpler algorithms in the cases where it is unnecessary to use deep learning.

5

Deep Learning

"Say from whence you owe this strange intelligence?"

Macbeth - Macbeth

Machine learning (ML), and more recently deep learning (DL), are rapidly growing fields of research that study algorithms and their associated tooling that can make predictions from data, using relationships learnt from initial training data. Recent advances in both available computing power, and new algorithmic advances, have enabled the stacking of more and more of these methods, each of the layers in this stack with large numbers of parameters, leading to the viability of deeper models, and the rise of deep learning rather than machine learning[1].

There are many ways of categorising deep learning methods. Two important categories are supervised and unsupervised algorithms. In a supervised algorithm, the outputs are well-defined, and a set of labelled data is available to direct the algorithm's training. That is, by knowing the predicted result and the real result, the algorithm can be updated directly to nudge it towards the correct answer, with the full process for this outlined later. In the unsupervised case, the goal is to infer the patterns in the underlying data, without having labels for the input data. Deep learning algorithms can also be distinguished by the different outputs that are required, either discrete or continuous, or a single output or multiple

[1]The distinction between ML and DL can change depending on who is asked. Here, deep learning will refer to techniques that utilise larger networks with many layers, not simpler techniques such as BDTs or other MVA techniques.

outputs. In each case, the desired output informs decisions about what sort of techniques or features to use, either to gain the most amount of decision-making power out of the input data, or to simply get an output of the form required. This leads to different architectures that aim to exploit the input structure to gain additional information out of the input data, rather than having to learn some structure from a one dimensional input.

This chapter will review the essential techniques of DL and provide an in-depth discussion of the types of network that will be applied in Chapter 7, with some additional context of how deep learning is currently used in particle physics at DUNE and other experiments.

5.1 Neural Networks

Artificial neural network (ANN) are a class of ML algorithm that take inspiration from the biological neurons that are present in the brain. An ANN has two main components; nodes and connections between those nodes. These nodes usually take the form of artificial neurons, where a neuron takes N inputs and passes them through an *activation function* f to give a single output, as shown in Figure 5.1. The input x is combined with some weights, w, and summed, before being passed through f to create the output of that neuron. The weights here are the tunable parameter of this neuron, allowing its value to be changed and react to the problem the neuron is made to solve. There is an additional term, the bias b, which is used to adjust the baseline calculated before the activation function is applied. This bias term can be considered like the intercept value for a straight line, with the weights enabling the gradient to be adjusted, but the bias applying a constant shift to the line to enable a better fit to the data. These artificial neurons can be stacked, with multiple artificial neurons comprising a single layer in a larger network. In this format, the output of one neuron either makes up the output of a single layer, feeding into a secondary layer of neurons, or if the neuron lies in the final layer of the network of neurons, it forms the output prediction.

Figure 5.2 is an example of the most common form of ANN, the multi-layer perceptron (MLP). The MLP is an example of a network formed of layers of artificial neurons. A MLP commonly has a minimum of 3 layers, one for input, one for output, and at least one "hidden" layer. Each layer is usually *fully connected* to the next layer, such that every neuron in one layer is connected to every neuron

in the next layer.

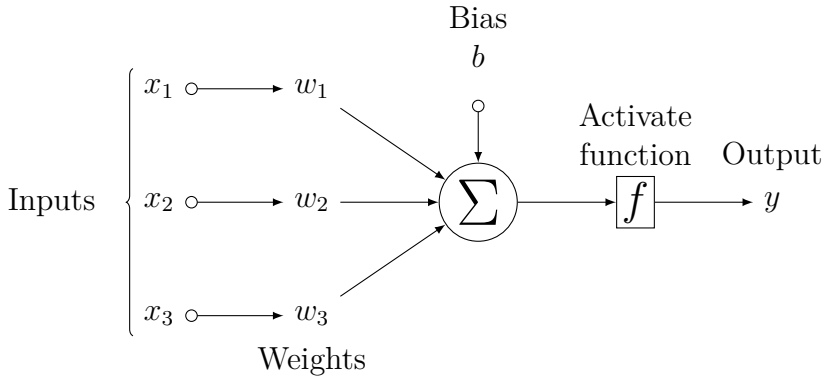


Figure 5.1: A graphical representation of an artificial neuron, which make up the individual nodes in an artificial neural network. x_1, x_2, x_3 represent the inputs to the neuron, with w_1, w_2, w_3 being the associated weights. Once the inputs and weights are combined by summing $b + x_i w_i$, the result is passed through some activation function f to produce the final output y. Here, b refers to the bias, which is a value used to adjust the sum of the weights and inputs to the neuron.

In a MLP, the nodes in each layer receive input from each preceding layer, which are then used to calculate their output value, which forms part of the input for the next layer. Given an arbitrary node i in layer j, the output $o^{i,j}$ is given by,

$$o^{i,j} = f\left(\mathbf{w}^{i,j} \cdot \mathbf{x}^{j-1} + b^{i,j}\right)$$

where $o^{i,j}$ is the output for the node, f is the chosen activation function, $\mathbf{w}^{i,j}$ is the weights vector, $\mathbf{x}^{j-1}$ is the input to the node from the previous layer, and b^{ij} is the bias.

The choice of activation function somewhat depends on the required output, with some activation functions giving continuous outputs, and others rounding to zero or one. Common choices include the sigmoid function, hyperbolic tangent, rectified linear unit (ReLU) and its modified versions, and the *softmax* function, with Figure 5.3 shows how some of these functions vary visually, and their definitions are as follows:

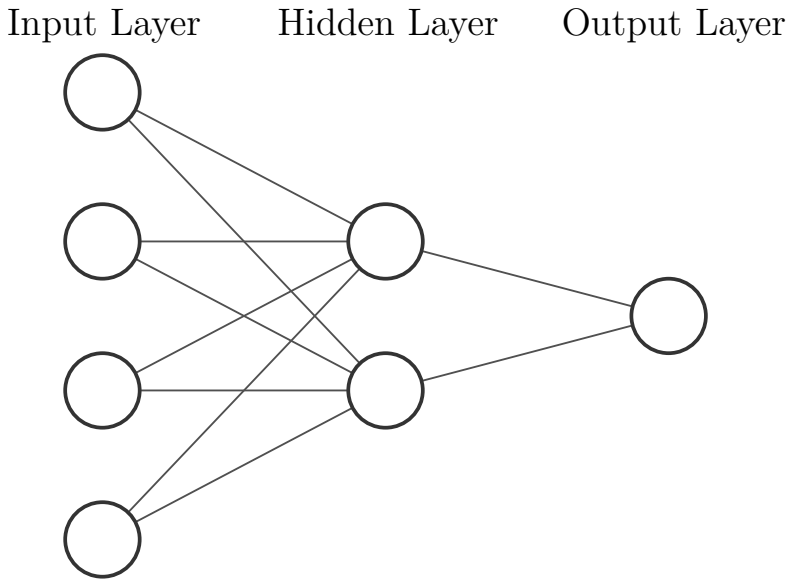


Figure 5.2: A diagram of the basic architecture of a feed-forward neural network. There are three layers of neurons. The input arrives in the first layer, before being passed to the middle hidden layer. Here, linear combinations of the input values are produced by combining the learned weights with the input values, before being passed to the final output neuron which produces the final inferred output. Figure generated using [80].

$$\begin{aligned}
\text{Sigmoid} : f(x) &= \frac{1}{1+e^{-x}} \\
\text{Tanh} : f(x) &= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \\
\text{ReLU} : f(x) &= \text{Max}(0, x) \\
\text{Softmax} : f(x_i) &= \frac{e^{x_i}}{\sum_l e^{x_l}}
\end{aligned}$$

Here, x_i indicates the current node and l refers to summing over all nodes in the current layer. This allows outputs from a layer that uses softmax to sum to one, which makes it a common choice for the final layer in categorisation tasks to get a value for each category whilst adding up to one. It is common for a combination of these activation functions to be used, for example using a ReLU based activation function for each layer, with a softmax or sigmoid function as the activation function for the final output layer.

Alongside picking an activation function to use for each layer, the number of layers and their sizes must be picked. Usually, the input and output layer have

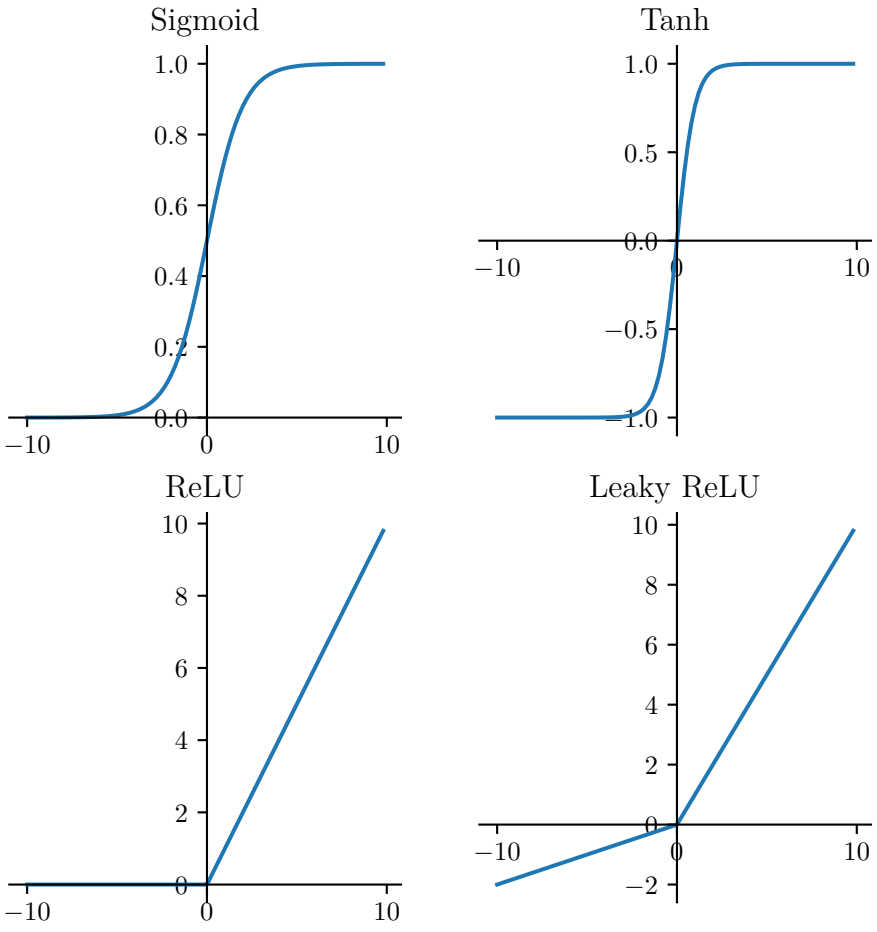


Figure 5.3: The shapes of four different activation functions. The equation for the sigmoid, tanh and ReLU functions are given in Section 5.1. The leaky ReLU refers to a variant of ReLU that has a slight negative slope for negative values, rather than being clamped to zero. In this case, the negative slope is given by $0.2 * x$, but the coefficient can be altered.

a well-defined size based on the input to the network, and the desired output. This output could be a single node for a task with a single continuous output, or something with binary classification, or multiple nodes for a categorisation task. The hidden layer size and how many hidden layers is where there is more ambiguity. According to the universal approximation theorem for ANNs, it is possible to approximate any function to an arbitrary precision with only a single layer with enough hidden nodes [81]. However, in reality, it makes more sense to use a lower number of nodes, split into multiple hidden layers versus a single hidden layer with many hidden nodes, as many fewer nodes are required when using multiple layers of neurons. There is a balance between how many hidden layers and their size, that must be found, as going too large gives diminishing returns but increases the computation cost of the network with each additional layer.

5.1.1 Gradient-based optimisation

A given network does not start out being able to complete a target task, being initialised with random weights that will produce an equally random, nonsensical output. In a supervised network, these weights can be gently pushed towards a solution with *gradient-based optimisation*, taking the initially random weights of the network and updating them over time to produce weights that give a sensible answer, assuming the network is of sufficient size and capacity to learn from the input data. A similar approach can be used in unsupervised networks, but with a lack of labelled training data, it is necessary to use scores or metrics to rate a given output.

When using a neural network normally, we give it some input x and have it produce some final output y, with information flowing forward through the network, starting from the initial information from the input and then propagating via each of the nodes of the network until the output is produced. This whole process is called *forward propagation*, and a similar method called *backpropagation* [82] is used to enable the network to learn. Backpropagation allows the information of the network to flow backwards from the output, and a loss is calculated, which is the difference between the training data and the predicted output. From this loss L, a gradient can be calculated, which can then be used to update the network's weights in a way that improves the calculated loss. This is typically calculated by quantifying the difference between the network's predicted

answer and the true output, at least in the supervised learning case. A common example of this would be the *mean squared error* of the training set. This allows the weights of the network that produced a wrong error to be nudged closer to a value that would produce the desired output.

This optimisation is a core part of machine learning-based algorithms, defined by minimising some form of *cost function* $J(\boldsymbol{\theta})$[2] by changing parameters $\boldsymbol{\theta}$ (which relates to both the weight and bias in this example). J typically refers to the average of the loss function L over the entire training data set. The partial derivatives of J can be computed, which in turn shows how J would react to changes to $\boldsymbol{\theta}$ (w and b). Minimising this loss function $J(\boldsymbol{\theta})$ is usually achieved through the use of gradient-based optimisation, which uses the partial derivatives, also known as gradients, of $\nabla J(\boldsymbol{\theta})$ to find the optimal $\boldsymbol{\theta}$ so J is minimal. Suppose that this loss function at a given time step t is $J(\boldsymbol{\theta_t})$. Given the first-order Taylor series of J about $\boldsymbol{\theta_t}$,

$$J(\boldsymbol{\theta_t} + \Delta\boldsymbol{\theta}) \approx J(\boldsymbol{\theta_t}) + \nabla J|_{\boldsymbol{\theta_t}} \Delta\boldsymbol{\theta}^T,$$

we know that for small changes to the parameters, as given by $\Delta\boldsymbol{\theta}$, J can be reduced by moving along the negative gradient, moving us closer to a more optimal answer. This technique is called gradient descent, and it chooses values of $\Delta\boldsymbol{\theta}$ such that it is parallel to $-\nabla J|_{\boldsymbol{\theta_t}}$. Then for the next time step, $t+1$, the parameters are now given by

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \alpha \nabla J|_{\boldsymbol{\theta_t}},$$

where α is the *learning rate*, a tunable hyperparameter of the network. The first-order Taylor approximation only holds for small values of $\Delta\boldsymbol{\theta}$, so choosing a suitable learning rate is important. It should in general be small, as using a learning rate that is too large can lead to repeated jumping over a minima in J. Conversely, if α is to small, that means the network will take longer to learn and train, or get stuck in an undesirable local minima. There is a balance in picking a suitable value for the learning rate.

[2]This can also be called an objective function, error function or a cost function.

Name	**Description**
Learning Rate	Controls how quickly a networks weights are updated.
Activation Function	What specific function to use for a given node.
Layer Size	The number of nodes that go into a given layer.
Layer Number	The number of layers in the network total.
Dropout	Percentage of nodes to be randomly ignored whilst training.
Epochs	The number of training iterations to perform.
Batch Size	How to split the data into batches, which are then trained upon.
Optimisation Algorithm	The algorithm used to control the training process.
Kernel Size	The size of the kernel applied to the input image.
Pooling Size	How much to reduce the image size by.
Padding	If padding should be applied to keep the image size consistent.
Aggregator	The aggregation function to use for message passing.
Message Passing	The rounds of message passing to perform.

Table 5.1: An overview of common hyperparameters used in training neural networks, and what they are for. Examples of some specific hyperparameters used in more complex networks are given in later sections.

5.1.2 Training a Network

Outside the mathematical process of gradient-based optimisation and how a network learns, there is an additional set of technical challenges involving training a network, related to both hardware constraints and software engineering issues. Training a neural network to be effective requires a large amount of labelled training data, at least in the supervised case. There are considerations around this data that need to be accounted for in the training process, such as its format, how the data is split between training and testing, and the amount of available data for training. Additionally, a broad explanation of the training process is useful for later reference. Finally, most networks have a large number of hyperparameters, that is, values that control the learning process in some capacity, that need to be determined. These hyperparameters, ranging from the size and shape of the neural network, to the value of the learning rate, need to be optimised to enable the network to efficiently learn the required problem. An overview of some common hyperparameters is given in Table 5.1.

Creating a suitable training dataset is an important first step for most deep

learning algorithms. In non-supervised cases, it is possible that a dataset is not needed, and instead some form of figure-of-merit score can be used to guide the network (for example, using the score in a video game). However, in a supervised case, a comprehensive and labelled dataset is needed, to compare the predictions against and make adjustments accordingly. The size of the dataset is dependent on the complexity of the problem, so like most of the training parameters, there is no simple way of knowing if more data are needed prior to training. For particle physics specifically, the sophisticated MC simulations we have for understanding and benchmarking the detector make this problem much easier, as we have a simple way of producing large amounts of data to train on[3]. Once data has been acquired in the correct input format for the chosen network, the data is usually split into three sets: a large training set, a smaller validation set and a final test set, though it is possible to set up the various datasets in different ways depending on the data in use and the aim of the training process. The training set is used to update the weights of the network, whilst the validation set is used as a benchmark of training performance whilst training is underway. The validation set is useful to ensure that the training process is not learning features too specific to the input training set, quickly, whilst training. Finally, the test set is used to assess the performance of the network on unseen data.

As most datasets for deep learning are in the tens of thousands of examples, accommodations need to be made in the training to deal with the amount of data, as it is not possible to load and training on all the examples at once. The most common way of dealing with this is the use of *mini-batches*, small samples of the training dataset that are trained upon and then have backpropagation applied to simultaneously. The size of this mini-batch is a hyperparameter of the network, with the size of the mini-batch impacting the convergence of the network. The use of many mini-batches usually leads to quicker convergence of the network, due to more frequent updates to the network, as well as alleviating restraints on the dataset size due to memory. One full pass through the dataset over every mini-batch is called an *epoch*, such that the network has seen every training example exactly once. This is then repeated until satisfactory performance is achieved, usually with some amount of shuffling of the dataset each time to ensure robustness of the learning process. During this time, the validation set can also be used to

[3]Access to sophisticated MC data is also a problem however, increasing the risk of learning features from a specific physics model, rather than the problem generally.

ensure the network is learning sufficiently and not overfitting, that is, not learning features that are specific to the training set alone that will not generalise to unseen data.

Like reconstruction metrics, there are common deep learning metrics to assess the performance of a trained model. As deep learning is applied to a variety of problems, the specific metrics depend on the problem. For example, classification problems traditionally use accuracy, precision, recall and more. Like completeness and purity in the context of reconstruction performance, these numbers need to be taken in context with each other. For classification-based tasks, accuracy is the simplest metric: how many correct predictions were made out of all predictions, whereas precision refers to the proportion of positive identifications that were actually correct, and is calculated with the number of correct predictions over every positive prediction. These two numbers tell two different things: accuracy says how often the network is correct overall, whereas precision indicates how good the network is at predicting a specific category. The distinction between these two metrics can be very useful, for example it is usually easy to achieve high accuracy in a dataset that is not balanced by always predicting the dominant category. There are further metrics that are useful to view together, but a more useful metric in the context of training is understanding how loss changes during the training process. As mentioned previously, we can calculate a loss value for the network, and this value should improve as the network learns. Similarly, we can look at the loss on the validation dataset, which is not trained on. This means the loss can be used to check that the network is actually learning, and to help tune the learning rate chosen, but also enables overfitting to be caught, as the training and validation set losses will diverge, as the model starts to learn features specific to the training set rather than the general dataset. However, it must be noted that loss itself is not usually directly comparable across different networks, unlike most other metrics. Instead, it is only useful to check the training process, and potentially compare different tunings of the same model. Figure 5.4 shows an example of how the training and validation loss can change over the duration of training, as well as how accuracy on the test dataset improves as the network learns.

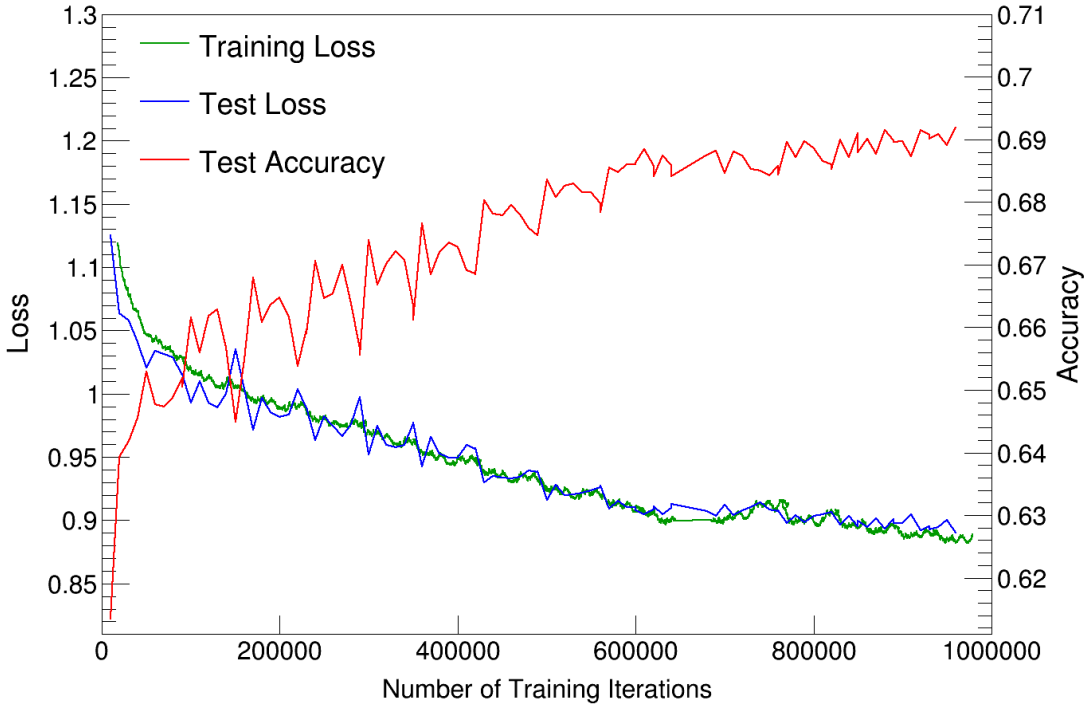


Figure 5.4: An example of how loss changes over the duration of training, from the NOvA CVN. The average training and test loss can be seen declining over the duration of the training, where the test loss is the average loss over 256,000 test examples and the training loss is the average over 128,000 training examples, averaged across iterations using a sliding window average. The accuracy on the test dataset can also be seen increasing over time. Figure taken from [24].

5.2 Network Types

There is a range of different architectures for deep learning networks, each with specific features of their structure that enable them to more easily learn from their supplied data. Certain problems are either more easily represented in a certain form, or can be more easily reasoned when given in a more natural form. For example, learning from an image directly is an easier task than flattening that image into a 1D vector and hoping the network can learn the associations between the pixel inputs. There is also the possibility of being able to encode additional information into an input more easily when given in a different form. Thus, having networks that natively understand their supplied input and can learn from that directly are useful. This has led to the rise of specific networks that are optimised for certain inputs, as well as networks whose structure is optimised for a certain task.

Some of these networks are specific to a certain style of data or problem. Examples of this include convolutional neural networks (CNNs), graph neural networks (GNNs) and recurrent neural networks (RNNs), each which are optimised for a specific input or problem definition. CNNs and their derivatives are designed to deal with multidimensional inputs on a fixed grid, which can be a more intuitive way of learning for some inputs, utilising the natural shape of the data, rather than requiring it to be flattened to a single dimension, which can make context of each data point harder to understand. This has lead to them becoming the standard network type to use to reason on images, from medical use to recognise features in tissue samples, to automated recognition of the contents of a photo for metadata tagging. Similarly, RNNs are optimised for dealing with sequences of data, which makes them ideal for dealing with data that depends on previous inputs in the sequence such as speech recognition [83], or other language-based applications, where the data's sequential nature is an important feature to predict the next word. Finally, GNNs are incredibly useful for leveraging the structure and properties of graphs, which can be used to show the relationships between things, making a GNN an ideal fit for problems based around relationships between objects. They have been applied to biological problems very frequently, representing molecules and their relationships, or the structure of a protein.

Other types of techniques exist within deep learning that are broadly useful, without prescribing the problem space or the input format. Examples include the

skip connections, where inputs are passed not only through a layer, but over a layer too, such that the activation of an input and the input itself can be utilised for follow-up layers. Likewise, the generative adversarial network (GAN) [84] can be used for a variety of tasks, and just refers to a class of machine learning algorithm that utilise two networks in contest with each other. This can be used to generate new outputs that should be hard to distinguish from their initial inputs, but do not prescribe the actual task at hand.

Finally, there are techniques that are used to improve the learning potential of any network, such as *dropout*, a technique of dropping out units of a neural network. Simply put, dropout refers to ignoring a random number of neurons during each training iteration. This should improve the robustness of the network, as the data it learns from is randomly zeroed, which in turn can help prevent over-fitting of the network.

5.2.1 Convolutional Neural Networks

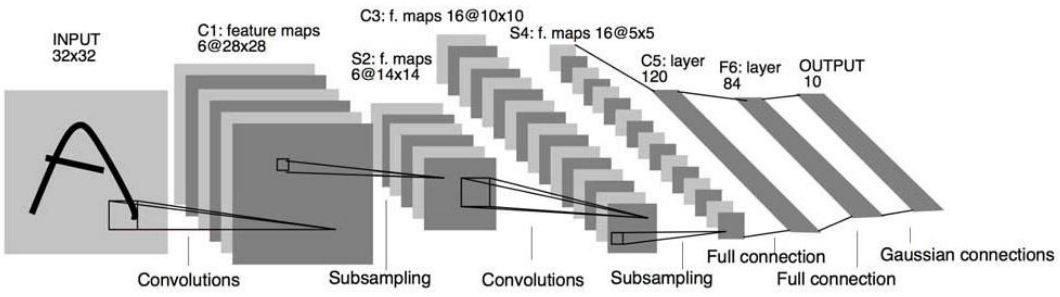


Figure 5.5: An example CNN architecture diagram, showing the LeNet-5, one of the original examples of a CNN [85]. The input image, and then layers of convolutions and pooling can be seen, ending in a final MLP. The size of the input getting smaller and smaller in each layer can be seen, with the size dropping most drastically after the pooling layers. As this network was created to classify handwritten digits, it has an output layer of size ten. Figure is from [85].

Convolutional neural networks (CNNs) are a class of ANN which are most commonly applied to images, whose input data lie on a fixed grid. The architecture of a CNN allows it to deal with higher dimensional data that a MLP may struggle with. This is because a MLP requires many nodes, both in layers and nodes per layer, to deal with data in higher dimensions, which in-turn impacts the computational cost and speed of the network. Similarly, the one dimensional

input to a MLP means losing a lot of the spatially correlated information that is present in the input image, either losing that data permanently or having to learn the relationships between the input pixels. The CNN architecture addresses these issues by incorporating the spatial proximity of the pixels that make up each image, making it easier for the network to learn spatial patterns within sets of images. An example architecture for one of the first CNNs can be seen in Figure 5.5.

The CNN, like the artificial neuron, was inspired by biological processes in the brain. This time, it is the connectivity patterns of the artificial neurons resembling the visual cortex in animals. Similarly, cortical neurons in the visual cortex only respond to stimuli from a set region of the visual field, known as the *receptive field*. This receptive field then overlaps across different neurons, building up a full image of the entire visual field. It was Kunihiko Fukushima who first used these concepts [86], building on earlier work from Hubel and Wiesel [87], to introduce many of the common layers still seen in CNNs.

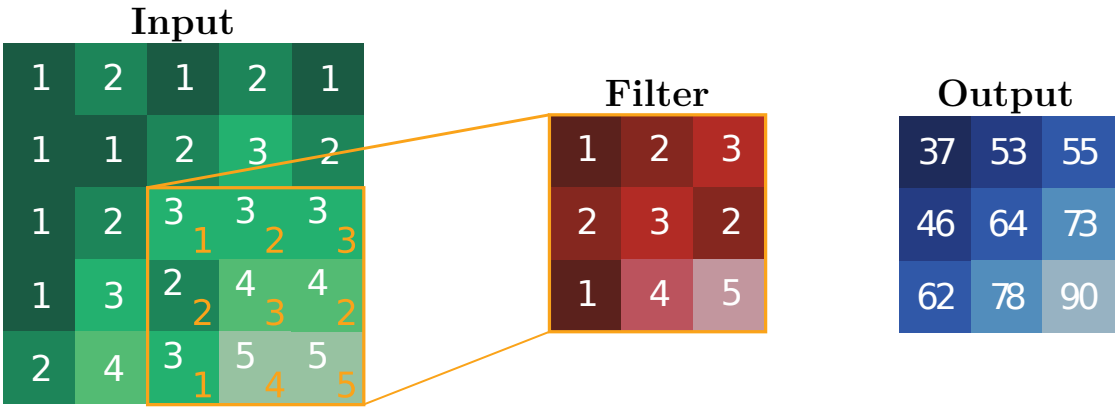


Figure 5.6: An example of a CNN filter being applied to an input image. The first matrix represents the image with its per-pixel values. The second matrix represents the filter that is being applied to the input image. Finally, the output is the result of the filter being applied to every location in the input, and then summing the elementwise products for each position. As the filter must fit exactly on the input, the size of the output is smaller. This is sometimes avoided by using padding (adding additional empty pixels around the input) to ensure the output size matches the input size.

In CNNs, the flat one-dimensional neurons that the MLP uses for its input and hidden layers are replaced with convolutional kernels. These are small matrices that contain a set of learned weights. They are applied to the image pixel-wise, by sliding the kernel over the pixels of the image and applying the convolution operator at each position. This operator is defined by

$$(x \times y)_i = \sum_{j=-\infty}^{\infty} x_j y_{i-j},$$

where x and y are discrete sequences that represent the convolutional kernel and image, respectively. As these are both finite, the actual bounds of the sum are from zero to N, with N representing the number of pixels in the kernel. The kernel is applied to all pixels in the image, producing an output that can then be used as an input for follow-up layers. This repeats, with many convolutional kernels per layer of the network, each producing output images that are passed to the next layer. One consequence of the kernel being applied to just the input image pixels is the reduction in size with each step, as the filter cannot cleanly fit on the outermost pixels without having empty pixels. In some cases, this is not desirable, such that the image is padded with zero values to ensure the input and output image are of the same size. An example of a kernel being applied to an input is shown in Figure 5.6, showing the input image and current kernel, as well as the produced output.

These kernels are essentially feature detectors, learning a specific feature in the input data, based on the value of its learnt weights that form the kernel matrix. In traditional image processing tasks, bespoke, human specified, kernels are used for many tasks, such as edge detection, sharpening and more. The outputs of these kernels are called a feature map, as they represent the spatial distribution of the feature that kernel has learnt. These output feature maps form the input for secondary layers of the network, which may be padding layers or follow up convolutional layers. The layering of the network determines the extent of the network's *receptive field*, which is the area of the input that a certain layer can see, due to the layering of the network. The receptive field is built up as the first feature map uses information from a small region of the input, but as convolutions are applied to that feature map they pull in information from a larger region of the initial input. The deeper and deeper the convolutional layers get, the larger the receptive field, the larger portion that a pixel in a feature map represents in the original input image.

Even when not using padding in a CNN, such that the image size decreases each step, CNNs have drastically larger number of parameters than a traditional MLP. This, combined with the additional cost of computing the many convolutional kernels for each layer, can lead to an increased computational cost. To combat

this, the use of *pooling* is common. Pooling [88] is a downsampling technique that helps reduce the number of parameters in the network, which in turn reduces the computational cost of the network. In pooling, data from an $m \times n$ region is downsampled to a single value, with the most common approaches being to take the maximum value from the downsampling region, or taking the average value, called *max pooling* and *average pooling* respectively. When pooling is combined with multiple convolutional layers, it can help increase the receptive field of layers deeper in the network dramatically. There is also the opposite operation, that allows the input to grow, rather than shrink. This is often used after downsampling has been applied, alongside the use of *skip connections*. This allows an image to be brought back up in size, usually back to the original size of the initial input, by reversing the pooling step alongside information from before the pooling was applied.

The output of a CNN is usually a prediction, classification or similar. Convolutional kernels are not especially suited for this task, so it is therefore common to flatten the data down and use a MLP for the final layers of the network. As there are usually many layers of convolutions followed by pooling, which drastically reduces the size of the input, the number of parameters at the MLP stage is usually much more reasonable, meaning a result can be achieved quickly. At a high-level, this can be seen as building up layers of feature maps, with later layers being feature maps built on feature maps to find higher level features, with the final dense layers being used to perform decision-making on these high-level features to produce a final prediction. However, there are certain cases where the feature maps of the CNN do not need flattening, and instead predictions can be made of the multidimensional data.

5.2.2 Graph Neural Networks

Graphs are a data structure, used to describe systems of relationships or interactions between objects. Taking a general view, a graph is simply a collection of nodes and edges[4], objects and interactions between those objects. A simple example of a graph is a social network, with nodes representing individuals, and edges representing friendship between two individuals. Information can be associated with both the nodes and edges, such as an individual's name, or the length of time two individuals have been friends. An example graph showing a social network is shown in Figure 5.7. A graph more generally can be used to

describe the relationships between nodes via their interactions, rather than just storing the properties of nodes, all whilst maintaining the same general structure. This enables a graph to represent a social network, the routing table for a computer network, interactions between drugs and proteins, and the interactions of particles in a detector medium.

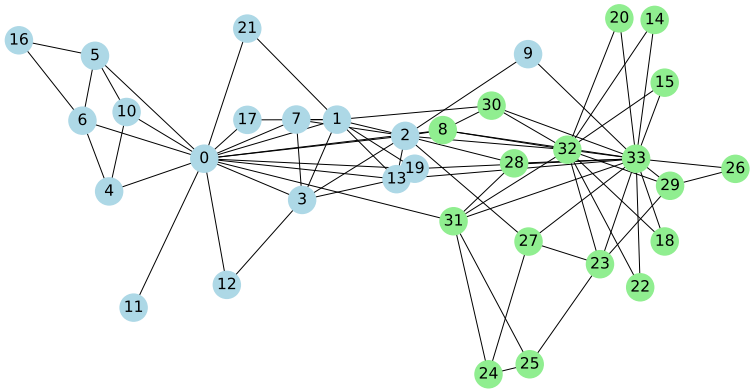


Figure 5.7: An example graph showing a social network from the Zachary Karate Club Network, which represents friendships between members of a karate club studied by Wayne W. Zachary between 1970 and 1972 [89]. Nodes represent each individual, and edges connect two individuals if they socialised outside the club. During the study, the club split into two factions, and Zachary was able to predict which nodes would fall into each faction based on the graph structure. Colours are used here to show the two factions. In the original study, and in this reproduction, node 9 is incorrectly predicted, but the rest of the members are correctly grouped with nodes 0 or 33, which represent the leaders of the two factions. Data is from [89], community detection and plotting performed in Python via NetworkX [90].

5.2.2.1 Neural Message Passing

First, the formal definition for a graph needs to be given, to give a better explanation of how deep learning techniques are applied to graphs. This formalism draws mainly from *Graph Representation Learning* by W. Hamilton [91]. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$. We can denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{E}$. In the case of an undirected graph, (u, v) is equivalent to (v, u). It is convenient to represent graphs via an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. In an adjacency matrix, the nodes of the graph

[4]There are many words for nodes and edges, including vertices and points for nodes, and links or lines for edges.

can be ordered such that every node indexes a particular row, column pair in the matrix. Edges can be represented as entries in this matrix: $\mathbf{A}[u, v] = 1$ if an edge is present, 0 otherwise. For undirected graphs, this adjacency matrix will be symmetric. As directed graphs are not used for the work in Chapter 7, they will not be covered in detail here, however it is useful for some uses of graphs that edges are directional such that a connection is only present one way.

One important property of graphs, especially in the context of deep learning, is the ability to have attributes or feature information associated with the graph. Usually, this is node-level attributes using a real-valued matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times m}$, where it is assumed that the ordering of the nodes in this matrix match the ordering of the adjacency matrix, and m relates to the number of features. It is also possible to have edge-level features, which encode the interactions between nodes, and in some cases graph-level features. These features can then be used to generate node-based embeddings. These embeddings encode the nodes as low-dimensional vectors that summarise their position, structure and features, which is useful for later learning. That is, we want to project nodes into a latent space where their geometric relations in this latent space correspond to relationships (i.e. edges) in the original graph. This is then easier to learn from, rather than the nodes and edges directly.

The defining feature of a GNN is the use of *neural message passing*, in which vector messages are exchanged between nodes and updated using neural networks [92], shown visually in Figure 5.8. This allows us to go from some graph $G = (\mathcal{V}, \mathcal{E})$, along with node features, $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times m}$ and use this to produce node-based embeddings $\mathbf{z}_u, \forall u \in \mathcal{V}$. This requires that each of our nodes have some features associated with them. For most datasets, this is not an issue, as there is a rich set of node-level features that can be applied. If this is not the case, there is the potential of using calculated features, such as node-based statistics, or a unique per-node identifier. Message-passing is applied to the GNN iteratively, and during each iteration k, a hidden embedding $\mathbf{h}_u^{(k)}$ corresponding to each node u in the graph is updated according to the aggregated information from u's graph neighbourhood $\mathcal{N}(u)$, where u's neighbourhood is defined as the subgraph of G induced by all vertices adjacent to u. This can be expressed as follows:

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{5.2.1}$$

$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right), \tag{5.2.2}$$

where UPDATE and AGGREGATE are arbitrary differential functions (i.e. they are neural networks themselves), and $\mathbf{m}_{\mathcal{N}(u)}$ is the "message" that is aggregated from u's graph neighbourhood $\mathcal{N}(u)$. The superscripts here are used to distinguish the embeddings and functions at different iterations of message passing.

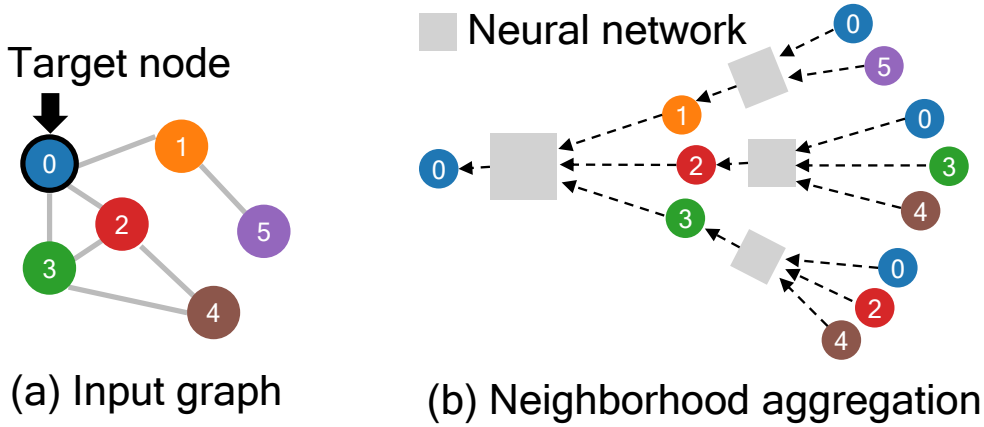


Figure 5.8: Overview of how a node aggregates information in a GNN from its nodes in its local neighbourhood. Here, information from 0's local neighbours $(1, 2, 3)$ is aggregated, then from their neighbours and so on. This figure is illustrative of a 2-iteration version of a message-passing model. It is of note that the aggregation computation graph makes a tree structure. Figure is from [93].

At each iteration k of the GNN, the AGGREGATE function is used to take as input the set of embeddings in u's neighbourhood and generate a message $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ based on this aggregated information. This is combined with the previous embedding of node u by the update function UPDATE to produce the new embedding $\mathbf{h}_u^{(k)}$. After the full K iterations of the message passing, we can use the output of the final layer to define our embeddings for each node, i.e.

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}. \tag{5.2.3}$$

This message-passing process, in its most basic form, allows nodes to aggregate data from their immediate neighbours, and further iterations of the message passing increase this aggregation, pulling in information from further and further

nodes in the graph. This can be seen as analogous to the concept of receptive field in a CNN. Every iteration of message-passing encodes information about the nodes k-hop neighbourhood, where the k-hop neighbourhood is the set of nodes at a distance of less than or equal to k from u. These node embeddings mostly encode two sorts of information: structural information about the graph itself, and feature-based embeddings about the features of all the nodes in the nodes k-hop embedding.

The actual functions and structure used differ depending on the task at hand, but in the most basic GNN framework, a simplification of the original GNN models proposed by Merkwirth and Lengauer in [94], and Scarselli et al. in [95], the AGGREGATE and UPDATE functions are given as follows:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathbf{h}_u^{(k-1)} + \mathbf{b}^{(k)}\right), \tag{5.2.4}$$

where $\mathbf{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neigh}}^{(k)} \in \mathcal{R}^{d^{(k)}\times d^{(k-1)}}$ are trainable parameter matrices and σ denotes an elementwise non-linearity, such as tanh or ReLU, and $\mathbf{b}^{(k)} \in \mathcal{R}^{d^{(k)}}$ is the bias term. This form of message passing is analogous to a standard MLP, as it relies on linear operations followed by a single applied elementwise non-linearity. First, the messages incoming from the neighbours are summed, then they are combined with the neighbourhood information from the previous embedding using a linear combination, and finally an elementwise non-linearity is applied. We can use this to define the basic GNN UPDATE and AGGREGATE functions:

$$\begin{aligned}\mathbf{m}_{\mathcal{N}(u)} &= \sum_{v\in\mathcal{N}(u)}\mathbf{h}_v \\ \text{UPDATE}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) &= \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u + \mathbf{W}_{\text{neigh}}^{(k)}\mathbf{m}_{\mathcal{N}(u)}\right),\end{aligned} \tag{5.2.5}$$

with

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \tag{5.2.6}$$

to denote the message that has been aggregated from u's graph neighbourhood. The aggregation functions used are usually much more simple, utilising functions that are used in other forms of deep learning, such as the use of pooling functions, *mean* [96], *max* [97], *sum* [98] and more. The impact of the aggregation function

can depend on the task. For example, it has been shown that *mean* and *sum* are suitable choices for tasks such as node classification, whereas max is favourable for data that is composed of many 3D points, however this is not always the case. There is no mechanism to identify the most suitable aggregation function currently, outside of empirical analysis.

After this, the learned node embeddings can be used to classify the nodes or similar, depending on the problem scope. For classification, a simple approach would be to use this learned node embedding as the input to a final MLP layer, to use the embedding per node to produce a classification per node.

5.2.2.2 Usage of GNNs

With an understanding of graphs and how a graph neural network functions, there are two important questions in the context of deep learning: What sorts of problems can be solved with a graph-based neural network, and when would a graph-based approach be used over something like a CNN.

There are several common approaches to deep learning applied to graph datasets. The first task that can be performed on graphs is node classification, the act of assigning a category to each node in a graph. In the case of the previously shown social network example (Figure 5.7), this would be predicting the faction that they fit into. Here we are predicting some label for each node, with the graph enabling the exploitation of information from nearby nodes to enable a more intelligent classification that includes information from a node's neighbours and more. A related task is the prediction of relations, edge inference. This would be the prediction of a friendship-based on node attributes in the context of a social network. This task can be more complicated than node classification and is highly dependent on the input dataset, ranging from simple heuristics giving useful results to requiring encoding of many hundreds of rules and interaction limits when being used on complex data such as biomedical knowledge graphs [99].

Both node and edge classification are lower level predictions, relating to individual components in the graph, and aim to infer missing information from the graph. Community detection, on the other hand, is a higher level task that aims to identify communities, that is clusters of nodes that are grouped by some property, in the graph. Finally, at the highest level, there are operations over the entire graph such as graph-level classification, such as identifying malicious software using a graph-based representation of its syntax and data [100], or the

prediction of a molecule's properties based on its structure [92].

The GNN can be seen as a more generalised CNN, operating over data that is less rigidly structured into a grid, but instead a graph with nodes at any position. The main remaining question is: When does utilising a GNN over other alternatives such as a CNN make sense? The most obvious initial answer is that the choice of network depends on both the input data structure and the required output for the current problem. However, in a lot of use cases, such as most applications in Physics, the data is not in any intrinsic form to begin with, such that the construction of a graph or an image may make sense. A graph may be a more efficient input for more sparse inputs where much of an image would be left empty, though there are also multiple variants of CNNs that are designed to operate over sparse inputs, such as the submanifold sparse CNN [101]. Similarly, graph-based networks can deal with variable sized inputs easily, whereas most forms of CNN and other ANN are not able to, which can result in the need for tiling images, which in turn can cause loss of performance from missing context in each tile. Combined, this makes the use of a graph-based network much easier, as it has additional flexibility in its input formation, not requiring the strict grid structure of an image, and enabling arbitrary scaling of the graph size depending on the example. Conversely, some problems may not require these features and the target problem is more intuitive when formulated based on input images and CNNs.

5.3 Deep Learning in Particle Physics

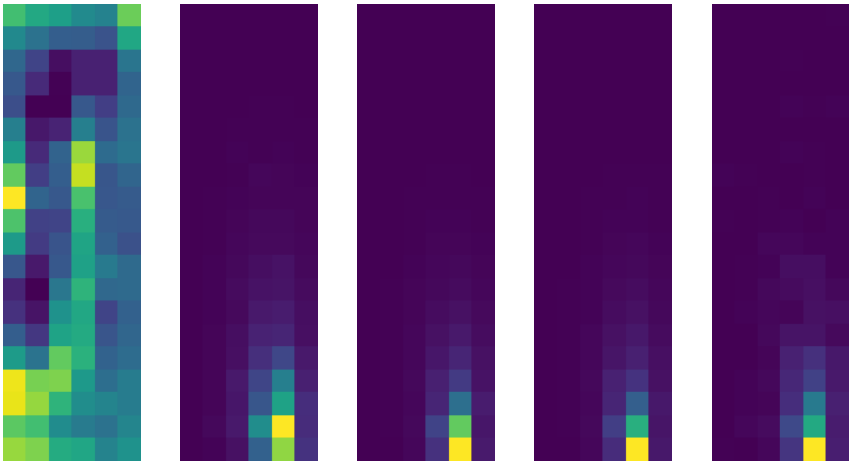
Machine learning has been in use in particle physics for decades now, but recent advances in computing, plus the use of graphics cards to accelerate deep learning[5], has led to widespread use of deep learning across the whole of the simulation, reconstruction, and analysis chain. With this increase in computation, the usage has moved from MVA based methods to deeper and deeper networks. The usage of these techniques is now widespread across most experiments, and a few of the use cases are highlighted here. However, it is important to note that particle physics must also overcome infrastructure and physics challenges with increased usage of deep learning. Deep learning in a lot of cases can be slow to process with a graphics card, meaning that computing infrastructure needs to be extended to include this additional hardware, or software improvements to aid access to shared compute infrastructure. There are also plenty of legitimate concerns around the

usage of deep learning and ensuring that the final models do not utilise features from a specific theoretical interaction model, which in turn bias physics analyses unknowingly.

- **Event Generation**: There is a desire to generate events even faster, as parts of the GENIE and GEANT4 steps can be extremely computationally expensive, which can slow down the generation of large statistics datasets. There have been investigations into if GANs could be used to generate physics events in a cheaper, faster way than conventional methods. An example of this would be CaloGAN [102], in use at ATLAS. It utilises a GAN that is trained to provide fast simulations of realistic particle showers in the calorimeters at ATLAS. A GAN has two components, a generator that attempts to produce realistic outputs to fool the second component, the discriminator. The discriminator tries to distinguish generated outputs from real outputs, in this case GEANT4 generated showers. The score generated from the discriminator is then used to improve the generator and further improve the generated outputs. Initial studies have shown that a speed-up of factor 10^5 could be achieved, compared to GEANT4 [102]. Similar studies have been performed at DUNE through the use of a model-assisted GAN (MAGAN) [103], in the context of the DUNE PDS. Here, the generated output is a proposed alternative to the memory-intensive lookup libraries used, which also take large amounts of time to produce. The MAGAN can produce 1 million samples in less than 1 minute, compared to the multiple days the simulation software takes [104]. Figure 5.9 shows an example of how the MAGAN can learn to reproduce the look-up library, and how its performance improves as it learns.

- **Signal Processing**: Deep learning has also been used in various ways for the task of signal processing. Wire-Cell has used deep learning to augment their signal processing through the use of a CNN [105]. Here, convolutions and pooling is applied to the input raw waveforms, before eventually scaling the input back through the use of up sampling steps to tag regions of interest in the original input image. Wire-Cell was able to show that inclusion of multi-plane information, combined with a deep learning-based algorithm,

[5]The architecture of GPUs means they can process the sort of maths used in neural networks many hundreds of times faster than even a fast, modern CPU. This has enabled networks to become even bigger.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 5.9: Example of MAGAN being used to mimic the PD look-up library. This shows arbitrary training examples from various points in the training process. **5.9a** shows an emulated image after one iteration, **5.9b** after 1,000 iterations, **5.9c** after 10,000, and finally **5.9d** after 17,000 iterations. **5.9e** shows the truth from the light map photon library. Figures taken from [104].

can offer significant improvements over traditional methods. A different usage of deep learning for signal processing is the use to fill in gaps in the detector. Due to age, hardware failures or manufacturing defects, parts of a particle detector can fail over time. In a LArTPC, this could result in dead channels in an APA, which in turn leads to gaps in the raw waveforms of the detector where no signal is found. Deep learning is being tested to help fill in this missing data, which in turn makes reconstruction easier as well as downstream analysis. This has been tested at MicroBooNE [106], as well as initial testing at DUNE.

- **Event Reconstruction**: As mentioned in Chapter 4, event reconstruction can benefit greatly from the use of deep learning. Here, deep learning is used to perform the critical tasks in reconstruction, such as clustering or vertexing. This has been used across multiple experiments, for example the Exa.TrkX group have applied graph neural networks to track particles in a LHC-like detector, as a step towards using it on ATLAS or CMS data [107]. On the other side of things, MicroBooNE has used a CNN to perform pixel level particle identification of its LArTPC data. This approach classifies the individual pixels in an event as instances of individual particle types. Finally, as mentioned previously, there is deep learning-based approaches

to the full reconstruction problem in use at the DUNE ND, which uses a variety of network types, including both CNNs and GNNs to build up a full understanding of the event, with outputs from one network becoming the inputs or part of the input for subsequent networks.

- **Event Classification**: One of the most common problems that deep learning is applied to is classifying events. This involves using the output of the detector as the input for a deep learning algorithm, to identify a given class of events. This may be a simple binary classification of signal versus background, or go into further detail such as sub-categories of signal to include particle flavours or more. There is a rich history of using machine learning for event classification, both D0 and CDF utilised NNs and BDTs to classify events [108, 109]. More recently, NOvA used deep learning to classify neutrino events [24], with the CVN also being applied at DUNE, as mentioned previously [69]. In the CVN, a CNN is used to identify interactions in the DUNE FD, to select electron or muon neutrino charged-current interactions. This was able to achieve both high-efficiency and high purity, achieving 95% and 90% efficiency for electron and muon neutrinos respectively, when the reconstructed neutrino energy was between 2 GeV and 5 GeV for ν_e and above 2 GeV for ν_μ. This does not rely on any reconstruction, instead taking the hits after noise filtering and deconvolution. The three views of the LArTPC are processed independently at first, before being combined once sufficiently processed through convolutional layers, to be convolved again before predicting the class of the input event.

A more comprehensive review of deep learning in use in particle physics is outlined in [110]. It is a rapidly growing technique used in particle physics, and new applications are found frequently.

6

3D Track Reconstruction in Pandora

Professor Farnsworth: *Pandora.*
Leela: *That dangerous, 3D planet? Can't we just send our avatars?*
Professor Farnsworth: *No! It's cheaper to just have you die.*

Futurama - S06E08 "Law And Order"

Reconstructing LArTPC events in 3D is a crucial part of the reconstruction workflow. The 3D representation of an event contains the information necessary to reconstruct particle trajectories and interpret calorimetric data, enabling measurement of high-level quantities such as the particle type and momentum. As the native output of most LArTPCs is 2D, the hits across the multiple views of the detector need to be matched together, such that a 3D hit can be created. DUNE will have 3 wire planes, resulting in 3 2D outputs to match hits across, with the 3 outputs meaning there is an amount of redundancy at this step. An example muon neutrino event is shown in Figure 6.1, showing the common coordinate system, and how the same event can vary across each of the different planes of the detector.

In Pandora, the 3D hit creation comes after the initial 2D clustering, with matches being made between these clusters. With these matches made, the next challenge and the problem addressed in this chapter is matching the 2D hits, to finally produce corresponding 3D hits. This is a crucial part of the reconstruction chain, with 3D hits forming the basis of both track fitting and shower characterisation, used extensively in the final analyses.

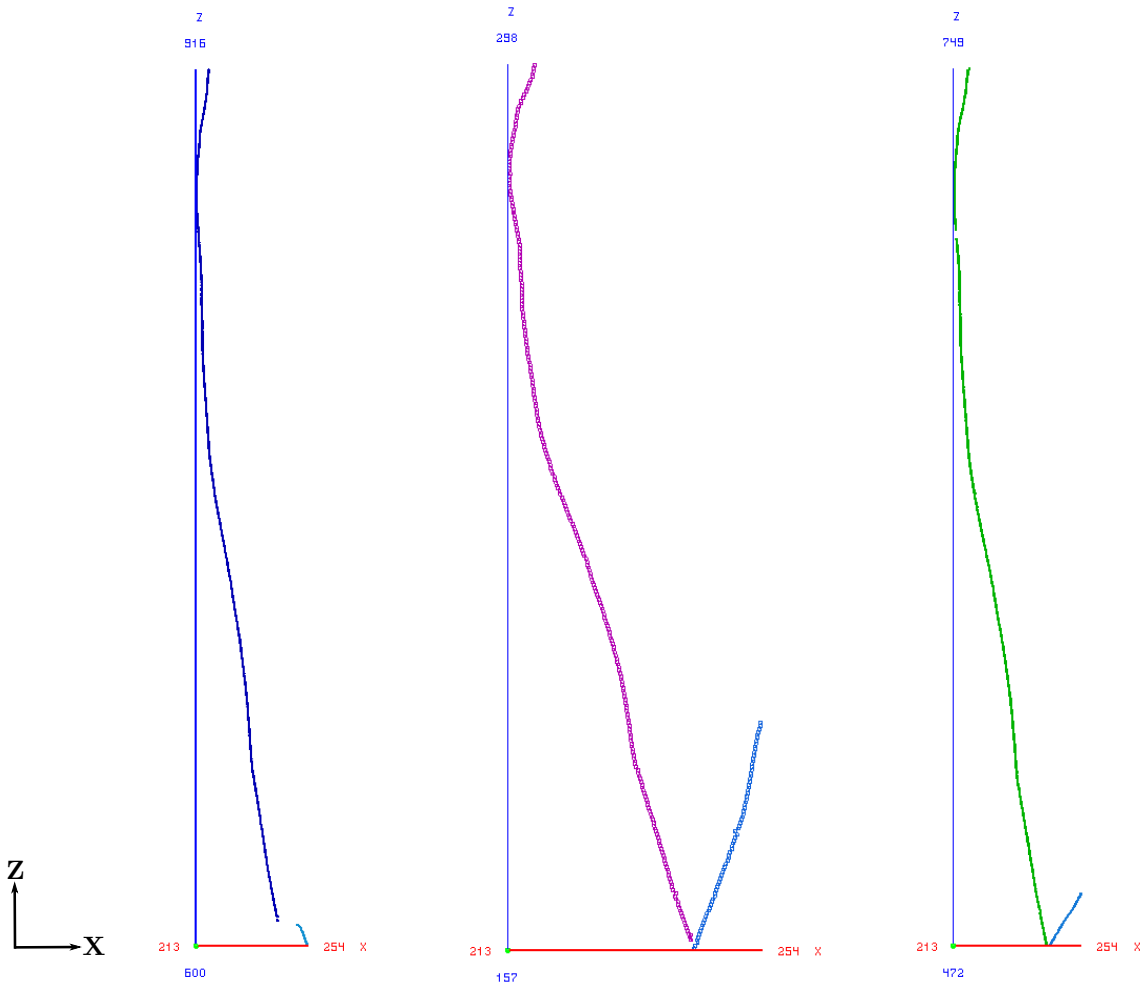
This chapter will outline the workflow that Pandora uses to match hits across multiple LArTPC views, including the issues encountered with the previous implementation, as well as the improvements to this workflow that were implemented as part of this thesis. This will be followed by an explanation of the performance improvements, including event displays. This work was specifically targeted at track-like topologies, but the broad approach outlined and subsequent improvements can be extended to showering particles.

6.1 3D Hit Creation Workflow

The creation of 3D hits in Pandora occurs approximately halfway through the reconstruction workflow, after there is already a 3D vertex and a 2D reconstruction of the event in each view is fairly sophisticated. The main concern of the 3D hit creation is to match the individual 2D hits in each view with their corresponding positions in the other two views to create a set of 3D coordinates. The 2D clustering in each view provides detailed information that can be used to match up trajectories and feature points between the views, enabling 3D reconstruction to take place. It is important that the 3D hit creation accounts for those cases where a hit is obscured or missing in the other views. Missing hits can happen due to detector defects, such as dead or unresponsive readout channels, or due to the varying orientations of the readout planes making a hit difficult to see if obscured by another feature of the event. Once a hit has been matched with its corresponding position or positions in the other views, the hit creation process uses the information to create a candidate 3D space point. A chi-squared metric is also calculated for each 3D position that can be used to resolve ambiguities and enforce quality standards.

Geometric 2D-3D Matching

First, an understanding of the coordinate system in use at a LArTPC is useful. The 2D hits, those that sit on a single plane of the detector readout, are encoded with an X coordinate, and a U, V, W coordinate, depending on the view. However, the critical feature of the LArTPC readout is that the X coordinate, which encodes time, is consistent across all three views, whereas the UVW coordinate, which encodes the wire coordinate, is not. That means that for a given event being



(a) U View **(b)** V View **(c)** W View

Figure 6.1: An example ν_μ event, showing each of the three detector views. It can be seen from the axis ranges that the X axis is common across each of the three figures, whilst the Z axis is not. It is also interesting to see the specific features that are visible due to the orientation differences of each of the views. For example, the short track coming out of the interaction point in the U view is going backwards and disjoint, unlike the other two views. Similarly, the muon track path varies between the views, and contains some gaps in the W view.

looked at in three views, the X coordinate will be consistent, assuming a valid $T0$ is known, which is the event start time. These 2D coordinates relate back to the true YZ coordinates with use of the wire angles θ_U, θ_V. For example, to calculate the Z position from UV positions,

$$Z = ((U * \sin(\theta_V) - V * \sin(\theta_U))/\sin(\theta_V - \theta_U)) \tag{6.1.1}$$

This feature of the X coordinate plays a key role in 3D matching, as it can be relied upon across each of the views, unlike the second coordinate. However, this reliance also means that events with certain topologies that make the X coordinate less reliable are more difficult to reconstruct. For example, in an electromagnetic shower, with its large tree-like structure that leaves deposits across many wires, it is not effective to use the X coordinate. Similarly, "isochronous" tracks, tracks that contain sections that arrive at the same time, are more difficult to reconstruct as they have large sections with the same X coordinate, making it much more ambiguous to match those parts.

There are a few general techniques used throughout the 3D hit creation process, which help match hits across the detector views. An explanation of these techniques follows, such that the algorithms and tools[1] which build upon these techniques can be described more easily later.

The first technique is the process of projecting hits, which is a critical part of Pandora that allows it to exploit the detector knowledge that Pandora has to take a hit in one view, and estimate where that hit would be in a different view. This process can also be extended to take a 3D hit and project it back into 2D, which is a useful for comparing a reconstructed 3D hit to its underlying 2D hits that were used to build it. This hit projection technique uses the geometry information that Pandora has from its loaded geometry files to allow accurate mapping of one position to another. The actual process for this is as follows, for projecting a 3D position into a specific 2D view:

- The X coordinate can be treated as the same as the input 3D position, such that the final output 2D hit has the same X coordinate as the input 3D hit. This is because the X coordinate is the same across all three views and is the single consistent coordinate, which is a very useful feature for the position

[1]Algorithm and tools have specific meaning in Pandora. A tool is a subcomponent of an algorithm, with one algorithm potentially utilising multiple tools to complete its task.

matching tools later on. Therefore, there exists a class of 2D-3D matching tools that consider slices in X, and match the U, V, W coordinates in each slice.

- The Z coordinate is where an understanding of the detector is required. Pandora utilises a transformation plugin that loads various properties about the detector, such as the wire angles used for each of the three views and, then commonly used calculations with those angles, such as $\sin, \cos$. This detector information can be combined with the 3D positions Y and Z coordinate to produce a final new 2D Z position. A similar approach is used for various other transformations between 2D and 3D hits, as well as taking multiple hits and combining them consistently.

- In the 2D-3D matching case, when projecting matching 2D hits into 3D, two broad approaches are used, split by how they match features of the track. First, there are "transverse" tools, those which match up tracks with an overlap in X, checking if some or all of their trajectories align in all three views. Secondly, there are "longitudinal" tools, which matches features of the tracks such as their start and end, using them to produce a common 3D position. We can then extrapolate between these known 3D positions to find additional matches.

A similar process can be used to estimate a position in a different 2D view for a given 2D hit. The ability to move quickly between views is useful for two main techniques. The first is that it allows a secondary view to be used, which means that the hits and any other related data such as fits built on those hits can now be used with a hit from a different view. This process is used to power most of the 3D hit creation tools, to find a matching position in a different view for later use. Following that, another useful technique involving projecting hits between views is that it allows stronger constraints to be applied to potential hits. A created 3D hit should lie near its base 2D hit and matched positions, such that projecting the 3D hit back into 2D should result in a small distance between the two. If a projected hit ends up far away from a 2D hit, that can imply the 3D hit is not a suitable match. Finally, projecting hits between views can help in cases where there are ambiguities in one view that a different view could resolve, due to the difference in readout plane angle between the three views. It should be noted that the problem here is over constrained, only two views are needed to actually

produce a final 3D hit. However, the presence of a third view means we have both some additional redundancy and an extra consistency check. Considered in terms of the underlying wires, it is always possible to find the associated wire crossing with only 2 views, but without a third view, there are no additional constraints or consistency, which makes the matching less reliable.

One impact of having views at differing angles to each other is that a single hit in one view, may match to many hits in a second view, if the track that produces those hits is at a large angle relative to one set of wires, but a small angle relative to the other set. This leads to the second key technique, the concept of a sliding fit. A sliding fit is a general concept, used throughout statistics and many forms of pattern recognition tasks, being particularly useful here to allow a decoupling from the discrete hits of the input, to a single continuous set which we can interpolate, extrapolate or project as needed. This means the potential discrepancy in the number of hits between views can be mostly ignored.

For tracks, the most appropriate form of this fit is a sliding linear fit. Here, a sliding linear fit refers to a fit over some input 2D hits, with the resulting fit unlocking a number of techniques. The 'sliding' part of a sliding linear fit refers to a sliding 'window' that moves across the input, averaging out the values that lie inside it. This allows a sliding fit to average out values, with the strength of this averaging effect controlled by the size of the window. This window can also be extended outside the fit, that is, move it entirely to one end of the fit, such that hits that lie outside the fit can be evaluated for how consistent they are with the fit itself. The fit itself in this case is a linear fit, containing two axis directions, built based on the axes of a principle component analysis (PCA) fit, as well as additional parameters corresponding to axis properties, such as the direction and intercept. With these general axes calculated, a sliding fit over the input hits can be performed, to calculate individual layers of the fit that are the same width as the sliding fit window, with these layers further combined into segments. This sub-structure in the sliding linear fit is put to use when comparing other hits that lie outside the fit, as well as finding the surrounding layers around an input position.

A sliding linear fit is also useful for 3D hit matching for a variety of reasons. The most obvious reason is that it allows a slight smoothing of the hits in a given view, which is useful to avoid the small local features in an event, if they have been caused by reconstruction issues, or similar. However, with a sufficiently small window size, local features can be followed if desired. An example of the

impact of the sliding window size on a 3D trajectory is shown in Figure 6.2. For most of the uses of a sliding linear fit, the first step is to move the current input hit into the coordinate system of the fit, which requires projecting the hit onto the axes of the fit, such that comparisons against the fit can then be performed. Once this is performed, a matching position in the fit can be located and used for the current tool.

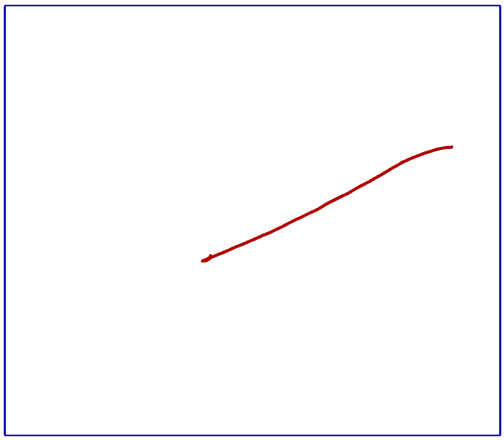
A similar process can be used to extend the concept of a 2D sliding linear fit into 3D, including running independent 2D fits on the various 2D planes for the 3D hits. This then unlocks similar benefits to the 2D fits, for further work in Pandora that utilises the 3D hits made during this process, as well as incremental improvements to the hits created during this step as a refinement procedure after the main hit creation has run.

Matching Tools

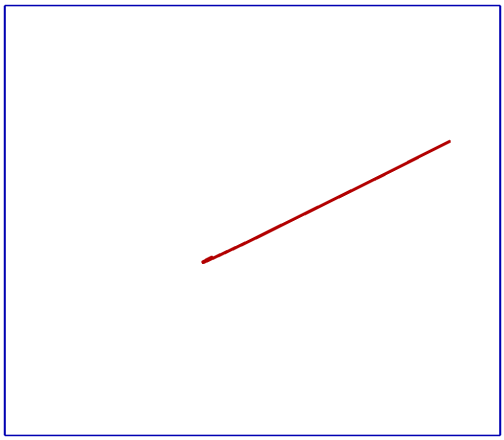
A suite of 2D-3D matching tools have been developed for the 3D hit creation workflow, exploiting geometric matching techniques, both "longitudinal" and"transverse", described above. Each tool takes as its input the list of 2D hits that do not yet have a 3D coordinate, and then searches for matches across the views to create a 3D hit. Each tool takes a different approach to account for the range of different track topologies and orientations that are found in neutrino events. This suite of tools is run sequentially, in a well-defined order, with hits that fail to find a match with one tool being passed to the next tool in the chain, until a hit is found, or every tool has been used. The 3D hits are all produced in isolation, with the resulting 3D representation of the event not being used to steer or fine-tune any of the matching tools.

The tools can access common information about the event, including the 2D clustering in each image. Before any tool runs, a 2D sliding linear fit is created for each 2D cluster, using the same sliding window size for each cluster and view. The complete list of these 2D sliding linear fits is made available for any of the tools that require it. Additionally, some tools utilise 3D start and end positions in their algorithms, which are calculated from the sets of 2D sliding linear fits by taking their start and end positions and matching these positions across the different views.

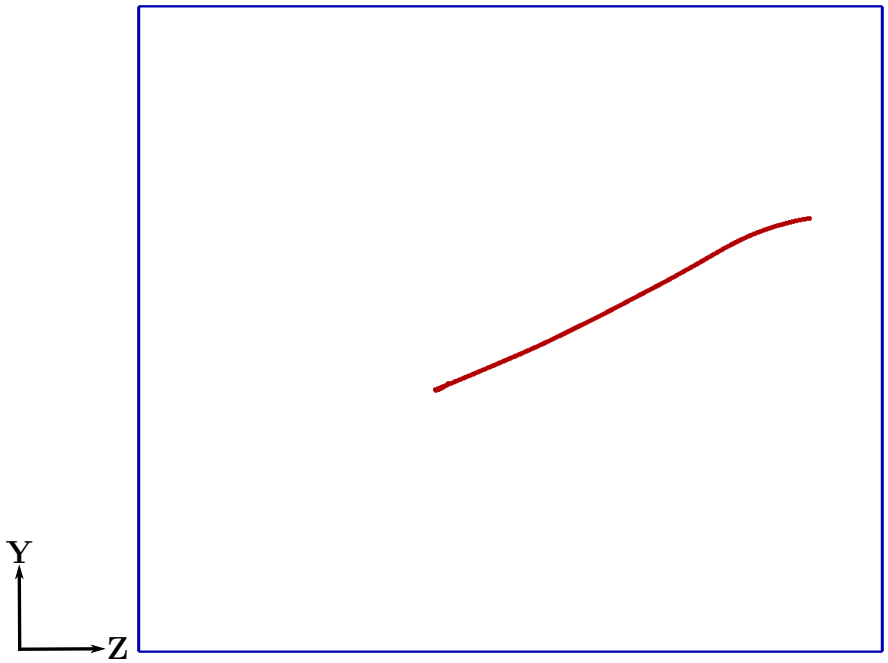
The current tools are outlined here, where each has a specific track topology they target, as well as a method of performing the matching.



(a) Window Size 10



(b) Window Size 1000



(c) Window Size 100

Figure 6.2: The impact of the sliding fit window size on a 3D ν_μ event. Figure 6.2a shows the impact of a very small window size, with some shaking and inclusion of individual hit level deviations, whereas Figure 6.2b shows the opposite case where the sliding fit window size is so large, it removes the important features from the event. Finally, Figure 6.2c shows a more sensible window size, that smooths, whilst maintaining the large features of the track.

- `LArClearTransverseTrackHits`: In this context, a transverse track is a track whose hits are primarily across the detector, that is, the hits mostly lie in the X plane. The clear transverse track hits tool is optimised to find hits that lie in this X plane. It does this by matching up the spatial coordinates at common X positions in each of the other views. This is achieved with the following calculation:

$$R = \vec{V}(\alpha) + \vec{D}(\alpha) * \frac{x - \vec{V_x}(\alpha)}{\vec{D_x}(\alpha)},$$

  where $\vec{V}$ and $\vec{D}$ refer to the fit vertex and fit direction respectively, R is the extrapolated position, and the α term is used to pick the most suitable vertex and direction from the fit. That is, depending on where the input point lies, it may be closer to one end of the fit or the other, so the most suitable end of the fit is chosen. If this step works as expected, then an extrapolated position in each of the other two views can be used later. Failures can occur here, mostly if the original fits in the other views are inconsistent with the input calorimetric position, such that the input X coordinate cannot be found, or it lies outside the fits in the other views. Both of these cases could be caused by earlier reconstruction errors, such as clustering hits into the wrong particle. In the successful cases, a match position is found in the other detector views, which can then be passed to the next stage of the process. Due to the design of the extrapolation process, this tool only runs on 2D clusters that are predominantly single-valued in X, such that a slice can be made on the X coordinate to then match across the views. If there are ambiguities in the X coordinate, instead, the later `LArMultiValuedTransverseTrackHits` is a more optimal choice to use. The 2D hits that are not reconstructed unambiguously by this tool then fall-through to the following tool.

- `LArClearLongitudinalTrackHits`: Conversely, a longitudinal track is a track whose hits primarily lie lengthwise down the detector, that is the Z plane. Unlike the `LArClearTransverseTrackHits` tool, this tool is optimised for clusters that have multiple hits at the same X position. To achieve this, this tool uses the reconstructed 3D start and end points of the track as the basis of the 3D hit creation. By interpolating between these two 3D points, a 3D hit can be made. To start this process, first each of the 2D

hits must be mapped onto the line between the 3D start and end points, by calculating `fraction` and `projection3D` values.

$$\texttt{fraction} = \frac{(\vec{E}_{\text{2D}} - \vec{V}_{\text{2D}}) \cdot (\vec{C} - \vec{V}_{\text{2D}})}{|\vec{E}_{\text{2D}} - \vec{V}_{\text{2D}}|^2}$$
$$\texttt{projection3D} = (\vec{V}_{\text{3D}} + (\vec{E}_{\text{3D}} - \vec{V}_{\text{3D}}) * \texttt{fraction}),$$

where $\vec{E}, \vec{V}$ and $\vec{C}$ are the end point of the track, the reconstructed vertex and original calorimetric hit respectively and, $_{\text{3D}}$ and $_{\text{2D}}$ is used to differentiate the original 3D vertex and end point and the 2D projections of them. The original calorimetric hit has no 3D representation, as that is what we are using this projection to find.

The calculated `projection3D` represents the position on the line between the 3D start and end points that is closest to the original 2D hit. This position can then be tuned by projecting this line-based position back into each of the 2D views, obtaining 3 2D positions. These 3 positions can finally be combined to obtain an improved 3D position for the input 2D hit.

- `LArMultiValuedLongitudinalTrackHits`: The multivalued longitudinal tool is heavily based on the clear longitudinal track tool, with additional relaxation on the initial matching process. This means that a match can be found more easily, at the expense of some accuracy. The initial 3D projection of the 2D calorimetric hit is identical to that mentioned previously, but the usage of that projected 3D hit in the fit is achieved using a projection into the fit's coordinate space, rather than through the use of the layer-wise interpolation function. This means that there are fewer constraints on the final calculated position, as it is missing the interpolation of the clear longitudinal track tool.

- `LArMultiValuedTransverseTrackHits`: Similar to the multivalued longitudinal tool, this tool is a less restrained version of the initial clear transverse track tool. Unlike the clear version of this tool, this uses a method much closer to the multivalued longitudinal tool, where matching layers are found in the fits in the other views. Here, the ambiguous sections of the track, where the X value is not single-valued, are split up, such that

each segment should now be single-valued in X again. Once this split has been made in all three views, the process can proceed similarly as in the `LArClearTransverseTrackHits` tool on each segment. The biggest failure mode here is when the wrong segments are matched across views, such that disjoint segments across views, but with matching values in X are combined, resulting in 3D hits that are inconsistent, as they were produced with hits that do not match.

These 4 tools can run in two modes, one that requires a match be found in both the other views, in the 3 view case at DUNE, or a 2 view mode where only a single matching hit position needs to be found. Crucially, this is just an option, not a requirement, so in cases where a matching position can be found in both views, that is used. However, in the few cases where a matching position can only be found in one view, then the tools must be running in the 2 view mode, as otherwise the matches are thrown out. Running with only two views, whilst functional, is not optimal as there is a lack of a consistency check, such that the chance of a hit being made wrong is much higher.

As there is a difference in the performance of the matching between the tools, Pandora runs the tools in a fixed order, to allow the best tools with the strongest constraints to run first:

1. `LArClearTransverseTrackHits`.
2. `LArClearLongitudinalTrackHits`.
3. `LArMultiValuedLongitudinalTrackHits`.
4. `LArMultiValuedTransverseTrackHits`.
5. `LArClearTransverseTrackHits`, with 2 views.
6. `LArClearLongitudinalTrackHits`, with 2 views.
7. `LArMultiValuedLongitudinalTrackHits`, with 2 views.

By running in this order, we produce 3D hits with the most performant algorithm first, before less and less constrained tools run to fill in potentially difficult points in the tracks where hit matching is more difficult. Each tool in the chain 'consumes' hits, such that the follow-up tools do not have to consider those hits when they run. It is for this reason that the 2 view versions of the tool run last, as they are only a fallback if the 3 view versions failed.

Creating a 3D Hit

Each of the tools outlined returns a list of matched 2D hits, containing the input 2D hit and one or two matched positions from the other two views. The next step is to actually merge this information to create a 3D space point that can then be associated with the input 2D hit. The actual hit creation itself can now be performed. This can proceed in one of two ways, using either two or three positions, depending on if a hit was matched in just one view, or two.

- When a matching position has only been found in a single view, a simple merge can be performed between the input calorimetric hit, and the calculated position from a different view. This merge takes into consideration the views the initial hit and the chosen position are found on, and then calculates an average X position based on these two input values, as well as an average Z component which is used alongside the two input Z coordinates. These average positions can then be used alongside the existing view projection tools to project into the missing Y coordinate, using a mixture of the averaged and input Z coordinates. This gives an output 3D coordinate, which has an associated score, internally referred to as a pseudo-chi-squared value, that is set to:

$$\frac{(\text{Input Calorimetric Hit}_x - \text{Averaged Hit}_x)^2}{\sigma_{\text{x}}^2} \tag{6.1.2}$$

  where $_x$ refers to the X component of the 3D position, and σ_{x}^2 is a scaling term that is set to 1 cm by default, but can vary when required. This mode has fewer constraints on it, due to the lack of a second matching position.

- When instead a position has been found in both views, a similar process is performed as the single match case, but with additional constraints due to the additional position being used. Here, rather than simply merging the two positions, a minimisation of the calculated YZ chi-squared needs to occur. This is achieved by taking the expression for the chi-squared and differentiating it with relation to both Y and Z. These two results can be set equal to zero and solved simultaneously. This step was performed in Mathematica [111], with the result copied into the Pandora source. This results in a properly minimised YZ, and a chi-squared value that can be

used in the follow-up steps, as well as the best values for both Y and Z to use in the 3D hit, which can be combined with the X term calculated in the same way as the two match case. Like the single match case, this chi-squared term can be combined with a score of how close each of the input calorimetric and fit-based hits are, to give a final overall chi-squared for the hit.

This whole process is run over every position for each view, if there are multiple matching positions in the other views. In reality, the only tool that outputs multiple matches is the `LArClearTransverseTrackHits` tool, as the rest of the tools only produce a single match candidate.

Chi-squared checks on 3D hits

The produced chi-squared values are used for a few different comparisons. Firstly, if there are multiple possible matches in the other views for a given calorimetric hit, then the chi-squared value is used to compare between the options, with the created 3D hit with the lowest chi-squared value being chosen. Secondly, there is a higher level cut that is used after picking each hit, to ensure that each hit is of a certain quality by not allowing hits over a certain chi-squared value, defaulting to 1.0. This ensures that hits that are too far from their input 2D hits can be removed, even if they are better than the other options.

6.2 Limitations

The current implementation of the 3D hit creation algorithm has several limitations, that will be outlined in this section. These limitations, from missing elements that are simple to add, to flaws in the overall procedure which require in-depth changes to the 3D reconstruction workflow. It should be noted that in general the limitations do not impact the majority of tracks, rather they occur in the more difficult event topologies, where 2D-3D hit-matching is difficult, or involves a large amount of ambiguity between different 3D candidates. For example, isochronous tracks are a common problem in 2D-3D matching, as the wire readout is much more ambiguous, leading to errors occurring during matching much more readily.

No Containment Constraints

A simple issue arises due to the original form of the chi-squared used in the hit creation. The original form did not consider the containment of the proposed 3D hit, such that it was possible for hits to fall outside the instrumented detector volume, but still be considered. In drastic cases, this could result in hits being meters outside the detector if an isochronous track or particularly difficult shower was encountered. Having a proposed hit fall on or just outside the detector's instrumented volume is acceptable, such that they should not be out right dismissed, but missing an entire component from the chi-squared means that unfeasible proposed hits could be chosen, rather than later removed with cuts or by picking a hit with a lower chi-squared value. An example of an event which has hits that lie outside the detector volume can be seen in Figure 6.3.

Rigid Algorithm Chain

A more fundamental limitation with the existing 3D hit creation workflow is the strict pipeline the hit creation follows. The various tools designed to match hits are designed around specific topologies of tracks, but the hit creation workflow does not consider this whilst producing hits. Instead, the workflow runs the tools in a fixed order, irrespective of the actual topology of the track. This means that the transverse track hits tool is run first every single time, even if the longitudinal track tool may make more sense for the current track. As each algorithm consumes hits that it uses, this means that a follow-up algorithm can never produce its full interpretation of the 3D reconstruction, instead only able to run over the remaining hits from the step or steps before it. This means that it is possible for hits to be reconstructed by a less effective tool, and never get the opportunity to instead be reconstructed by a more appropriate tool. This is all because the tools were ordered based on their estimated performance, rather than allowing for variance depending on the current particle topology.

Reconstruction Failures

There is one advantage of having a strict pipeline for the production of 3D hits however, where less constrained tools can be ordered to run as the final tools, essentially running them only when they are required. This means that the tools that have the least number of constraints can be set to run only on the remaining

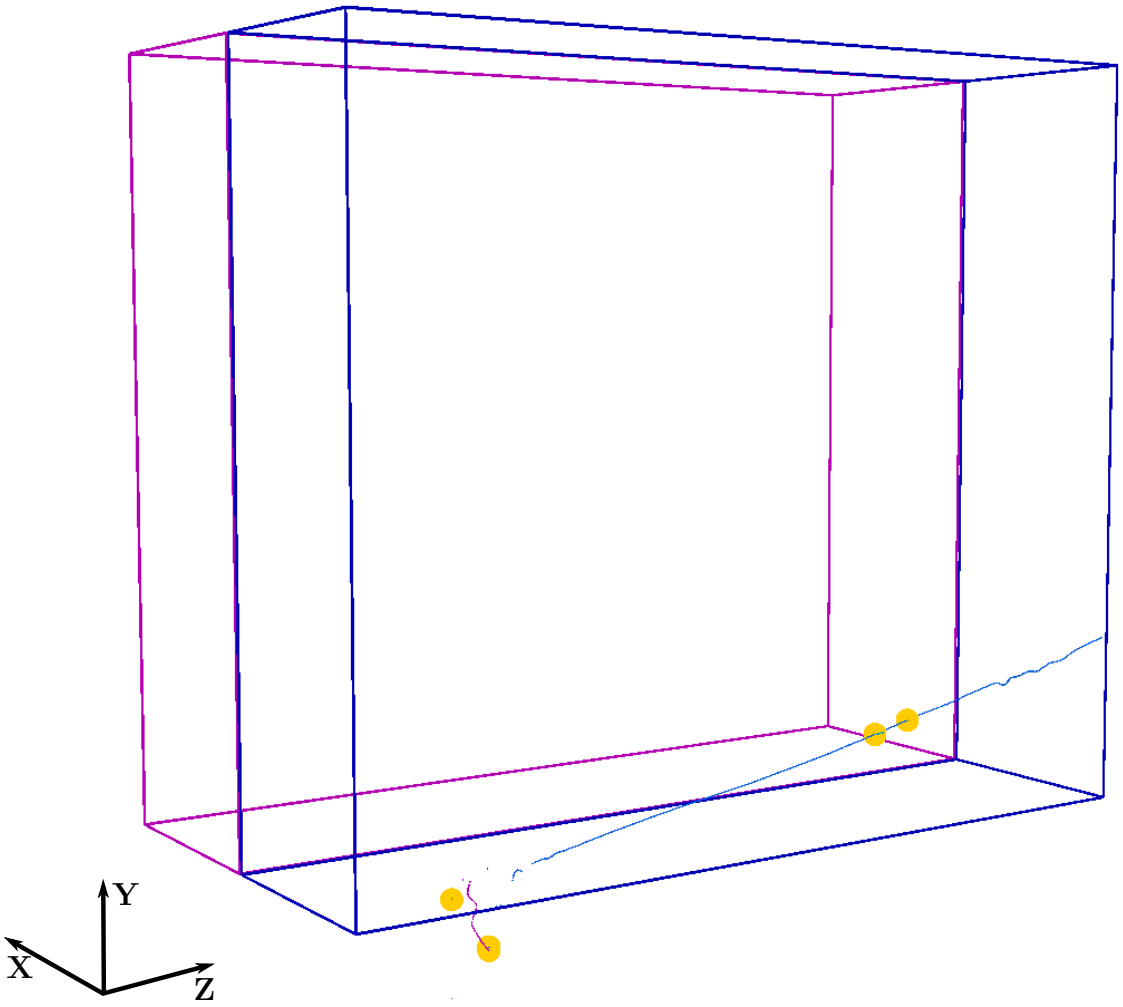
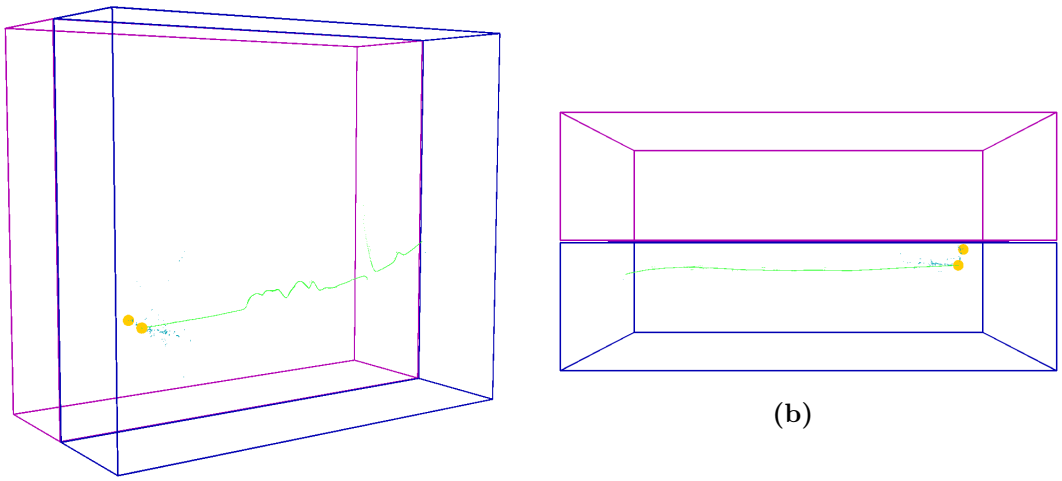


Figure 6.3: An example of a 3D containment issue in the DUNE FD. Here, an interaction has occurred close to the bottom of the detector, and a particle has been reconstructed outside the detector, with a broken particle trajectory. The yellow dots show interaction vertices, whilst the colours of the particles are used to represent different particles.

hits after all the other tools have run. In the majority of cases, this is a good thing, such that the less constrained tools are running only on small numbers of hits, and filling in the tiny gaps that are left for the most awkward hits. This is desirable, so the tools are not running over the full set which could cause unphysical results to occur due to the lower constraints on the hit matching, rather just helping the small cases where every other tool failed. However, the issue with these tools arises when the previous tools all failed. Running with fewer restrictions over the full set of input 2D hits means that the tools are likely to produce very unphysical results, leading to very broken outputs. A few examples of this are shown in Figures 6.4 and 6.5. These mostly show isochronous tracks that have failed many of the initial 3D hit matching tools, instead of falling back to more suitable tools, leading to poorly matched 3D hits. This results in an inconsistent and unphysical 3D hit reconstruction. If only part of the tracks are difficult to match, small regions can be reconstructed with less constrained and less effective algorithms, leading to wavy / bumpy tracks, due to the smoothing between multiple tools. If the whole track is difficult to match, the entire 3D reconstruction can be incorrect. It should be noted that if viewed from the right orientation, the events do look sensible, due to the constraints on the X coordinate from the original input hits.

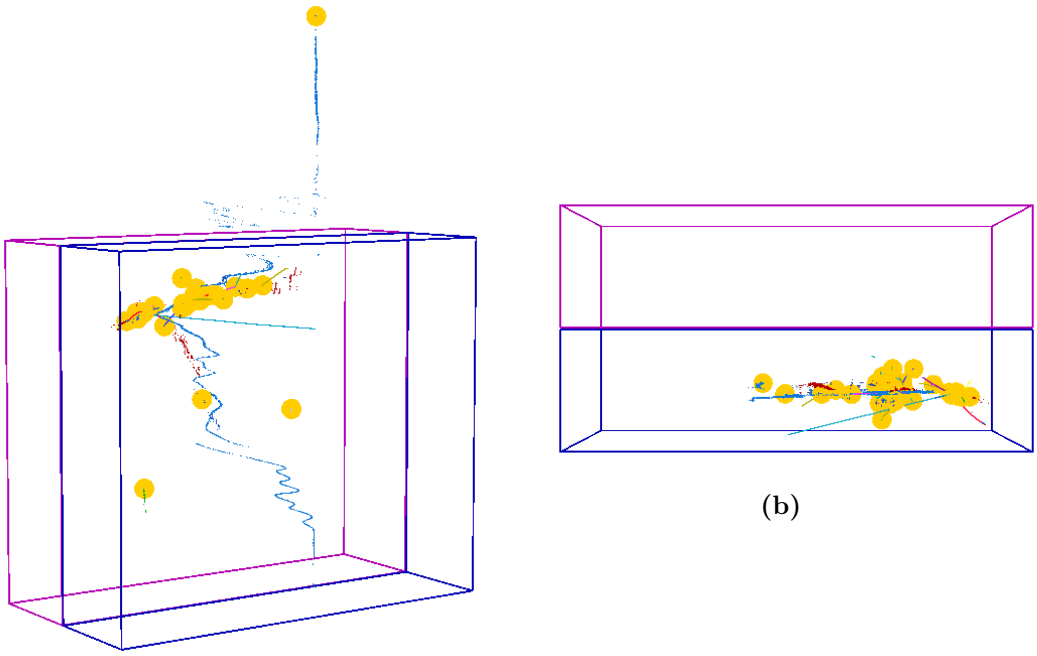
Under-utilisation of 3D Event Information

Finally, a minor issue is how marginal areas of missing hits are dealt with. The 2D-3D hit matching is performed for each hit independently, such that the emerging 3D representation of the event is not used to guide the further reconstruction. It is possible for all the algorithms to run, but to still leave gaps in the 3D hits, that could instead be filled if the rest of the newly produced 3D hits were used. These gaps should be simple to interpolate over, but no such step is performed, commonly resulting in tracks with a small gap in 3D which does not exist in 2D. If instead a 3D interpolation step or similar was performed, to allow the newly generated 3D hits to inform the 3D hit creation step, these small gaps could be avoided. Special care does need to be taken as it is possible to have legitimate gaps in the particle trajectories, either due to gaps in the LArTPC geometry, or in some cases due to detector defects (for example MicroBooNE has regions of dead wires due to detector age and other impacts). Interpolation over these areas would need to be avoided, whilst detecting and interpolating over legitimate missing regions, using the new 3D hits and existing 2D hits to anchor and guide the interpolation.



(a) **(b)**

Figure 6.4: An example of 3D smoothing issues in the DUNE FD due to the 3D reconstruction algorithm pipeline. Figure 6.4a shows an event where a long muon track has been reconstructed with two different algorithms that did not produce consistent outputs. The result is bumps produced where the two outputs are smoothed between. Here, the reconstruction failures are likely due to the isochronous nature of the track at points, as well as small delta rays making any matching harder. Figure 6.4b shows the same event looking down on the detector, where it can be seen the 3D reconstruction is consistent in this view, due to restraints in the production algorithms.



(a) **(b)**

Figure 6.5: An example of a 3D reconstruction failure in the DUNE FD due to failures in all the 3D reconstruction algorithms. Figure 6.5a shows an event where a track-like particle has been reconstructed with multiple issues, including unphysical tracks and hits outside the detector. This specific issue is due to earlier issues in the reconstruction chain leading to merges between multiple particles, causing matching issues at the 3D reconstruction stage. Similar to Figure 6.4, Figure 6.5b shows the same event from a top-down view, showing the event is consistent when looking from above, due to constraints built into the 3D reconstruction algorithms.

The strength of the interpolation must also be carefully considered, to avoid cases where an entire track is interpolated from a small number of input hits.

6.3 An Improved Approach to 3D Hit Creation

As part of this thesis, a suite of targeted improvements were implemented, to improve and extend the 3D hit creation algorithm, with the aim of addressing the limitations described above. The goal was to develop a modified workflow that would return more consistent 3D tracks, and consider the boundaries of the detector. Parts of this work target 3D tracks specifically, as shown previously they can have very unphysical representations in the 3D reconstruction, but extensions are outlined that would be required to allow this to work for shower-like tagged particles as well. The reasoning for specifically targeting 3D tracks to start with, is to provide a firm basis for particle tracking and trajectory-fitting in DUNE, which relies critically on a precision reconstruction of 3D hits. In contrast, the use of 3D hit information for shower characterisation, Michel electron tagging, energy reconstruction and more, is not as prevalent currently, although this will become more important for DUNE physics goals in the future.

6.3.1 Inclusion of Containment Constraint

The simplest change was one to update the existing chi-squared score, associated with every 3D hit, to include a penalty term that depends on the 3D containment, rather than just the X coordinate of the input 2D hits. Updating this to include a term based on the maximum distance outside the detector makes 3D hits that fall outside the detector both less likely to be chosen compared to other 3D hits that are reconstructed inside the detector volume, but also means hits that are produced outside the detector volume by a large amount are not allowed, as they will fail the cut on the chi-squared quality that is applied to every candidate hit. This is a small change that ends up improving events significantly, stopping the most obvious issues in an event. This improvement was implemented in the base class for the 3D hit generation, and as such applies to all 3D hits created, rather than just track-like or shower-like hits.

The actual change to the chi-squared terms listed previously is the inclusion of a new term, χ^2_{YZ}, which alongside the χ^2_{X} term means that the displacement of the

hits outside the detector is considered, not just the displacement of the created 3D hit X coordinate compared to the input hits. The resulting penalty term in full is

$$\text{Penalty Term} = \chi^2_{\text{Fit}} + \chi^2_{\text{X}} + \chi^2_{\text{YZ}} \tag{6.3.1}$$

with χ^2_{Fit} being the original term calculated when running the minimisation, from Mathematica, to produce the best Y and Z values, as outlined previously in Section 6.1. This χ^2_{YZ} term is calculated by considering the detector geometry, and storing the maximum displacement out of the detector. This maximum displacement is then squared and scaled by a scaling factor, σ_{YZ}. For the DUNE FD this scaling factor is set to 10 cm, which should allow hits up to around 3 cm outside the detector, with the cut-off being closer to 5 cm[2]. The actual code change is outlined in Algorithm 1, where the four distance calculation functions use the detector geometry that Pandora loads to calculate a displacement from each of the YZ detector faces.

Algorithm 1: Get distance to detector edge.

Input : V, the full detector volume, and $\vec{p}$, a 3D position.
Output: d, the maximum perpendicular distance outside the detector volume.

$distToEdge \leftarrow 0$;
$bestY \leftarrow p_Y$;
$bestZ \leftarrow p_Z$;
foreach $tpc \in V$ **do**
    $distToEdge \leftarrow$ `Max(`$distToEdge$`, GetDistFromTop(`tpc`, `p`))`;
    $distToEdge \leftarrow$ `Max(`$distToEdge$`, GetDistFromBottom(`tpc`, `p`))`;
    $distToEdge \leftarrow$ `Max(`$distToEdge$`, GetDistFromFront(`tpc`, `p`))`;
    $distToEdge \leftarrow$ `Max(`$distToEdge$`, GetDistFromRear(`tpc`, `p`))`;

Figure 6.7 shows a before and after of this change, cutting off hits that previously would fall outside the detector volume. Crucially, this change does not disallow hits that fall outside the detector volume, as this is technically possible by a small margin. Instead, it prefers hits that are reconstructed inside the volume, but removing any hits that are reconstructed too far outside the volume.

The value of σ_{YZ} can be varied, increasing it to relax the strength of the 3D containment term of the total chi-squared value associated with each hit. Varying

[2]The design of a LArTPC means that it is possible for charge to arrive from either side of the APA. This means that hits can be produced 'outside' the detector, whilst still being sensible.

this value shows how much the addition of this term impacts the hit creation, as at very high values of σ_{YZ} where the cut is doing nothing, hits can be constructed upwards of 7 m outside the detector. Figure 6.6 shows the impact of varying σ^2_{YZ}, by showing the displacement outside the detector for all hits that fall outside the detector. It can be seen that increasing the strength of this term not only applies a limit on the maximum displacement outside the detector, but also reduces the number of hits that fall outside the detector as a whole, rather than just moving the hits at high displacement to slightly lower values. The exact values for this can be seen in Table 6.1. The performance of these hits is analysed as part of the later improvements, as strictly cutting the hits that fall outside the detector does not improve the 3D reconstruction overall, unless those hits are now placed in the correct place after the cut is applied.

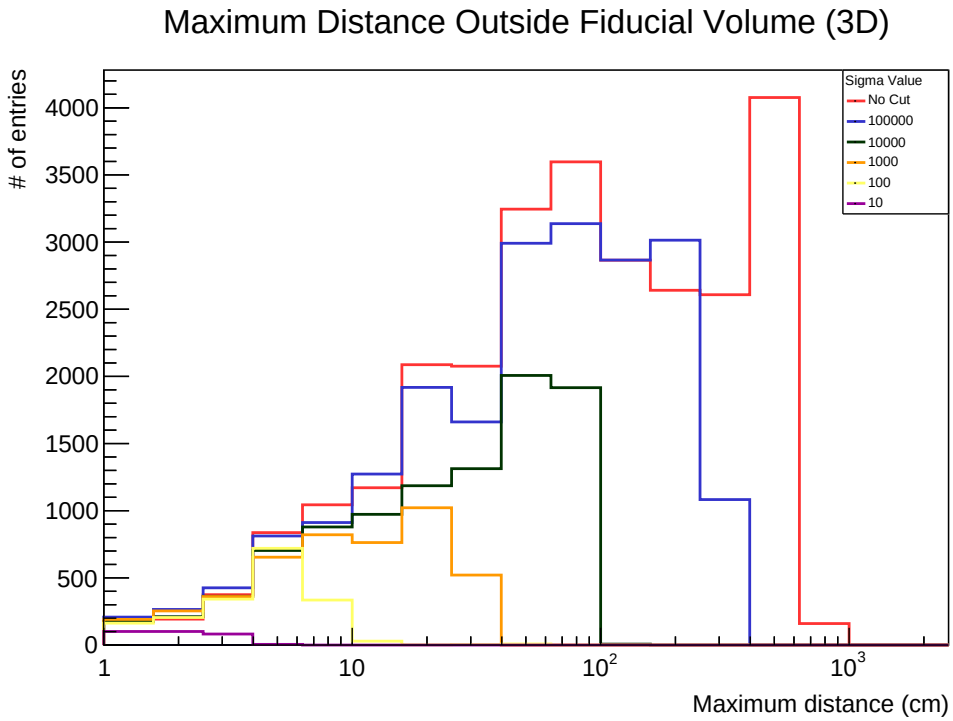


Figure 6.6: The impact of varying σ^2_{YZ} on the maximum displacement out of the detector volume. The maximum displacement is calculated as outlined in Equation 1, squared, then scaled by the σ^2_{YZ} term. If this term is varied, from no cut at all, down to 10 cm, the maximum displacement out of the detector drop from around 7 m down to around 5 cm. The total number of hits outside the detector also changes drastically, as better hits are chosen, and worse hits are disallowed with the increased chi-squared failing the cut. Values lower than 10 cm were tested and impacted the maximum distance further, but 10 cm was chosen as it allows the small amount of displacement allowed in a LArTPC detector.

$\sigma^2_{\mathbf{YZ}}$ **Value**	**Average Displacement**	**Number of Hits**
No Cut	156.31 ± 1.00	30469
100000	85.04 ± 0.58	22028
10000	31.74 ± 0.27	11126
1000	10.04 ± 0.13	6014
100	3.07 ± 0.06	3203
10	1.30 ± 0.03	835

Table 6.1: The average value for the maximum displacement outside the detector, for various values of σ^2_{YZ}. Varying this value alters the strength of this term, which in turn means that hits are viewed more or less favourably compared to other hits later on, or in more extreme cases will fail to pass the cut on the total chi-squared term that each potential 3D hit must pass. Number of hits refers to the total number of hits that fall outside the detector. Here, at the lowest sigma value of 10, only 835 hits lie outside of the detector, which is 0.01% of the total 3D hits produced.

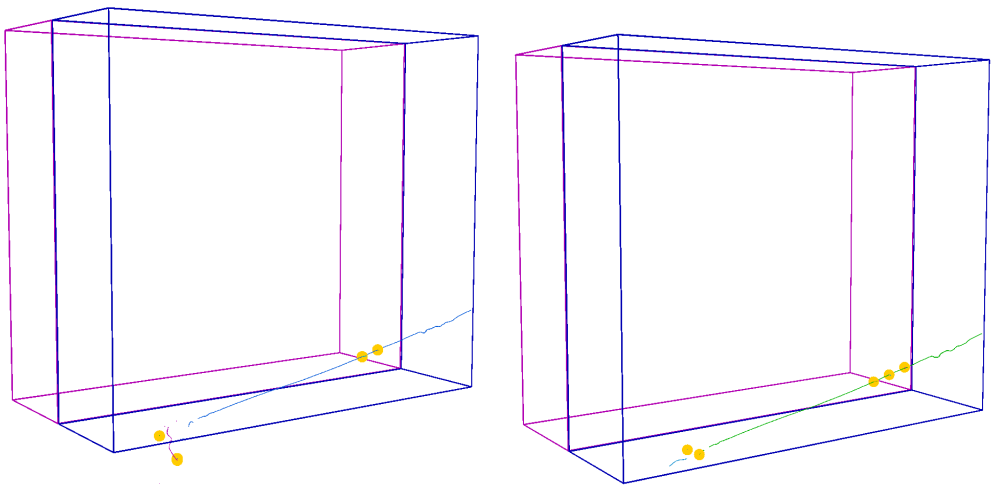


Figure 6.7: A before and after comparison of including a 3D containment term to the 3D hit chi-squared term. This shows an event that has interacted very close to the bottom of the detector, and has been reconstructed outside the detector. In the after, the hits that were reconstructed outside the detector are now either removed, or moved inside the detector depending on where the original 2D hits lie. This change in reconstruction also changed one of the vertex positions to inside the detector. Other changes to the after reconstruction are explained further in Sections 6.3.2 and 6.3.3.

6.3.2 Hit Interpolation

A second self-contained improvement is the addition of hit interpolation for missing 3D hits. This aim of this method was to help in the small cases where a suitable result is found, but there is a small part of the track that is difficult to match between views. Filling in these gaps with other algorithms can lead to inconsistencies, so instead a different approach was used to interpolate the hits there, using the 3D information that has already been reconstructed.

The process for hit interpolation is done using 2D based constraints, rather than 2D hit matching, and is performed as follows:

1. Remove any 2D hits that have already been used to create a 3D hit, as interpolation is not required for these hits. Store the currently created 3D hits.

2. Build a 3D sliding linear fit over the current 3D hits. This is like the 2D sliding linear fit, with some additional features. There is an additional axis calculated from two PCA steps, as the 3D fit is built upon multiple 2D-based fits. When built, the usage is broadly the same as in 2D, with the ability to smooth hits, reduce a dependence on the number of hits, and check consistency with the existing hits.

3. The created 3D sliding linear fit forms the core of the hit interpolation. For every 2D hit that does not have a corresponding 3D hit, project it into the coordinate space of the created 3D fit and calculate an `rL` value, which is the longitudinal displacement of the current 2D hit from the 3D fit axis in the 3D hit's coordinate space. With this `rL` value, the 2D hit can be projected onto the 3D fit to get a fit-based position. This is achieved by projecting the hit onto both of the underlying 2D fits that comprise the 3D fit, and combining the result. The result of this is a 3D hit that has been created from an underlying 2D hit with no hit matching, rather just using the existing 3D hits to estimate the position of the missing hit.

4. Finally, some constraints are applied to each of the interpolated hits. First, the 3D hit is projected back into 2D, such that it may be compared against the initial 2D hit it was based on. This distance is used as a pseudo chi-squared value for this interpolated hit, to compare the various interpolated hits. A final potential 3D hit is then made, combing the calculated 3D

position, the input 2D hit and the pseudo chi-squared value, as well as a flag that indicates this hit is interpolated, such that follow-up 3D hit algorithms can interact with them differently.

This sliding fit-based interpolation is repeated for every 2D hit that is missing a 3D hit, with limits on the total number of interpolated hits allowed. This limit ensures that the final reconstructed 3D hits are mostly based on the hit-based matching, which should be more accurate than the single hit interpolation method. If too many hits (i.e. the number of hits is over some configurable percentage threshold) are interpolated, the interpolation is cancelled early, as it is unlikely a realistic or useful result has been obtained.

Additional modifications were considered, but not implemented, as they were deemed unnecessary currently. For example, it is possible stronger constraints on the hit interpolation could be achieved by attempting some form of hit matching after the interpolation, either by comparing the interpolated hits to find 2D hits that produce similar 3D hits, or by projecting 3D hits into a view other than the input 2D hits. This could be used to further constrain the interpolated hits, which in turn could improve their positioning. A second change would be using the interpolated hits to improve the initial 3D sliding fit, such that larger gaps could be interpolated over, as there would be more hits in the fit to improve its interpolation performance. This approach was not used in the end due to concerns that without sufficient constraints, having a repeating process of producing and using 3D hits in a fit could lead to significant divergence from the true 3D positions, as each additional cycle would be using an even greater number of interpolated, and less constrained hits. This approach could be useful if additional constraints were added, but would need strict tuning to ensure that large majorities of hits are not interpolated.

6.3.3 Decoupling Tool Ordering

A key limitation of the existing implementation is the rigid, waterfall-like structure of the 3D reconstruction tools. Due to their fixed ordering, it is not possible for the most suitable tool to be run per track, nor is it possible to backtrack and use the output of one tool in cases where two different outputs are not consistent.

The improvement here is to decouple the tools from each other, allowing all six implementations to run independently of each other, to produce a full 3D

reconstruction. This means that every tool is given the complete set of 2D hits, rather than only the remaining hits from the previous step. This unlocks a lot more potential from the 3D reconstruction tools, but also introduces a number of follow-up issues. Now, rather than a single 3D hit for every 2D hit, there is at most six 3D hits per 2D hit, and a decision is needed to pick the best hit from this set of all hits. This introduces a lot more available information to the 3D reconstruction, which allows it to be more powerful, at the expense of needing additional tooling to select between all these hits.

To explore this decoupled approach, firstly, an assessment of this superset of hits was performed. This allowed an understanding of how these hits would look in events that previously had issues. Events that had no issues previously will have each of the six tools all produce the same reconstructed 3D hits, and are not of much concern, but the badly performing events could contain some correct total path made up of a chosen set of hits from the full superset of 3D hits. Figure 6.8 shows an example neutrino event in DUNE, overlaying all the candidate 3D hits outputted by the six matching tools.

This assessment showed that the tools that were currently implemented were able to offer sensible reconstructed 3D hits, even in the events with unphysical final 3D reconstructions. Overall, every tool is able to produce plenty of correct hits, though almost every tool also outputs some number of invalid hits. This means that overall, there is a mix of both correct hits and incorrect hits. With the previously rigid structuring of the tools, it is possible that one of the early tools produces an incorrect hit, where a correct hit would have been produced by a later tool, which a decoupled strategy could avoid by allowing both tools to produce their full track representation. Similarly, mixing hits between different tools leads to bumps and offsets in the 3D reconstruct, due to each tool reconstructing a slightly different 3D trajectory. Instead, if a method could be developed for selecting a single coherent and consistent path through the 3D cloud of all candidate outputs, this could result in an overall improved 3D track reconstruction.

Several approaches were considered to pull out a single consistent path from the many overlapping tools, but the simplest and most effective approach was through the use of random sample consensus (RANSAC) [112]. RANSAC is an iterative method for estimating the parameters of a model from an observed dataset, whilst ignoring outliers in the dataset. The most basic example would be fitting a straight line to a set of 2D data:

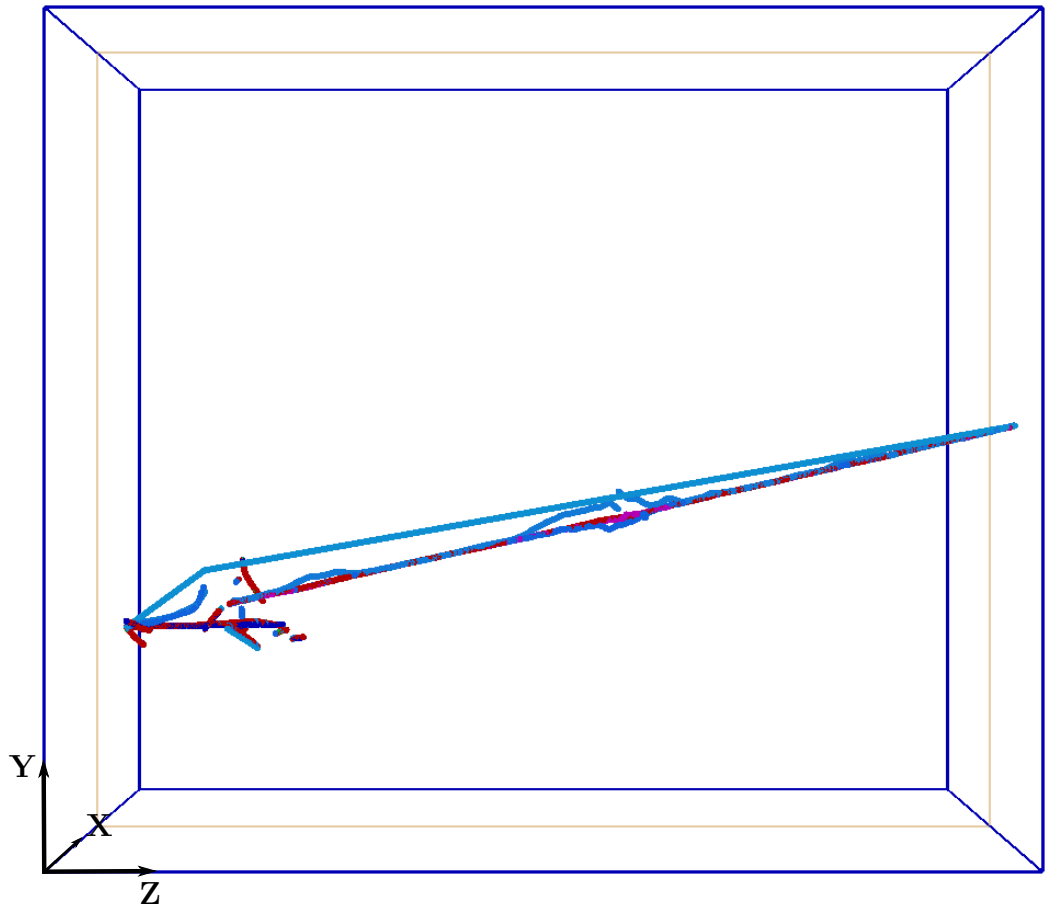


Figure 6.8: Example event display of an event with multiple 3D matching tools running simultaneously. Here, all the tools outlined in Section 6.1 are run over the full set of input 2D hits, to produce a 3D hit superset, with multiple options for each 2D hit. The different colours are only used to indicate different tools, but are not consistent across particles. It can be seen that there is both sensible and unphysical options available for each particle, with the main muon having the most obvious issues, including a straight line fit that does not align with the muon trajectory.

- First, take two random points from the dataset, and create a line using them. The randomness here is set up to be reproducible across runs and compiler versions, to ensure the 3D reconstruction is repeatable, by seeding the random number generation with properties of the current event. This should ensure that multiple runs of Pandora produce the same results, without using a fully fixed sequence of sampling.
- Next, evaluate this new model, by counting how much of the initial dataset is consistent with the generated line. For a line, this could be achieved by counting the number of points that lie on the line, or lie within some threshold of the line.
- Repeat this process N times, generating N lines and evaluating their performance. The best line is the line that has the largest number of points that lie on or near the line.

Through this approach, with a suitably large value of N (found through later tuning using the metrics outlined in Section 6.3.4), a model can be found for the data that is outlier resistant. If outliers are chosen to generate a line, then the number of points that lie on that line will be much lower than a line generated with two non-outlier points. This approach fits the problem at hand well: Most 3D reconstruction tools will output 3D hits that lie near each other, such that RANSAC should be able to find a sensible model that fits those hits, whilst ignoring any hits from the less constrained tools in the tool chain. Crucially though, in the cases where only a less constrained tool can produce a 3D hit due to the track shape being harder to reconstruct, a RANSAC fit that is able to fit those sections of the track will perform better than other models. Overall, the use of RANSAC ensures the path agreed upon by most of the tools will be chosen, which is the desired result so that less constrained tools do not throw off the results.

The model chosen for the 3D track fitting was a 3D plane, as this was deemed to be the simplest approach that reflects the vast majority of 3D tracks. That is, with three randomly chosen points, produce a flat plane between all three that extends out infinitely. Scoring for this model is simply achieved using a distance threshold, counting the number of hits that lie within some threshold of the 3D plane. What this means is that the full set of outputs from the previous step can be passed through many iterations of RANSAC, with a best model chosen based on the score outlined previously. Once a best model is found, all the hits that are

consistent with the best model may be selected as final 3D hits, subject to clean up to ensure uniqueness.

This model does however mean that the features of tracks such as kinks or curves cannot be accurately modelled by RANSAC, complicating the problem. The two potential solutions are to either consider a follow-up step to the model fitting, to add additional consistent hits to the generated model, or to extend the model used to track features of the event more closely. The first option was chosen, as it was deemed the simplest of the two approaches. Additionally, this approach made sense as a clean-up procedure was required already, to pull out a single final 3D hit, rather than the RANSAC generated list of inlying, consistent hits which could contain duplicate potential hits for a single input 2D hit.

Using the RANSAC model as the starting point, an iterative fitting procedure was added, utilising the 3D sliding linear fits as the core of the fitting. Conceptually, the linear fit is used to move along the RANSAC selected hits, before extending out to query nearby hits and check if they are consistent with the current fit. If they are, they can be added and the fit updated, which when used with a small enough sliding fit window, ensures that small local features in 3D can be followed, whilst also allowing a large enough window to be chosen that unphysical features are ignored. This process can be run twice, once from one end of the track over the selected hits and out the other end, and then repeated in the opposite direction. This allows the fitter to work with no forward or backward bias, which is useful for running on cosmic rays, or if the primary direction is reconstructed backwards. An example of this is shown in Figure 6.9, where a curving muon track has the first 60% of it fit with RANSAC, before the iterative fitting procedure fits the rest of the track.

This fitting procedure would allow the fixed straight line models that RANSAC fits to model a more realistic track with its kinks and curves, but also highlighted some issues with the RANSAC fitting procedure for very curved tracks. It is possible for some tracks to curve a large amount over the full distance of a DUNE FD detector, such that a straight line is only able to model a small portion of the real track positions. Worse, in a very curved case it is possible for some tools to fail for large portions of the track, resulting in a straight line fit from the track start to end. This is an issue, as RANSAC may prefer hits on a broken, straight line fit over a shorter but more realistic ft that only models part of a curve. Based on this, two additional features were added: Firstly, a weighting was associated

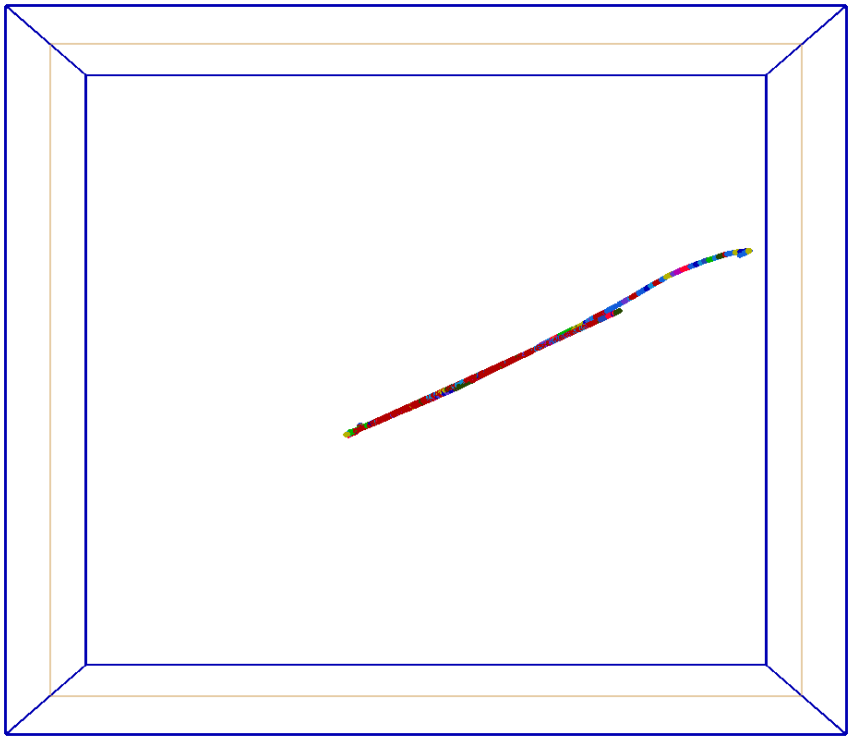


Figure 6.9: An example of the fit growing process on a ν_μ event. Here, the different colours used indicate different iterations of the fitting process, with the large red portion being the initial RANSAC fit. On the right of the track, it can be seen that the fitting process has extended the short RANSAC model to follow the full curve of the track over many iterations of the fitting procedure.

with each of the tools. This weighting allows the tools to be distinguished between in RANSAC or the fitting procedure, such that the less constrained tools are used less. This weighting is used similarly to how RANSAC is used, avoiding tools in cases where there are better alternatives, but to fall back on them in cases where the other tools fail.

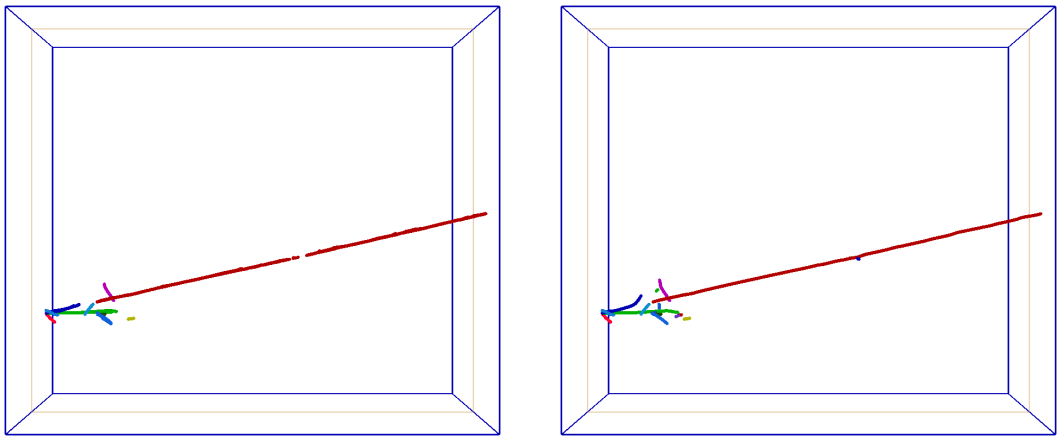
Secondly, rather than just taking the single best model from RANSAC, which stores every sampled model, instead take the best and second best, with the second-best being defined as the best scoring model with the most distinct hits compared to the best model. This scoring again uses the count of how many hits lie within some threshold of the 3D plane, but also with an additional constraint on distinctness. A requirement on distinctness helps ensure that the second-best model is not just the same model shifted over by a few centimetres. This second model is treated the same as the best model, being passed through the same fitting procedure. With both these features added, two final results are built, which helps avoid cases where poorly reconstructed candidate outputs from tools can take over RANSAC.

After this full process has run over the two models, there are two sets of consistent hits, built from a combination of a RANSAC model and iterative fitting. From this set of consistent hits, a single match is required for each 2D hit. This is achieved by selecting the hit that is the most consistent with the rest of the other hits, by comparing the displacement of the hit from the fit. Once this stage is complete, there is a singular 3D hit position for each 2D hit, rather than potentially many.

Finally, the two options are chosen between by calculating a score, by summing the number of hits in the fitted output, the number of hits in the original RANSAC result, and the number of favourable hits. This gives a good balance between 3D completeness and the quality of the hits, by picking the largest result with the most favoured results in. An example comparison of the initial output of RANSAC, compared with the final fitted result can be seen in Figure 6.10.

6.3.4 Performance Improvements

The performance improvements that both RANSAC and hit interpolation bring to the 3D reconstruction will be evaluated together, as the implementation for each is linked, such that each 3D hit matching tool is run, interpolated, and then RANSAC is used to model and fit this superset of 3D hits. It is possible for the



(a) Initial RANSAC output **(b)** Final RANSAC output

Figure 6.10: A RANSAC based approach, applied to all outputs for an event. Taking the full set of hits from Figure 6.8 as input, RANSAC can produce the shown fit initially. Figure 6.10a has not had any cleaning or further hit selection applied, which means there are duplicated 3D hits (hits that came from the same parent 2D hit) in the selected output, as well as gaps as no interpolation or further fitting has been applied. Figure 6.10b then shows the final chosen hits after the full RANSAC based procedure is run, including hit selection and further refinements to the RANSAC output.

interpolation to run stand-alone, however this method of operation was not tuned for.

One nuance of the reconstruction improvements reported here is that, luckily, the pathologies that are addressed here occur in a small subset of events, with most of the 3D track reconstruction working without issue and achieving reconstructed hits that reflect the underlying truth well. This means that most of the improvements made here are targeted at improving the tails of distribution whilst not causing issues with events that were reconstructed well. As a result, in some case it is useful to handscan some of the events that do change, as the result can be very drastic, with large changes being made in the 3D reconstructed hits to reflect the truth information in a more realistic way. Performing this detection automatically to plot only the tracks that change more than a set threshold is difficult, as there is no easy way to compare the reconstructed particles once the results are saved, as there is no unique identifier per-track, that would allow direct comparisons. For this reason, the metrics shown below instead show every track. The ideal result is to observe a reduction in the tail whilst the main peak of the plot remains consistent, indicating that the most broken events have been fixed,

and migrated towards a better result.

The most obvious comparison to make would be a comparison between the 3D hits and the true 3D positions in the MC. However, this is technically difficult due to the data reduction routines implemented within the DUNE simulation, which down-sample the truth information. This means that the 3D MC positions are sampled, reducing them to only the required hits to save space and to simplify the simulation output. This can be mitigated, as the positions where the direction changes should be kept such that the gaps can be interpolated over using the detector resolution to fill in the gaps. However, a large complication is the matching of these interpolated hits back to 2D hits, such that a relationship between 2D hit and 3D hit can be made, to compare reconstructed and simulated 3D positions is very difficult. These two factors combined mean that a direct measurement of the error in 3D position is hard to calculate. Luckily, an approach similar to the 3D matching tools can be used, instead of building a 3D sliding linear fit over the MC hits, which removes the requirement of a hit-to-hit based metric. Instead, the reconstructed hit can be projected onto two fits, one MC based and one reconstruction based. When these two agree, the reconstruction and MC are aligned, such that the 3D reconstruction has been performed well. This can be seen in Figures 6.11 and 6.12, where Figure 6.11 shows the average squared distance of a hit from a fit built on truth information. This allows a comparison against the MC information, but without the complications of the down-sampled truth information. The 3D hits that are close to their true position will be close to the MC fit, so will overall have a lower squared distance. Figure 6.12 on the other hand, shows the average displacement from a sliding linear fit built on the reconstructed hits. This then shows the average deviation from the fit, with a high average deviation indicating reconstructed hits that are inconsistent with each other. A low average displacement shows that hits are, on average, closer to each other hit-to-hit. It can be seen that the addition of RANSAC drastically reduces the tail in both the MC and reconstruction-based fits, corresponding to hits that lie very far away from the MC fit, and a reduction in hit-to-hit jitter. This means that RANSAC is choosing hits that are more consistent with the reconstructed hits around them, as well as being generally closer to the underlying truth.

One of the most drastic issues with the 3D reconstruction for tracks is the cases where multiple disjoint results end up being brought together, resulting in large unphysical deviations in the tracks, such that 'bumpy' tracks are output from

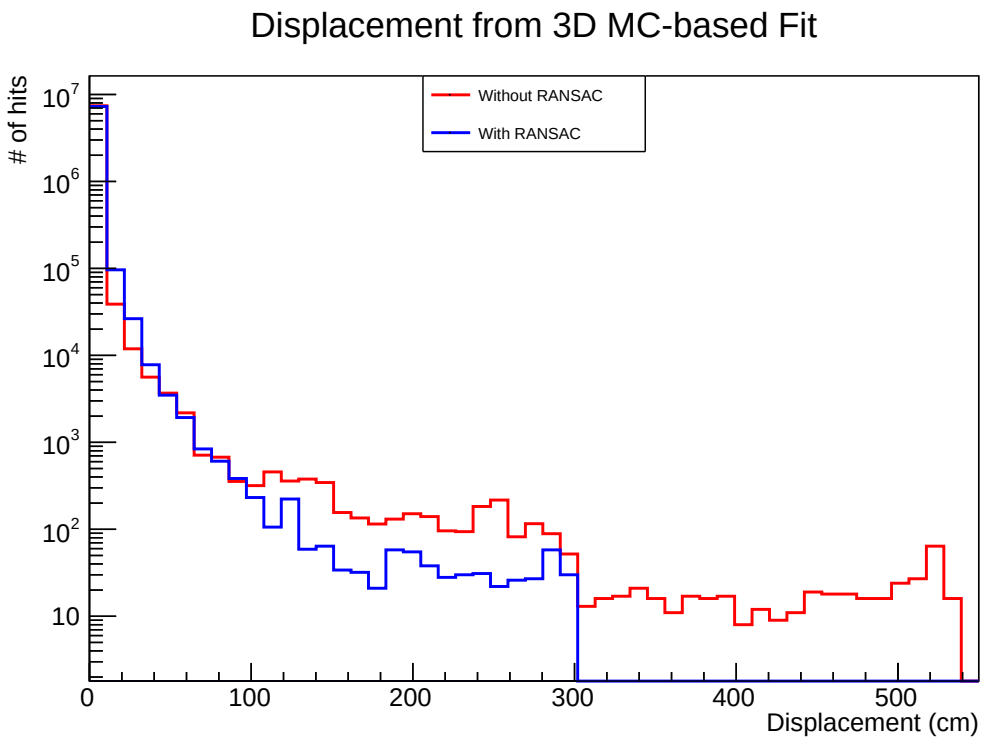


Figure 6.11: The average squared distance of a hit from a MC based fit, with and without RANSAC. Here, the 3D MC is used as the basis for a 3D sliding linear fit. Each of the reconstructed 3D hits for a track can then be projected on to the fit, and the squared magnitude of the difference between the reconstructed position and the MC based fit position can be stored. This plot then shows the average value of displacement per hit. This average has its tail drastically reduced with the inclusion of RANSAC and associated improvements. The most drastic changes can be seen with the removal of very high values of displacement, due to severe issues with the reconstruction, resulting in hits meters away from their true position.

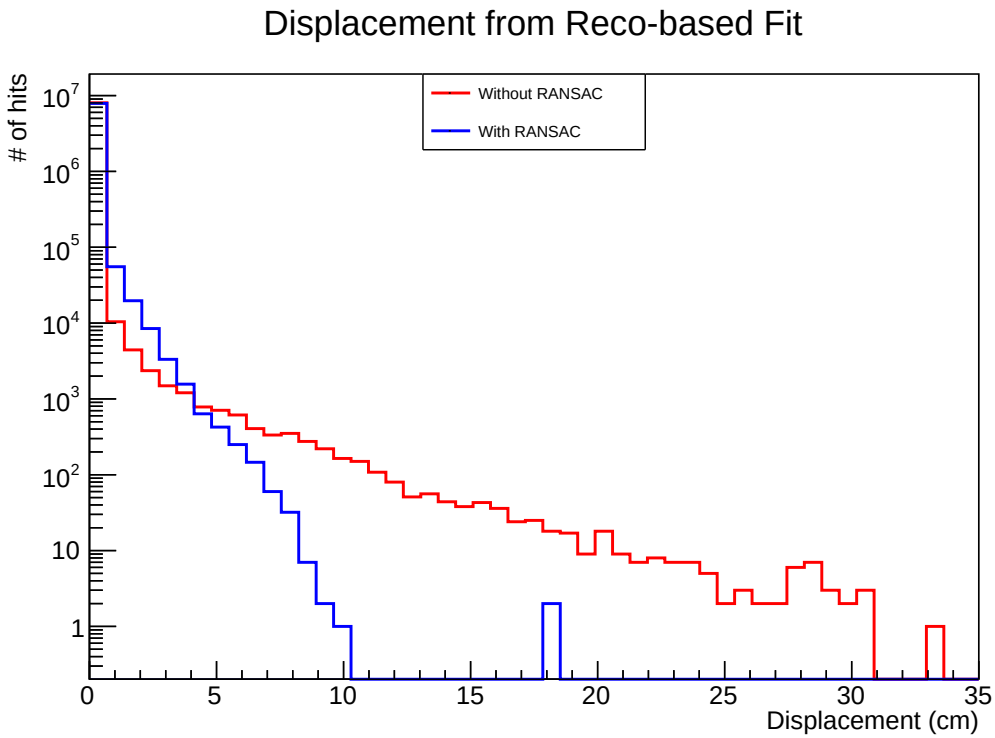


Figure 6.12: The average distance of a hit from a reconstructed based fit, with and without RANSAC. Unlike Figure 6.11, this uses the non-squared displacement, and the fit is built over the reconstructed hits. This allows an understanding of the consistency of the reconstructed hits, as a higher value indicates a higher average displacement from the fit, which in turn is only possible when the position from hit to hit varies drastically, as seen in failure cases such as Figure 6.5a. RANSAC can enforce consistency to the hits, which is lacking when RANSAC is removed.

Metric Name	Without RANSAC	With RANSAC
3D Completeness	87.5 ± 0.00	87.4 ± 0.00
Number of 3D Hits	514 ± 8	509 ± 8
Track Length	115.7 ± 1.65	113.2 ± 1.60
Hit Creation Failures	0.81 ± 0.00	0.81 ± 0.00

Table 6.2: The impact of RANSAC on various 3D hit creation metrics. The average value is given for each metric, and the addition of RANSAC does not negatively impact any of the metrics in a large way, whilst also unlocking the performance improvements shown in Figures 6.11, 6.12 and 6.13. A loss of some hits is expected in cases where RANSAC is not able to improve the 3D reconstruction, as shown in Figure 6.16. Additionally, this helps show that RANSAC is not simply ignoring hits to improve the previous metrics, rather selecting better hits.

Pandora. One way to quantify this is to measure the deviations in angle from a sliding linear fit. This is achieved by calculating the direction of a sliding linear fit on average, and then taking the dot product between this average direction and a direction calculated at a specific reconstructed position. That is, the jitter in a hit's direction relative to the average fit direction is calculated by,

$$\text{Hit Jitter} = \arccos\left(\vec{D}_{\text{Fit}} \cdot \vec{D}_{\text{Hit}}\right) \tag{6.3.2}$$

with $\vec{D}_{\text{Fit}}$ as the average direction of the fit, and $\vec{D}_{\text{Hit}}$ as the direction of the hit, calculated by projecting the given hit onto the sliding linear fit and using the two underlying 2D fits to produce a direction in 3D. Evaluating this distribution, should allow some measure of how 'straight' a track is, with the desire being that a track is broadly more straight over its full length, with higher values relating to lots of deviations from a mostly straight line. This can be seen in Figure 6.13, where the addition of RANSAC results in a sharp drop-off for high values of angular displacement.

To complement these distributions, Table 6.2 shows a few of the additional variables that were tracked, including the 3D completeness and the various sizes of the output. These numbers provide useful additional context that RANSAC can choose the most appropriate hits, which in turn improves the metrics above, rather than RANSAC just ignoring hits and improving the metrics through the removal of hits.

Finally, as a more subjective measure of the impact of RANSAC on the 3D

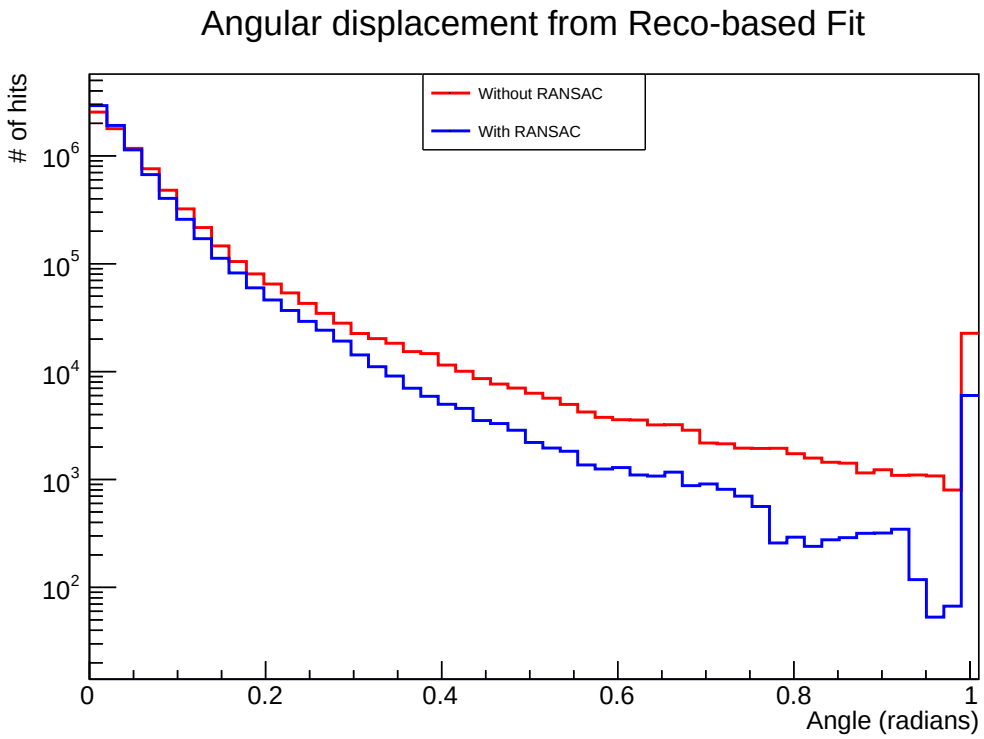
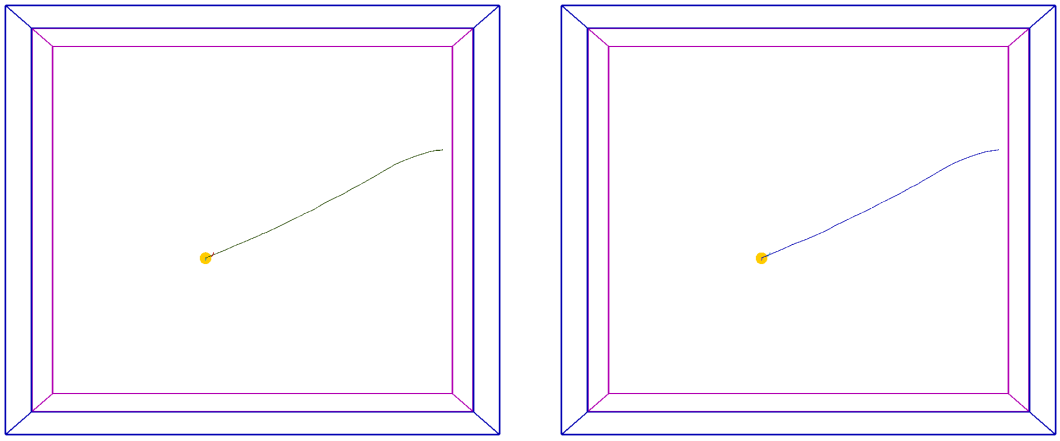


Figure 6.13: The angular displacement from a reconstruction-based fit with and without RANSAC. Values greater than 1 are clamped to 1. Lower overall angular displacement indicates that the tracks produced by RANSAC are straighter, with less jitter in the hits direction relative to the fit over the full length of each track. This is a desirable property, as without RANSAC, large deviations in the angular displacement can be seen when inconsistent results are merged, such as in Figure 6.4a.

reconstruction, and to understand where and how failures occur even with these further updates, a handscan of some events was undertaken. This allows a better understanding of how the input 3D hits to RANSAC are being used, as well as the fitting procedure that follows. It also means that cases that still fail can be investigated.

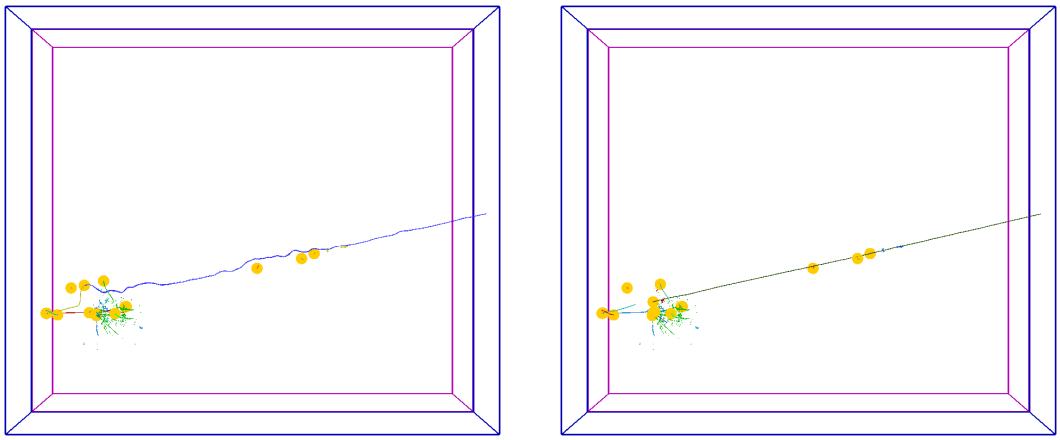


(a) Without RANSAC **(b)** Including RANSAC

Figure 6.14: A working ν_μ event that RANSAC does not change. RANSAC is able to select the most appropriate hits from all the produced hits, which enables it to improve events with unphysical topologies. In working events, however, where all tools produce the same or similar outputs, its impact is minimal. However, this is a good thing when the majority of events do not require RANSAC. This means that RANSAC can be enabled for all events, but only impacts those which need it. Differing hit colours represent distinct particles, and yellow circles show reconstructed 3D vertices.

For most events, such as Figure 6.14, the addition of RANSAC does not change the 3D reconstruction meaningfully, by design. For short tracks, RANSAC is set to short-circuit and avoid much of the fitting procedure for performance reasons, and for longer tracks with no ambiguities, every input tool produces the same output, such that there are few wrong choices to be made. The final output is the same with and without RANSAC picking between the output hits.

However, in cases where there were significant issues, RANSAC can either fix or drastically reduce the issues seen in an event. For example, Figure 6.15 shows an event that has a very unphysical track that has two main issues. Firstly, there are large deviations in the track where two tool outputs were smoothed between, resulting in a largely 'bumpy' track that is very unphysical and does not accurately reflect the real particle trajectory through the detector. This is mostly caused



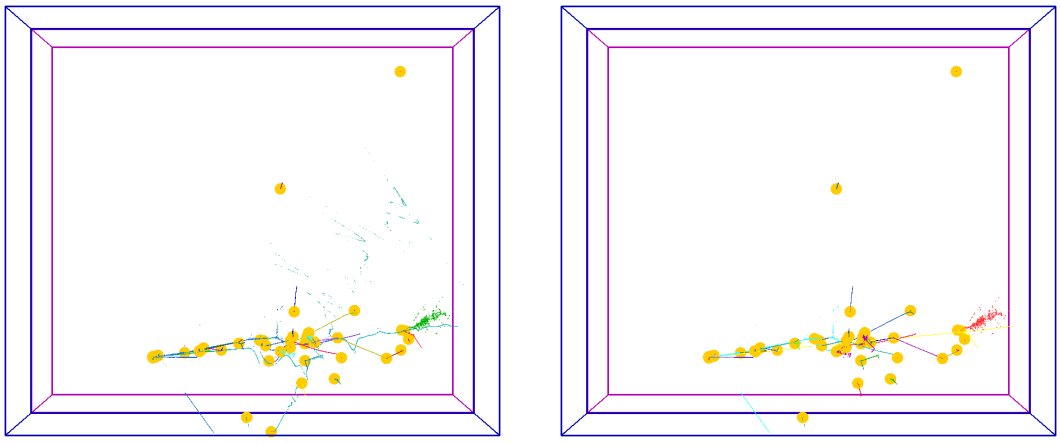
(a) Without RANSAC **(b)** Including RANSAC

Figure 6.15: A ν_μ event fixed by the inclusion of RANSAC that previously had an unphysical trajectory. Here, an earlier reconstruction failure results in the long muon track being split near the start of the track. This, along with the isochronous nature of the track at points, results in a track that is difficult to reconstruct with a single algorithm, resulting in disjoint trajectories being combined and smoothed between producing bumps throughout the track. RANSAC is instead able to smooth this track out by picking the consistent hits, resulting in a 3D reconstruction that represents the truth, as well as being consistent with the delta rays present on the track.

in this case due to the track being incorrectly split earlier in the reconstruction chain due to a secondary particle crossing the track, resulting in a split. This reconstruction issue cannot be fixed here, but the impact of it has been drastically reduced due to the addition of RANSAC meaning that the most sensible hits can be utilised, rather than merging two disjoint outputs.

In some cases, the addition of RANSAC and interpolation is not enough to fix the 3D reconstruction. As shown in Figure 6.16, if none of the six tools used are able to create sensible hits for a track, then it is not possible to intelligently find the best hits from the total set of hits. This is the major limitation of this improvement that its power is unlocking the full performance of the previously outlined tools, but does not offer additional options outside of interpolation in cases where these tools do not create useful results. Further work could extend the existing tool list such that RANSAC has additional options to use. However, even in the cases where the addition of RANSAC cannot produce a 3D reconstruction that accurately reflects the underlying truth, it does enforce stricter constraints on the 3D hits, which can help prevent some events having drastic failures.

Overall, the addition of RANSAC drastically improves the most unphysical



(a) Without RANSAC **(b)** Including RANSAC

Figure 6.16: A broken ν_μ event that the addition of RANSAC and interpolation does not fix. The addition of RANSAC does drastically constrain the output, resulting in less spurious hits as well as very unphysical results. The output does however still have multiple issues, including missing hits and tracks that have been locked to straight lines. This outlines the main limitation of RANSAC, where it can only produce a sensible output, if there are realistic hits in the input point cloud.

tracks, enforcing that the used 3D hits are consistent with each other. For smaller issues, RANSAC results in a more consistent track, with fewer deviations in the track, whilst also not breaking the reconstruction for tracks that worked without the inclusion of RANSAC. Combined with interpolation and an improved chi-squared, with the chi-squared changes also enabled for shower-like topologies, Pandora produces more consistent and sensible 3D output for use in later analyses.

7

Pandora Deep Learning Shower Growing

"Will it serve for any model to build mischief on?"

Don John - Much Ado About Nothing

Electromagnetic showers form a key part in neutrino oscillation physics measurements, used to tag the key ν_e appearance channel. Accurately reconstructed showers are needed not only to tag an electron neutrino interaction, but to get an accurate estimation of the neutrino's energy. However, they have a complex event topology, making them challenging to reconstruct, as shown in Figure 7.1; whilst a track can broadly be characterised as a straight line with a few kinks, showers have a complex tree-like structure. As an electromagnetic cascade develops within an event, the edges take on a diffuse and fuzzy structure that is complicated to reconstruct algorithmically, with the problem becoming even more difficult when you have multiple interactions in a single event. Defining the edge of a shower that has interacted next to a second shower or a track can be a complicated problem, even for the human eye, especially around the vertex region.

To address the challenges of shower reconstruction, Pandora builds up its showers in several stages across the entirety of its reconstruction pipeline. There are four key steps in the construction of showers: the initial hits are targeted by a series of 2D clustering algorithms, producing a collection of proto-clusters. This is followed by two streams of bespoke 2D clustering algorithms, to target tracks and showers independently. The 2D clustering algorithms that run in the

shower stream are collectively known as the 'shower growing' algorithm chain. With grown 2D showers, a 2D/3D matching procedure is run for both the track and shower clusters. Finally, there is a suite of final 'mop-up' algorithms, which use the additional information available in 3D to merge additional shower-like clusters, resulting in an overall, more complete shower with less orphaned shower fragments in the event. An example of these stages for a neutrino event is shown in Figure 7.1.

These issues make shower growing one of the hardest pattern recognition problems in LArTPC neutrino physics. An effective shower growing algorithm needs to be designed to deal with the complex environment of an electromagnetic shower, able to merge together many shower-like clusters whilst avoiding both under-clustering (producing a low completeness shower) and over-clustering (producing a low purity shower). For this reason, any algorithm must understand the underlying topology of an electromagnetic shower, or be able to learn it. This means the problem lends itself nicely to multi-algorithm reconstruction, with targeted algorithms chained together to tackle different aspects of shower reconstruction. However, it is also a perfect development ground for deep learning, with a network trained to understand the detailed topologies of an electromagnetic cascade, such that it can accurately reconstruct showers.

In this chapter, a deep learning-based approach to shower growing will be outlined, exploiting graph neural networks (GNNs) to intelligently merge proto-clusters into larger shower-like clusters. First, this chapter will outline the existing workflow that Pandora uses to perform this crucial shower growing step, before exploring the process that was used to understand and later improve this step. This starts with understanding potential improvements to the shower growing via cheating, outlining the issues with the current growing, and then explaining the deep learning-based approach that was taken to improve it. Finally, there is an outline of the training process, to tune the various hyperparameters associated with this method. Performance metrics are left to Chapter 8, with analysis to compare the improvements on both MC and real data from ProtoDUNE-SP.

7.1 Existing Shower Growing in Pandora

Pandora currently uses a topological method to grow 2D showers, based on algorithms that harness both event information such as the vertex, and positional

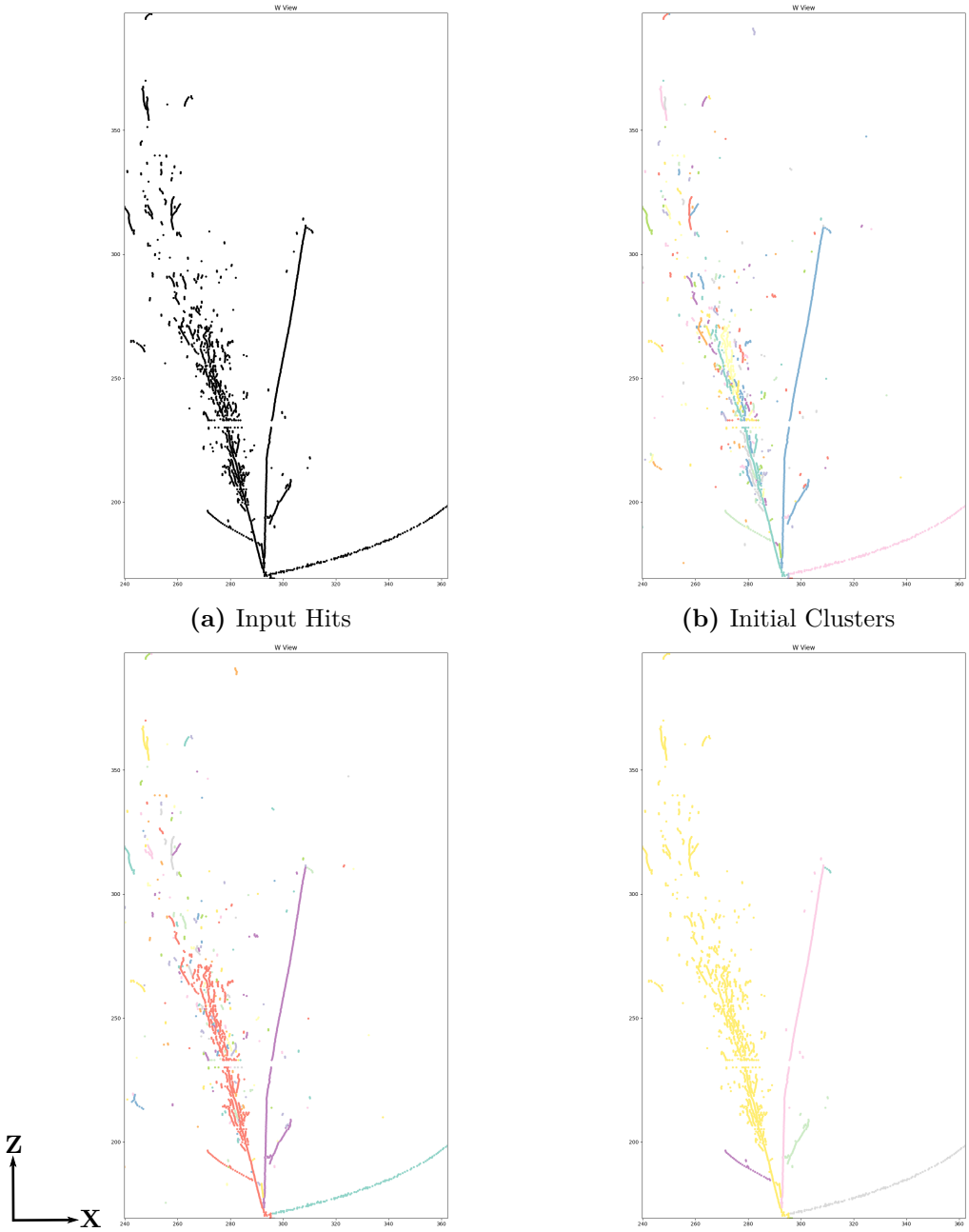
information like how close and aligned clusters are, to steer an algorithm that then makes decisions on how to merge shower-like clusters. Broadly, the approach taken by Pandora is to target underlying structure in the initial clusters, first by building a shower 'spine', a set of clusters that forms the centre of the shower, then attaching 'branches' on to this spine to build up a complete 2D shower. This growing happens in stages, first attaching the clearest branches, then followed up by attaching branches that are more weakly associated, but have the greatest overall association with the current shower, compared to other showers.

7.1.1 Shower Growing Algorithm

The existing, topological growing in Pandora starts by splitting the shower growing step into two steps: showers that start at the reconstructed vertex, and then non-vertex showers. The algorithms and process used are the same for both, but the addition of a vertex as a hook to constrain and aide the shower growing means that vertex-associated shower growing is performed first. Additionally, the vertex is usually the start of the shower spine, which makes it useful to start there to avoid issues where a small, secondary shower is lost as a spine was not formed early enough.
The full process consists of six steps:

1. First, the current reconstructed vertex is loaded, and is used to populate the list of candidate clusters, for later use. This list is composed of clusters that are tagged as shower-like by the DL hit tagging algorithm (as outlined in Section 4.3.8), with the required number of hits (which is five for the DUNE FD), along with an association check between the 2D cluster and a 2D projection of the current 3D vertex. For this, each 2D cluster is augmented with additional pointing data, based on a 2D sliding linear fit[1]. This includes finding the most upstream and downstream hit in the cluster, to perform later checks against these known positions.

   These augmented clusters are then compared to the vertex, to check if either end of the cluster is close to the vertex, or if the cluster is a valid emission from the vertex, based on the cluster direction and angular projections from the vertex. If a cluster is deemed to be either adjacent or emitted from the vertex, it is vertex associated and is stored as a vertex-associated cluster, also known as a shower spine.



(a) Input Hits **(b)** Initial Clusters

(c) After Growing **(d)** Final Clusters

Figure 7.1: The four main stages of the shower growing process, on an example ν_e event. Here, colour is used to show different clusters, although the limited number of available colours means colours are used many times. Starting from the raw, independent hits of the event, initial clustering takes place, which produces many small clusters shown in Figure 7.1b. The first round of shower growing, showing in Figure 7.1c shows that the main part of the shower is clustered, but there are parts missing from the shower. The final output in Figure 7.1d shows the complete shower, with the additional clusters being added in follow-up mop-up algorithms.

2. Using the list of spines, the remainder of the clusters that were not associated with the vertex are stored as potential merge candidates.

3. These two lists, of merge candidates and spines can then be combined, to find the merge candidates associated with each of the vertex-associated clusters. This is achieved by iterating over every merge candidate and finding any unused candidates that are associated with the shower spine. This works similarly to the vertex association, but with additional constraints. Using positional information, such as the width of both the spine and the merge candidate, as well as their direction relative to the beam, a cluster-to-spine map is generated which encodes the strength of the topological associations between the merge candidate cluster and spine.

4. A strong association is returned for clusters where one cluster encloses another, or if the two clusters overlap. A medium association is returned for branching clusters, clusters that have either end of the cluster within some threshold of each other. Finally, a weak association is for clusters that are nearby, based on a set threshold. If none of these limits are reached, no association is returned. These associations allow the strength of the merge potential to be stored, such that multiple clusters can be compared against a single spine, with only the strongest association being kept. This process is repeated for the full combinatorics of every spine against every candidate, until a full map is built, storing the strength of the association for each pair.

5. The penultimate step is to identify the actual cluster merges, based on the association map that was built previously. Here, every candidate cluster is iterated over, finding the strongest association from the full map. Weak cases can be ignored in some circumstances, as well as removal of clusters in cases where a candidate is equally associated with multiple spines, to avoid an ambiguous merge. Once every candidate has been checked, the final merge step can begin.

6. Finally, the association list that was built in the previous step is used to merge each spine and its associated merge candidates into a single cluster, with Pandora internal functions taking care of the managing of Pandora's internal understanding of the cluster list and more, such that it can be shared between algorithms. Clusters that are merged, both spine and candidate

clusters, are stored in a used cluster list, so the remaining growing algorithms can ignore them.

Once this procedure has finished, the showers that are topologically connected should be grown, and the only remaining step is to deal with the remaining showers that are not associated with the vertex.

This procedure works the same as the vertex-based growing, but instead of building up a list of spines, a single merge candidate is chosen and then used through the same association finding code and then growing code. This single merge candidate is chosen by iterating over every valid shower-like cluster, that is over the hit threshold and was not used in the vertex growing step. This ensures that every remaining shower-like cluster can be the basis for growing, such that showers that are far from the vertex can be grown.

The result is a fully grown event, with showers being grown based on topological information, based on comparison between clusters. This process is first performed on clusters that align with the vertex, but then more broadly to pick up any remaining clusters.

7.1.2 Topological Shower Growing Evaluation

To evaluate the performance of the shower growing reconstruction, the reconstructed showers can be compared with the output of a so-called 'cheated' reconstruction chain. Cheating here refers to using MC information in algorithms, such that a perfect algorithm can be created. This is a useful tool, as it allows the question "What if algorithm X was perfect?", to be analysed, such that the impact of improving a single part of Pandora can be assessed. The results of cheating studies can then feed back into development work, with cheated algorithms that unlock large overall performance gains being the target of the next iteration of development work, whereas cheated algorithms that do not unlock large gains indicating that the performance is being limited elsewhere. The cheating tools implemented in Pandora enable multiple parts of the algorithm chain to be cheated, meaning we can find out which areas need the most work to unlock some certain physics analysis [113].

In the context of shower growing, this allows us to disentangle two things: Can the output of the Pandora reconstruction be improved by more performant shower

[1]Both the sliding fit, and the 3D to 2D projection utilises the same underlying code as outlined in Section 6.1.

growing, and is the current performance limited by the initial 2D clustering that is used to seed the shower growing algorithm chain. A large improvement when cheating the shower growing would indicate that work on improving the shower cluster growing is a worthwhile target for development work. However, if the cheated growing does not unlock much additional performance, it is likely that the initial clustering that Pandora performs to group shower-like hits together into small clusters, is introducing too many reconstruction issues, such that even a perfect shower growing step is not able to improve the final result in terms of completeness or purity.

Luckily, cheating the shower growing is a simple task, and relies on a simple algorithm outlined in pseudocode in Algorithm 2 on page 144. Put simply, the cheated shower growing only needs to look up the main truth particle for each cluster, based on the hits that it contributed to the current cluster, and if two clusters share the same MC particle, merge them[2]. The only restrictions applied to the cheating are that the clusters must be made up of at least five calorimetric hits, and must be shower-like tagged. These requirements have been carried over from the existing shower growing, to make comparisons between cheating and the existing growing more fair.

Due to Pandora's modular nature, the configuration file for the standard reconstruction chain can be altered, such that the single line that defines the shower growing step now uses this cheated module, rather than the regular module. This allows an understanding of the cheated shower growing, whilst maintaining the rest of the real reconstruction chain. This is important, as it means a realistic reconstruction chain is used, with only a single cheating step.

This cheating study was performed against 1,000 MC ν_e events in the DUNE FD, with the only change in the reconstruction chain being the single change from regular to cheated shower growing. To evaluate the performance of the shower reconstruction, the hit-based purity and completeness of the reconstructed 2D showers is calculated before and after the existing Pandora shower growing algorithm chain. The shower growing algorithms are then replaced with a cheated version of the chain, and the performance metrics are re-calculated. To study the importance of shower growing within the complete particle-flow reconstruction, the full reconstruction chain is then run through to the end of Pandora, and the final

[2]The MC generators used here treat an entire shower as a single particle, rather that the more realistic simulation of individual photons and electrons in something like pair production, simplifying this process.

Algorithm 2: How cheated shower growing is performed, run once per view.
`isValid` checks if a cluster is valid for growing, based on its shower-like tag, size, and if the cluster has been merged already.
`getMCForCluster` is an internal Pandora function, that returns the MC information for a cluster, based on the particle that contributed the most hits to the cluster.
`MarkClustersToMerge` marks the clusters as used, and stores them to merge after the full process is complete.

Input : $\vec{C}$, a list of clusters, specific to one view.
Output: A list of merged clusters, specific to one view.
foreach *cluster* $\in \vec{C}$ **do**
    if !`IsValid(`*cluster*`)` **then**
        continue;
    $mc_1 \leftarrow$ `GetMCForCluster(`*cluster*`)`;
    foreach *otherCluster* $\in \vec{C}, cluster \neq otherCluster$ **do**
        if !`IsValid(`*otherCluster*`)` **then**
            continue;
        $mc_2 \leftarrow$ `GetMCForCluster(`*otherCluster*`)`;
        if $mc_1 \equiv mc_2$ **then**
            `MarkClustersToMerge(`*cluster, otherCluster*`)`;

	Completeness		Purity	
	All Showers	Largest	All Showers	Largest
Before & After Shower Growing				
Before Growing	07.8%	30.3%	95.3%	93.6%
Existing Growing	17.2%	53.1%	94.2%	90.5%
Cheated Growing	68.8%	69.1%	86.5%	95.9%
End Of Pandora				
Existing Growing	71.6%	88.1%	85.8%	87.7%
Cheated Growing	87.1%	92.8%	86.7%	89.3%

Table 7.1: Comparison between the cheated and existing shower growing for 1000 ν_e events. The first section of the table shows the completeness and purity for before and after shower growing, comparing the existing shower growing and the cheated shower growing. The second section of the table shows the final shower completeness and purity at the end of the Pandora reconstruction chain. In both cases, all showers are given, as well as a second value that only shows the single largest shower in the event. Statistical errors are omitted as they are below 0.01%.

completeness and purity metrics are calculated, for both the topological shower growing and the cheated shower growing. This distinction between the metrics was chosen to give the deepest insight into the exact changes the shower growing is making, as well as giving the overall final performance that Pandora ends with. In both cases, the metrics are reconstructed-object first, as the interest here is how the reconstruction performs and can be improved.

Table 7.1 shows the resulting metrics, averaged over 1,000 events for every reconstructed 2D shower, and for the largest shower in the event. Looking at the largest shower in the event is useful, not only because it is the crucial feature of a neutrino selection and energy reconstruction, but also to help normalise the plots somewhat, to a single shower per event, rather than many entries per event. In Table 7.1, it can be seen that there is significant room for improvement, with a 16% performance delta in completeness for the largest shower. This then leads to overall large improvements for the full particle-flow reconstruction, of order 4% for the largest shower and up to 15% for every shower in an event. All of this is achieved whilst maintaining high purity.

First, it is worth explaining why the cheated shower growing is still not able to achieve 100% completeness and purity. As the hit clustering and hit-tagging is not cheated, the missing performance is due to earlier reconstruction issues. If the hit-

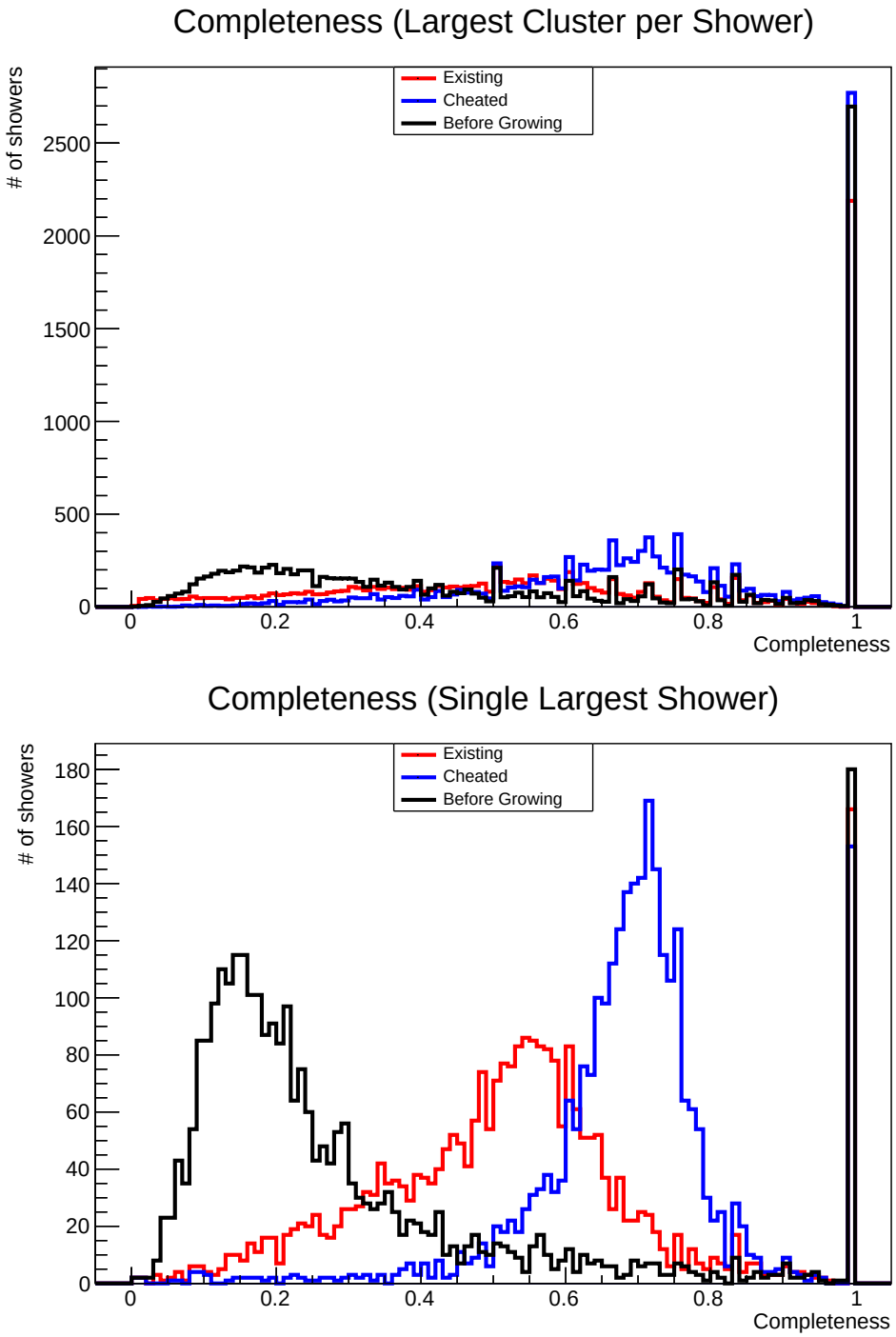


Figure 7.2: A study of the cheated shower growing performance, before and after the shower growing step. Here, the before growing stage is common to both the existing and cheated results. In the first plot, the reduction in showers at 100% completeness for the existing shower growing is due to merges made that remove small showers (5-15 hits total). In the largest shower case, this is due to the largest shower changing from a small, 100% complete shower, to a more useful large (100s of hits) shower.

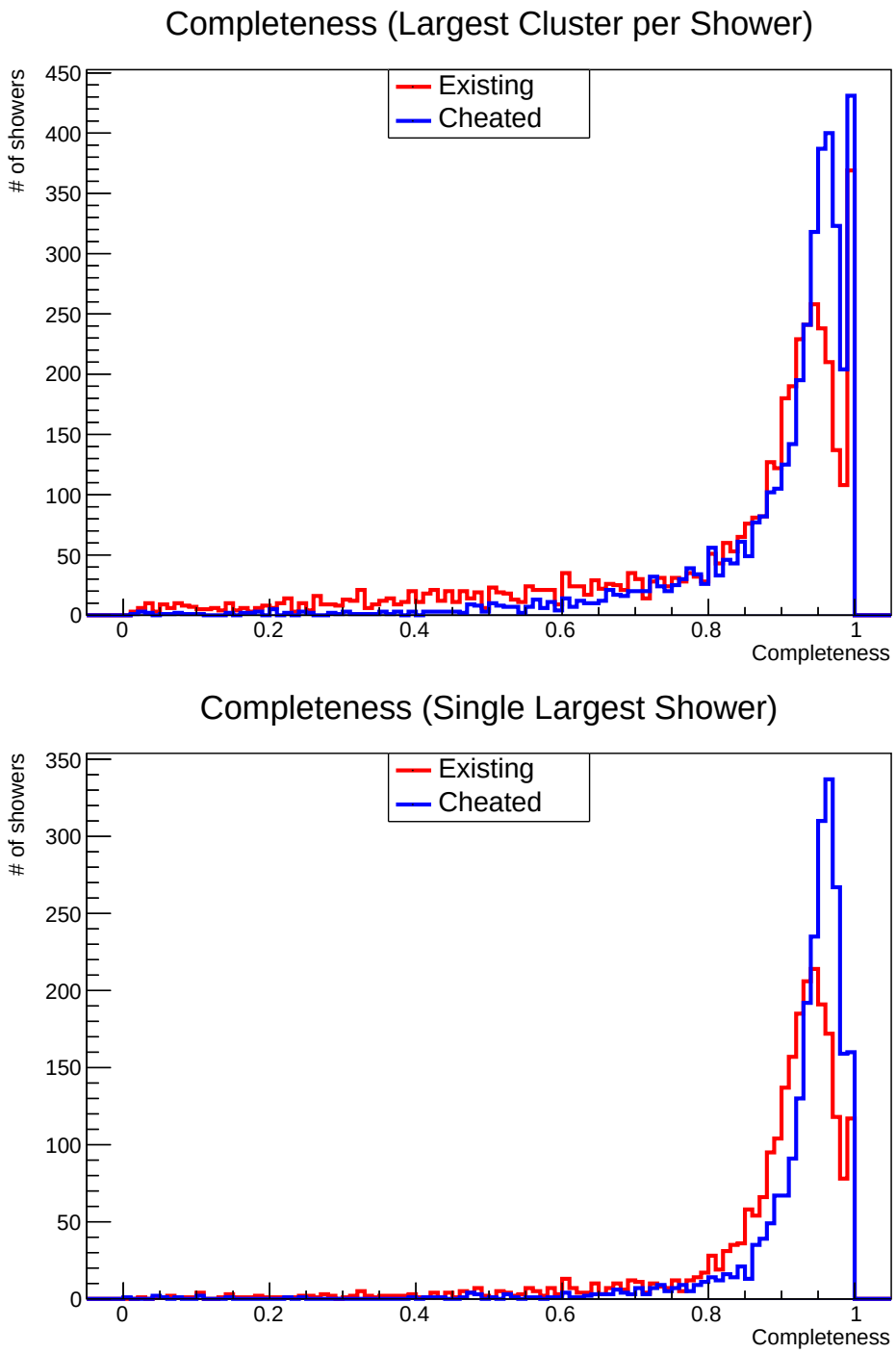


Figure 7.3: Comparison of the cheated shower growing performance, at the end of Pandora. This shows that the performance delta that is shown in Figure 7.2 is narrowed considerably after the shower growing step finishes, due to further mop-up algorithms which also improve the shower clustering. However, in both the largest per shower and largest shower in the event plots, there is still a clear performance difference, of more complete showers and fewer showers with low completeness.

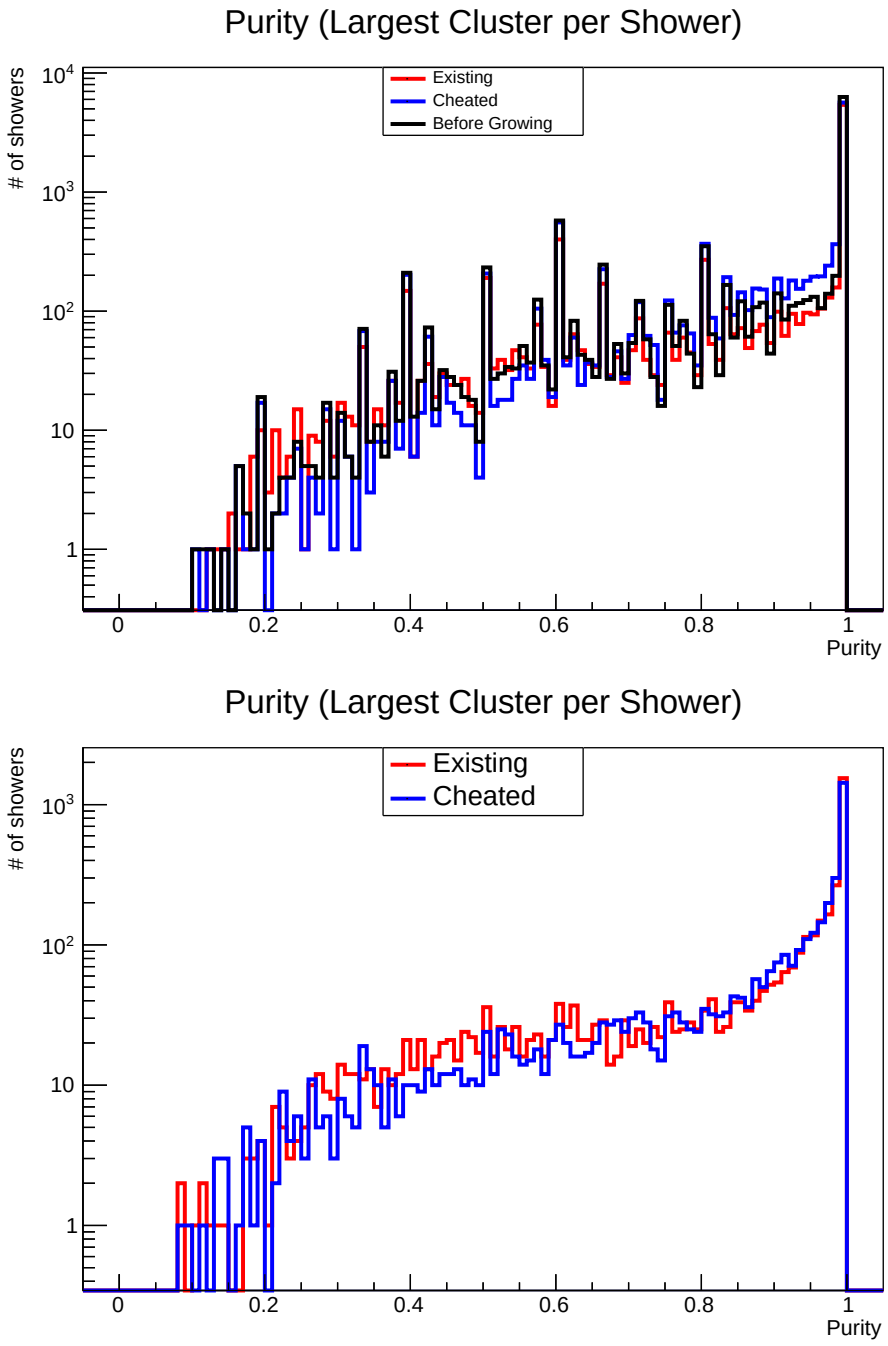


Figure 7.4: Comparison of the cheated shower growing purity, both around the shower growing step, and at the end of Pandora. Both plots here show the largest cluster per shower, rather than all showers or only the single largest shower in the event. In both cases, the purity does not change dramatically, indicating that it should be possible to gain improvements to the completeness, without altering the overall purity drastically. The lower count at 100% purity in the end of Pandora plot is likely due to the presence of highly pure, but low completeness showers, which are instead merged in the cheated case.

tagging algorithm does not correctly tag a hit, then the resulting cluster may be incorrectly tagged as track-like, which in turn means that cluster cannot be used in shower merging. Similarly, if the hit clustering merges hits from multiple particles, either tracks or showers, then the completeness and purity of the cheated shower growing cannot hit 100%, as the input clusters are made from multiple particles.

Figure 7.2 shows the distributions of completeness obtained before the shower growing, after the shower growing and on cheating the shower growing. These distributions demonstrate the potential gains that are possible if the reconstruction is perfect. There is a clear reduction in the number of low completeness showers, and a sharper peak at higher completeness values for the largest shower plot.

Whilst Figure 7.2 is encouraging, the final shower completeness and purity of Pandora is much higher than directly after the shower growing. Due to the suite of mop-up algorithms that also merge in clusters based on other information and improve the shower clustering further, an important check is to ensure that the improvements to the shower growing can make it through to the end of Pandora, as it is possible that the improvements the cheating makes to the clustering, are the same changes that the existing mop-up algorithms would perform, which would make improvements to the shower growing less important. Figure 7.3 shows that whilst the mop-up algorithms can drastically improve the performance of the existing clustering, the changes that the cheating makes are distinct and do overall improve the shower clustering.

Throughout this process and the upcoming new developments, shower completeness was the targeted reconstruction metric to improve, as it was decided that the shower clustering was already very pure, so making improvements to the completeness was the most sensible target. Figure 7.4 shows an example of how the purity is changing for both the before and after steps, as well as at the end of the reconstruction. Whilst high purity is obviously an important metric, high purity at the expense of low completeness is less useful than improving the completeness with marginal changes to the purity. If that was not the case, then instead a metric that combines both completeness and purity would make a more sensible target, and may be a more effective goal for certain physics goals.

Combined, these figures show us that changing the shower growing step could improve the performance of Pandora, with a performance gap of over 10% for all showers. Whilst it is unrealistic to be able to match cheating perfectly with a reconstruction algorithm, it is a useful indicator that shower growing is a useful

and worthwhile place to spend development time.

7.1.3 Existing Limitations

To create a useful improved shower growing, it is first necessary to understand the limitations with the existing shower growing algorithm, which leads to performance issues. There are three main issues with the current shower growing implementation, which the cheating is improving.

Firstly, it is difficult for the existing algorithms to separate overlapping showers, or showers that are adjacent to each other. The features that are in use mean that it can be difficult to distinguish close but distinct showers from each other, especially once the showers have begun to spread. This is because showers that are close by to each other can be difficult to disentangle when only considering the topological features of the clusters, especially when only using a single cluster pair, rather than a whole shower view. If instead the whole shower was considered, such that an association is built up within the full context of an entire potential shower, different strengths of associations may be possible, instead of the limited cluster against cluster approach.

Secondly, the existing algorithm does not utilise the full amount of topological information available to it. After a spine is grown, it along with the candidates merged into it are removed from consideration. This means that the larger, more easily grown clusters are not used as an input, even when they could provide additional constraints to follow-up growing steps.

The final issue is that the shower growing in its current form is very conservative, and only makes merges when it is very confident, therefore missing many potential merges. This is because, initially, the shower growing was made to form a solid base for the rest of the shower algorithms, the shower 'trunk', with the remaining shower-like clusters being attached to the most appropriate shower trunk as the reconstruction continues. This is demonstrated in its strict vertex-based approach to start the growing. This means that the resulting completeness after the shower growing is still somewhat low. This was intended to be propped up and improved by the remaining algorithms, but the cheating results show that there is a noticeable performance improvement at the end of the reconstruction chain if the shower growing step can produce a more complete representation of each shower. This can also feed into other algorithms, as a better reconstruction of a shower earlier means that other algorithms can perform better, as well as

avoiding potential issues where small shower-like or track-like clusters are merged incorrectly into other particles.

To address these limitations, a more intelligent growing is needed, which takes a more global event view into consideration could provide additional performance, without drastically impacting the performance of the growing step.

7.2 GNN Based Growing

Shower growing is a step that is very visual, with orientations of clusters, the distances between them and the shapes of the clusters all feeding into the decision-making process, along with more global event information such as any other particles that are nearby and if the cluster is more compatible with another particle. For this reason, and to exploit the recent integration of DL in Pandora, as outlined in Section 4.3.8, a deep learning-based approach was undertaken. A deep learning model of some form could take as input the full set of every shower-like cluster, giving it a more global event view, using this additional information to more intelligently distinguish between multiple showers in an interaction.

Two approaches make the most sense for the data of a LArTPC: a sparse CNN or a GNN. The format of LArTPC data is naturally very sparse, with large amounts of the data being zero, but with locally dense areas of 2D hits, which makes it an ideal candidate for use with sparse CNNs and GNNs due to their ability to ignore the unused parts of the data, rather than waste computing resources computing over empty parts of the data such as in a more traditional CNN. However, from a more technical point-of-view, the GNN is a more sensible choice due to its more mature integration with `libtorch`, the C++ interface for PyTorch, whereas sparse CNNs have less support in C++, at least at the time of development. This is important for deep integration in the Pandora codebase, rather than the more common deployment of deep learning in a Python environment. Therefore, it was decided to develop a new shower growing algorithm based on a GNN approach.

The addition of graph-based deep learning required the inclusion of an additional deep learning library, PyTorch Geometric [114], which includes headers for inclusion in the Pandora C++ codebase. This library provides tooling and additional models to aid development of graph-based deep learning methods.

The actual issue of shower growing can be approached from a few different ways whilst utilising a GNN[3]. The simplest approach is to formulate growing as

a classification problem, selecting an input cluster and then classifying nodes as a merge candidate or not. Secondly, this problem could be formulated as an edge prediction network, such that the network predicts the existence of edges between nodes, which in turn could be used to power merge decisions. Thirdly, a more event-wide approach would be a clustering-based approach, that clusters the whole event in a single pass, such that nodes from a single shower are clustered together. Of these approaches, the first was used here as it seemed the simplest approach to use, whereas an event-wide clustering approach is a more cutting-edge use of GNNs, which means that the techniques for it are much less mature compared to node or edge classification. This decision was also shaped by the technological limitations that exist around available computing hardware, as outlined later.

7.2.1 Graph Structure

The chosen graph structure is a crucial component in graph-based deep learning. As the graph is the singular input to the network, and the operations used in graph-based deep learning exploit the structure of the graph, using a flawed graph structure can severely impact the performance of the network. Multiple approaches were considered, with the main decisions outlined here.

Nodes

The nodes of the input graph can represent the clusters of the showers in the graph. However, as there is a large amount of useful information in the underlying 2D hits that make up a 2D cluster, utilising the underlying hits directly is desirable. The compromise with using hits directly, with a node-based feature (and associated edges) to link back to the initial 2D cluster, is the drastic increase in the number of nodes. For that reason, some amount of per-cluster rounding may be useful. That is, the coordinates of the 2D hits that make up each cluster could be rounded to the nearest whole factor, effectively rounding the hits to a fixed grid. For example a hit at $53.1, 109.8$ cm could be rounded to $53, 110$ cm if rounded to the nearest whole cm, or $55, 110$ cm if rounded to the nearest five cm. Applying this rounding whilst still only grouping hits from a single cluster to avoid ambiguities between clusters would drastically reduce the number of calorimetric hits, which in turn

[3] A more general introduction to graphs and graph neural networks (GNNs) is given in Section 5.2.2.

means there are fewer nodes in the graph. This speeds up the processing to create the graph, and the model training and inference, or allows a larger model to be run due to the smaller input size. However, rounding does also reduce information, such that clear connections between clusters may now be less clear due to the impact of rounding. Rounding was implemented as a tunable parameter, with lower values leading to more nodes and more information, but also complicating the graphs. The impact of rounding for an example can be seen in Figure 7.5. In all cases, when grouping hits to produce nodes, grouping is only performed with hits that belong to the same cluster, to ensure that one node represents either a single hit, or part of a single cluster. This means that with no rounding at all, a single cluster will have a number of nodes equal to the number of 2D hits that make up that cluster. With rounding, this number may be reduced with extreme values reducing a cluster to a single node.

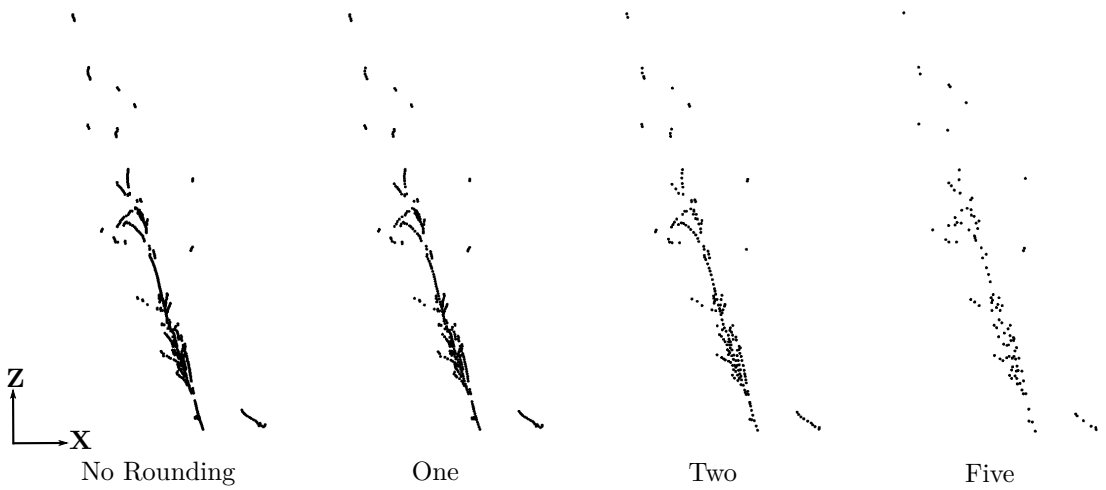


Figure 7.5: Various strengths of node-based rounding on an example ν_e event. 7.5a shows the original input event, and then Figures 7.5b, 7.5c, and 7.5d show various strengths of rounding, with the input hit positions rounded to the nearest integer, nearest even number and nearest multiple of five respectively. It can be seen that large details of the event remain throughout, but smaller details are quickly lost.

Another early decision was if the shower growing should include both track-like and shower-like clusters. Including track-like clusters would add a lot of information to the graph, but would also increase the complexity of the graph considerably. Both the nodes and edges would need to include some form of encoding to indicate the type of cluster, and the network would need to learn the difference and not to use track-like clusters for growing. It was decided to not

include track-like clusters, to reduce the complexity of the graph. This does mean that some of the more global event level information is missing, which could help distinguish between some showers, so the inclusion of track-like clusters may be a sensible extension when it is feasible to offload some computation of the network to a GPU. This does, however, make the network more similar to the existing shower growing, as that also does not consider the track-like clusters. The track and shower tagging is included as part of the input selection, however.

Edges

Edges are used to represent connections, relationships, between nodes. Here, the edges are used to connect up nodes, with the edges encoding information about the relationship between clusters, including topological information. Alongside reducing the number of nodes, reducing the number of edges is also an important decision. This means that edges should only be made in some cases, rather than producing a fully connected graph, so an intelligent way of deciding which nodes to connect and which to avoid is needed.

The simplest way to choose edges in the dataset is a simple distance-based metric, such that all nodes within some distance are connected. This is simple, and means that there are many connections that can be useful in the message passing step of a GNN. However, it can result in excessive numbers of edges for certain parts of the graph, or even produce a fully connected graph in cases where the event itself is small. As such, a K-nearest neighbour (KNN) approach was considered that is used in other graph-based approaches. Here, each node is connected to its K-nearest neighbours, that is, the K nodes that are nearest to the node. This avoids the issue of excessive numbers of nodes in dense events, but can have the opposite impact: in some events, the nearest node may end up many centimetres or even metres away, which is unlikely to be useful information. An example of how varying K for a ν_e event can be seen in Figure 7.6, demonstrating the balance in choosing a suitable value.

The final design of the network utilises both a KNN and a distance-based cut, such that nodes that are a significant distance away are not connected. This helps avoid the issues outlined previously, though tuning was required to pick the optimal balance of distance and how many neighbours to connect to.

A second relationship that needs to be encoded in the graph's edges, outside of cluster-to-cluster relationships, is the fact that a single cluster may be represented

by many nodes. That is, a single cluster of N hits may be split into M nodes, and edges are needed to outline the relationship between these M nodes, such that the network can learn and exploit the fact that some nodes are already related. These edges are used to encode an already well-defined relationship, so the main issue was not about what to connect, but instead how dense the connections should be. Every node in a cluster could be connected to every other node, or there could be a connection only to the previous and next node. Testing showed this decision did not make any difference to inference performance, so producing edges between only the next and previous node was chosen to keep the number of edges low.

The final feature around edges was an optional feature, designed to protect nodes around the vertex. The feature allows a vertex radius to be defined, and the nodes in that radius have a strict cut-off on the angle between the two nodes. This allows edges that are close to the vertex between clusters that have well aligned pointing information, but forbids them in cases where the underlying clusters are not consistent to some threshold.

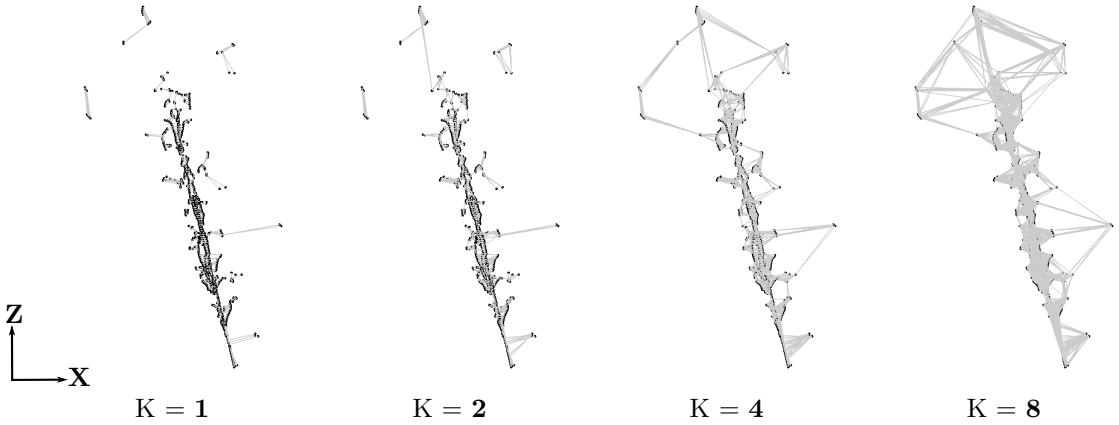


Figure 7.6: Impact of varying K in K-nearest neighbour for a ν_e event. There are two key impacts from varying the value of K. Firstly, there is a trade-off around the graph being more connected, such as connections to the nodes near the top of the shower, and too connected, such that connections are being made to nodes that are physically meters away in the detector. Secondly, as K increases, the density in the trunk of the shower increases, and can reach a point where there are a lot of short edges in areas of high node density, which can in turn slow the network down. Implementing a distance-based cut-off means the first issue can be avoided somewhat, but tuning the value of K is difficult, as it directly impacts the message-passing stage of the GNN. There is a balance between processing time, and having enough edges such that each node has many neighbours to share information with.

Features

Outside the actual structure of the graph, the features associated with each of the nodes and edges is critical. Each node should contain some cluster level features, that outlines features of the actual cluster the node is based on. Similarly, the edges should have cluster-to-cluster based information, calculated between the two clusters the nodes represent. These features, alongside the structure of the graph, then power the GNN to allow it to make decisions.

The features chosen here are based on the original shower growing code, as well as some additional features, chosen to exploit the graph-based structure of the data.

The node features are,

- An **input flag** to indicate the input to the shower growing network. This flag allows the network to understand which set of nodes correspond to the input cluster, for which merge candidates should be generated.
- The **number of hits** in the current cluster. This is useful as a single cluster can be split into many nodes due to both rounding and just converting a set of hits into a node. This allows the network to understand that a single node is part of a much bigger cluster.
- The **mean position** of the current cluster, split into X and Z components. These values are scaled closer to a zero to one scale, to bring the values into a range that is more easily learnt from, rather than the hundreds or thousands of the original coordinates, which can potentially cause issues for the learning process of the network.
- An **orientation** indicator, encoding the same values as outlined for the pointing clusters in Section 7.1.1, to outline how the original cluster is oriented in the detector.
- Finally, the **mean vertex displacement**, which is a displacement value for how far the mean position of the cluster is from the reconstructed vertex of the event. This value can be used to protect nodes that lie close to the vertex, by applying stricter restrictions on the edges within some distance of the vertex.

Similarly, there are edge specific features, to encode features calculated between two different clusters. For internal edges, those that link up nodes from the same original cluster, these values are set to zero.

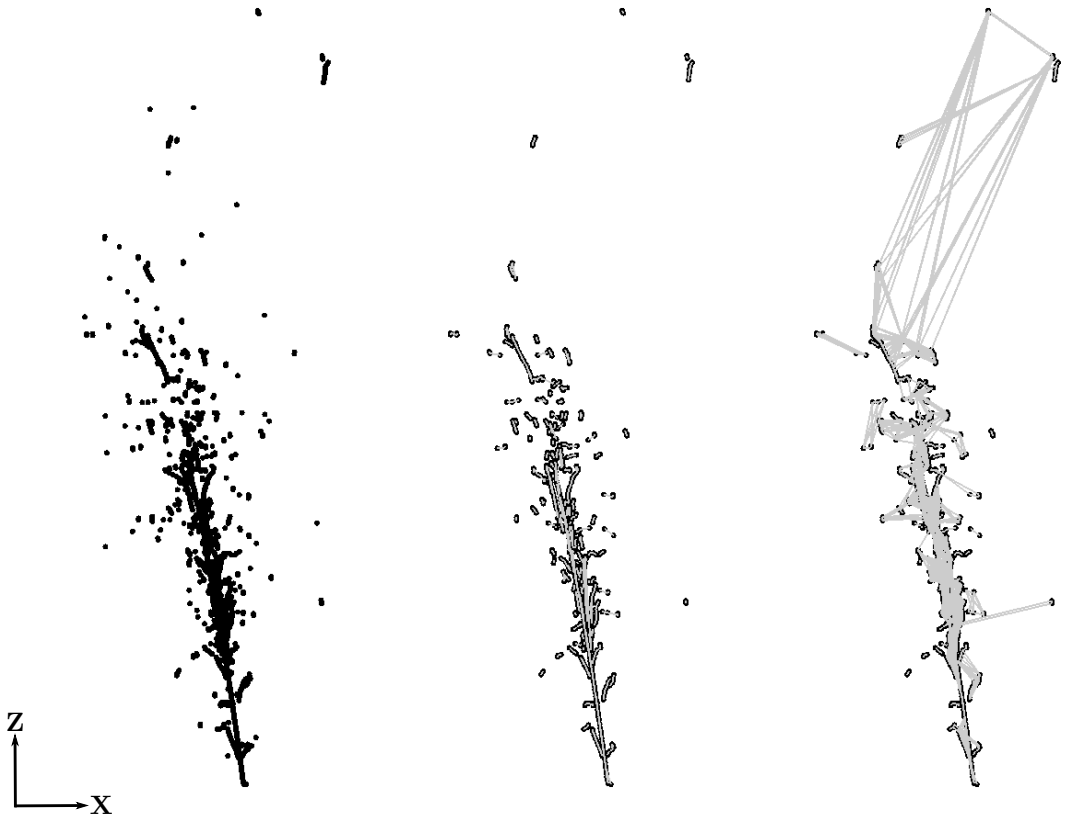
- An **internal edge flag**, to indicate an edge that is used to connect two nodes from the same original cluster. The rest of the features will be set to zero.
- The **mean distance** between the two nodes, calculated as the difference between the mean position of the two clusters.
- The **distance of closest approach**, which is the shortest distance between any of the nodes that make up the two clusters. This feature is useful for nodes that have a central position that may be further apart, but with hits at the cluster edge that are close together.
- Finally, a measure of the **angle between** the two clusters. This is calculated by first running a PCA over the cluster hits, and then taking the difference between both the X and Z axes components of the two current clusters, to calculate the total angle between the two clusters in radians.

These features, combined with the outlined structure of the graph, produce a final input graph. This graph structure was chosen based on the style of network chosen, whereas a different style of network, such as an edge prediction network, may have a significantly different structure. This graph then forms the basis of a graph-based neural network. An example of an event looks as a graph is shown in Figure 7.7.

7.2.2 Network Structure

The GNN first takes as its input the graph representation of the showers in the interaction, with features and structure as described previously. This includes a cluster tagged as the input cluster. The network will then perform inference on this graph, resulting in an output per node which is a percentage for merging or not merging with the initial input cluster.

The chosen network architecture was built using existing network modules built into PyTorch Geometric, partially to keep complexity down, but also to ensure easier compatibility with the `libtorch` interface when exporting the trained



(a) Raw Event **(b)** Internal Edges **(c)** All Edges

Figure 7.7: An example ν_e event and corresponding graph. Figure 7.7a shows the raw hits from the event, in the W view. Figures 7.7b and 7.7c show an example graph for the same event, with 7.7b showing only internal edges for clarity. Black denotes calorimetric hits in 7.7a and nodes of the graph in the remainder, with the grey lines representing edges of the graph. Clusters that are present in 7.7a, but are missing from the graph are either track-like tagged, or did not pass the 1 hit threshold in the shower growing graph building. This graph was created with a rounding value of 1 and KNN set to 9. For examples of varying the rounding value, see Figure 7.5.

model out in a C++-compatible way for later use. The exact modules used were the `GENConv`, the GENeralized Graph Convolution, as well as the `DeepGCNLayer` [115]. The `GENConv` layer takes care of the actual graph convolution, whereas the `DeepGCNLayer` encapsulates the graph convolution, the layer-wise normalisation and the chosen activation function, and runs them in the desired order whilst offering skip connections to previous layers as well. The use of these modules as well as the base Torch building blocks such as MLPs, various activation functions and more means that producing a C++ compliant version of the network is not difficult.

Three components comprise the full GNN; There are two encoders for the inputs, and then layers of graph convolutions, before a final output layer:

- **Input Encoders**: First, there are two linear layers, which apply a linear transformation to the incoming data of the form $y = xW^T + b$, with y being the output of the layer, x the input, b the bias term, and W^T is the transpose of the weight matrix for the layer. These two linear transformers act as encoders, one for the node features and one for the edge features. This allows the features of the graph, especially the categorical ones, to be transformed into a mode that is more easily learnt from.

- **Graph Convolutions**: After the node and edge level features have been passed through the encoder layer, there is the actual graph convolution layers. These are `DeepGCNLayer` blocks, built containing a `GENConv` layer, the ReLU activation function and an elementwise layer normalisation block. These three components are combined into a single `DeepGCNLayer`, that encompasses the correct running order of these layers, as well as applying dropout. There are a number of hyperparameters for both the graph convolution and the encapsulating layer, which are outlined later. Whilst there is only a single set of encoders, there are many layers of graph convolutions, to build up event level information.

- **Output layer**: The final component is a final linear layer. This takes the output of the N^{th} `DeepGCNLayer`, and compresses the result down to the output number of classes, which in this case is two, to align with the 'should merge' and 'should not merge' classes. For this reason, a logarithmic softmax[4] is applied on the output of this layer to produce a final percentage that sums to 100% for the two merge options.

Each of the encoders and the graph convolution layers utilise a number of hidden nodes. For the encoders, this simply means that the input goes from the N input features to some number of nodes. There are then an equal number of nodes throughout each layer, until the final linear layer that reduces the output to the required two output classes. For the `GENConv`, it constructs the messages that are passed during the message passing stage as follows:

$$\vec{x}'_i = \text{MLP}\left(\vec{x}_i + \text{AGG}\left(\text{ReLU}(\vec{x}_j + \vec{e}_{ji}) + \epsilon : j \in \mathcal{N}(i)\right)\right) \tag{7.2.1}$$

with MLP referring to a MLP of a chosen size, here 64. The AGG function is also user defined, and in this case, the message aggregation scheme used was the softmax function. $\vec{x}'_i$ refers to the result node i, and $\vec{x}$ is the current value of the nodes for node i or j, with e_{ji} referring to the edge between the two nodes. Finally, ϵ is a small positive constant, which is 10^{-7} by default, and is used to ensure the messages sent are always positive, even when the original result may be very close to zero. Ensuring a positive result is needed for some of the aggregation schemes, such as softmax. ReLU is as outlined previously. Overall, this means that for a given node, its features are updated based on this `GENConv` layer, taking as initial input the 64 inputs from the encoder for the node features and edge features, and then propagating this through until the end of the graph convolutions, building up information from the neighbours of the node per layer.

Supporting this layer is the encapsulating `DeepGCNLayer`, that takes care of running the layer normalisation, then the activation function, dropout and then the graph convolution, in that order. There is an optional skip connection that can then be applied after the convolution, if desired. The addition of this residual connection has been shown to help training of deeper GNN architectures [116], and similarly the ordering of the components can also have a drastic impact on the training of deep networks [117]. Because of this, the `DeepGCNLayer` takes the approach of applying the residual connection at the end, as it has been shown that activation functions such as ReLU can impede the representative power of deep models. Early on, this was tested, and the results achieved on other datasets seems to hold here for LArTPC data.

The final overall network architecture is outlined in Table 7.2, with the exact hyperparameters used.

[4]Logarithmic softmax is used over regular softmax due to increased numerical performance and better gradient optimisation, which are important for deeper models.

Name	Description
Input Graph	Round to nearest integer $K = 8$ for KNN
Input Encoders	Node Input Size = 6 Edge Input Size = 4 Output Size = 64
`GENConv` Operator	Number of Layers = 24 Input Size = 64 Output Size = 64 Softmax aggregator Layer-wise normalisation 2 MLP Layers
Normalisation Layer	Input Size = 24 Elementwise-affine
Activation Layer	ReLU activation
`DeepGCNLayer`	Combines the 3 layers Applies 10% Dropout `res+` Skip Connections
Output Encoder	Input Size = 64 Output Size = 2 Log-softmax output

Table 7.2: The chosen graph and network parameters for the DL shower growing, split into the input graph and then the various network layers.

7.2.3 Implementation in Pandora

With both the input graph structure and network implementation decided, there is a final step of how to most effectively use them both to implement a shower growing algorithm. The most important decisions are selecting the input cluster, which is required to generate the input graph, and how to most effectively use the classification score outputted by the network, to implement a form of shower growing.

Input Cluster Selection

An important part of the selected network structure is picking a suitable input to grow. The network is designed to answer the question "What clusters should I merge with the input cluster?", but this requires a sensible choice for the input cluster. Picking a cluster that is representative of the shower, such that its position and shape inside the shower aids the growing process, is a crucial step. If, instead, a small cluster on the far edge of a shower is picked, the network will need to work harder to find the shower it is part of. A secondary consideration is that Pandora already has a suite of tools and algorithms that are set up to build upon the early showers, as shown in Section 7.1.2, so building up the main core, the so-called 'trunk' of the shower, is important, as it enables the existing algorithms to keep working.

This process could be built in, with a simpler network being used to pick the most optimal starting node, before running the main network. However, for now, this employs a similar approach to the rest of Pandora, and instead each cluster has a score calculated for it, which is then used to select the best input cluster.

This score is designed to get the largest cluster that covers a large area, such that it should cover a large part of the shower. Additionally, the track-like and shower-like scores associated with each hit are also included. This gives a balance of selecting not just the biggest cluster, but the most representative one. The final score is calculated to balance these three values:

$$\text{Cluster Score} = \left(\frac{\text{Shower-like Total}}{\text{Cluster Size}} - \frac{\text{Track-like Total}}{\text{Cluster Size}} \right) * \text{Cluster Area}, \quad (7.2.2)$$

where the track and shower-like totals are the sums of the track-like and shower-

like percentages from each of the hits that makes the cluster, and the size and area are based on the number of hits and a bounding box area calculated using those hits.

This score is calculated for every cluster, then stored. The highest scoring cluster is used as the current input, as it should be the largest, most shower-like cluster that covers a large portion of the event. Figure 7.8 shows the variance in score for an example event, as well as how the selected cluster can change based on this score.

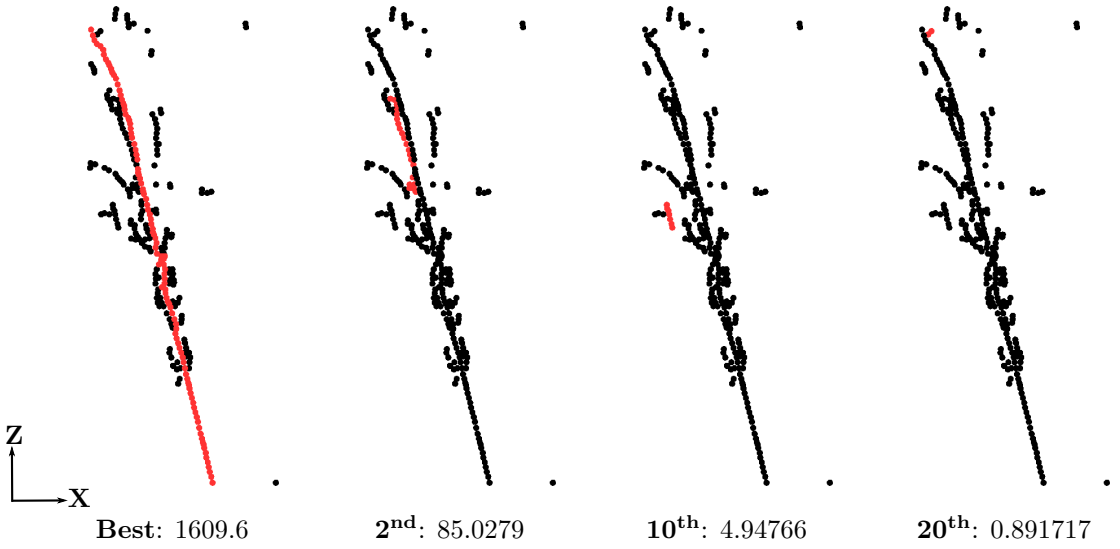


Figure 7.8: A comparison between the different potential inputs to the shower growing, for an example ν_e event. From left to right, shows the cluster with the best score, the second-best score, tenth-best score, and the twentieth best input. The best cluster is large and covers the full shower, in this example forming the 'spine' of the shower. The lower the score, the smaller and less shower-like the cluster is. Edges have been excluded from this graph to aid visualisation, and the nodes in red make up the selected cluster. The score for each cluster is given under the example, with a higher score being better.

Implementing Shower Growing

With each of the previous bits of work in place, the final step is to produce an actual Pandora algorithm, to complete the process of merging clusters and producing grown showers in an event. The core of this is based on the steps outlined previously, but there are still some additional questions that need to be addressed once the GNN is being used.

The first steps, after loading the shower-like clusters and running the scoring

method to select an input, is to build the graph itself. This proceeds as outlined earlier, producing a graph with hits averaged to nodes, and the desired node and edge-based features. This comes together to produce an input graph, with the selected best cluster tagged as the input. This graph is passed through the trained GNN, producing two percentages for each node that sum to 100%. These percentages relate to the strength of performing a merge, or not. Using these percentages is the first of the additional questions that a full implementation of shower growing must answer; How should the network output be used to merge together shower-like clusters? The outputs are given in a node-based way, rather than a cluster-based way, which means the node-based results must be summed per cluster before any decision is made. This decision can be as simple as a count of how many nodes think a merge is required versus how many do not, or can involve additional thresholds on the strength of the decision. Setting thresholds here is one simple way of tuning the aggressiveness of the cluster merging, as requiring the 'should merge' percentage is over 70%, 80% or 90% will impact how many merges are made overall. The shape of this distribution can be examined to guide this tuning process, with Figure 7.11 giving an example distribution for the scoring, though the distribution can change drastically if other parameters of the network are altered. The end result of this step is a per-cluster decision to merge or not with the input cluster, based on some tunable threshold.

These results then steer the cluster merging itself, with the input cluster and any merged clusters removed from any further consideration. However, this leads to the second issue: the shower growing in its current form is only capable of growing a single shower, where a typical LArTPC event may contain many showers. Instead, a decision must be made at this point: is the process complete? If there are only a few, small clusters left, or the previous shower growing step was unable to significantly grow the chosen input cluster, the process stops. However, if there are still many shower clusters and candidate input clusters, this whole process can be run again, with a modified graph that accounts for the clusters removed. These stopping criteria are based on the following checks:

- The number of hits added in the previous step.
- The number of remaining hits in the event.
- If the last shower growing step performed any merges.

- How many clusters were merged in the last step.

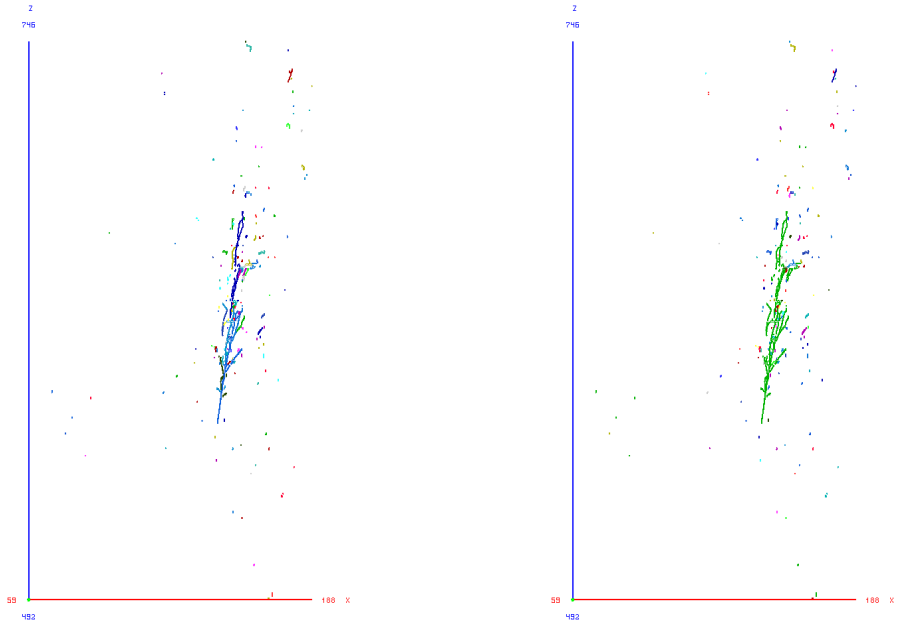
These four checks ensure that the shower growing is repeated only if there is a suitable number of hits remaining in the event, as well as the previous shower growing step was useful. If there is either a low number of hits left, or the previous growing resulted in a small change, it is unlikely the deep learning shower growing will be able to improve the event further.

Once every required iteration of the deep learning shower growing has been performed, the algorithm is complete. The existing shower growing is then run, as it can still be useful to merge small clusters that otherwise would be ignored by the strict checks for ending the DL shower growing. Additionally, as the current implementation of the DL shower growing drops merged clusters from any subsequent runs, it is possible for large showers to become split, if an inappropriate input cluster is chosen. Running the existing shower growing, which is designed to merge clusters conservatively, can help this issue by merging the grown individual clusters of the shower. This full process is shown for an example ν_e event in Figure 7.9, comparing the existing and new growing against each other.

7.2.3.1 Deployment within Pandora and LArSoft

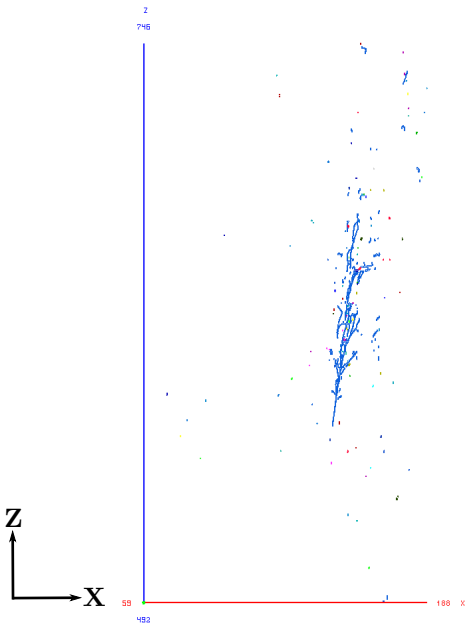
The entirety of this process is built into Pandora stand-alone, utilising `libtorch`. In general, this can then be called from LArSoft, as part of any existing reconstruction chain, transparently such that any user of Pandora is not aware that a DL algorithm is being used. However, one complication with this is that the default build of LArSoft utilises a fairly old version of `libtorch`, that is not supported by PyTorch Geometric. This necessitates using a stand-alone build of `libtorch`, as well as PyTorch Geometric, with which the stand-alone Pandora is then built against. In the future, once support for a newer build of `libtorch` exists inside of LArSoft, this work can be called more transparently.

As this work exists in C++, rather than Python, which is the more common deployment for deep learning models, work was needed to export the Python model to *torchscript*, which is a way of serialising a PyTorch model, such that it can be used in a process with no Python dependency. This required small adjustments to the modules built into PyTorch Geometric, to ensure that only methods supported by torchscript were in use. Similarly, additional work was needed inside of Pandora, to link against the new library modules, and to effectively use the tensor layout



(a) Before Growing

(b) Existing Growing



(c) DL Growing

Figure 7.9: The DL shower growing applied to an example ν_e event. 7.9a shows the initial clusters for the event, with different colours representing the different clusters. The existing shower growing, 7.9b, is able to group many of these clusters, but struggles around the edges and contains many small clusters inside it due to the five hit threshold. The DL clustering, 7.9c, shows more clusters brought together, resulting in an overall more complete shower.

of the graphs and graph network output, such as building a graph in the form expected by PyTorch Geometric, and parsing the results generated from running the loaded GNN.

As LArSoft is run across a wide array of computer architectures, as well as on various University computing clusters when used for large-scale productions, such as producing high statistics event samples, Pandora is not able to assume there is a GPU available. This restriction means that the network must be sufficiently constrained to have a reasonable execution time, which in turn impacts decisions such as the network and graph size, as well as the usage of the network. However, future work to enable GPU-as-a-Service (GPUaaS) inside LArSoft [118] could mean that larger networks are possible in the future, without sacrificing runtime on machines that do not have local access to a GPU. For example, running the DL shower growing with a graph that also accounts for the track-like clusters in the event, or the removal of rounding in the event could be possible with GPUaaS, as well as similar changes in other parts of Pandora.

Together, the deep learning-based shower growing should be able to improve the initial shower growing in Pandora. The DL shower growing can address most of the concerns with the existing shower growing, by giving a broader overview of the whole event, with an aim of clustering the majority of the shower, rather than just the core. The next step is then to train the network for this task of shower growing, and start to tune the various hyperparameters of the network to find their optimal value.

7.3 Network Training

Chapter 8 contains an in-depth look at the performance on simulated and real data, but before that, the network itself must be trained. The training process is important to understand how the network is learning, to verify that the training results in a more useful network that is able to accurately infer outputs, whilst also not exhibiting features of overfitting and learning specific features from the training dataset that will not generalise to unseen data. Additionally, the various parameters of the network need tuning, to find the most suitable for configuration for performing shower growing.

The dataset for the training process consists of around 48,000 ν_e events, which are then turned into actual training, validation and test samples. These are then

used to train the network, validate the network training process during training, and validate the network performance in full respectively. To start, the events are transformed into input graphs. That means not only producing a graph, but selecting an input cluster and the desired output for that input graph. That means a single event with one shower in, may make up tens of input graphs, with each graph consisting of a different selected input cluster. This obviously differs to the actual use case of the network, but allows a single event to be used for multiple training inputs, increasing the variety of input clusters the network is trained on. For the training set, a small number of random clusters are chosen as input from each event, so for an event with hundreds of clusters, only a small subset will produce input graphs. These graphs will be almost identical to each other, only changing the single node feature that indicates which is the current chosen input cluster. The random selection ensures that each event is only seen a few times, as training on every possible combination of input cluster could lead to overfitting much more easily. For the test and validation sets, every input cluster is tested, for an understanding of how the network performs when given any form of input. The only pre-selection used at this stage is size, where every cluster with more than five hits can be used as an input, which mirrors the same cut made in the existing shower growing. Additionally, only the collection plane results were used, as this provides the clearest input to train from, and there should be no physics difference between a shower in one plane to the next.

Overall, this results in around 100,000 training samples, 8,000 validation samples and 80,000 test samples. Whilst this may seem like an odd split between test and training sets, the actual number of underlying ν_e events in the training set is 44,000 whereas there is only 4,500 events in the test set. This is because every single input cluster in each event is tested for the test set, rather than a 10% sampling and a cut of at most 35 input clusters in the training set case. This means that despite the close number of samples, the training set contains a much wider range of electron neutrino events, which is more useful to learn from.

With this dataset in place, the actual training can take place. The training took place on either an Nvidia V100 GPU at The High End Computing facility at Lancaster University, or an Nvidia Tesla T4 GPU available via The University of Manchester. In both cases, the network was trained using a batch size of 16, the Adam optimiser [119] and a PyTorch scheduler. The Adam optimiser is used to update the network itself, in place of something like the classical stochastic gradient

descent. Adam is broadly better than most other optimisation algorithms, able to converge faster and requiring less parameter turning, such that it is usually the recommended default. A scheduler in this context is an optimisation option that allows the adjustment of the learning rate hyperparameter whilst the network is running. Specifically, the `ReduceLROnPlateau` scheduler was used, which reduces the learning rate by a factor of ten once the learning has stalled. This can allow for a larger learning rate early on, but quickly dropping it once needed, or help in cases where the network gets stuck.

Once the dataset is loaded, the actual training process can begin. The training set is split into mini-batches and passed through the network, losses are calculated and then propagated backwards, and the Adam optimiser takes a step. The loss for training was the negative log likelihood loss, which is a common loss function to use in classification problems. This was chosen as it is commonly used here, but also because it allows slight weighting of the loss calculated through the use of an optional weight tensor. This weighting allows a manual rescaling weight to be applied to each class, meaning that the network loss can be tuned somewhat. In this case, as ultimately completeness is the goal, the loss function was weighted to make getting the classification of the current shower wrong impact the loss more, compared to getting the classification of a node unrelated to the shower wrong. Once this is complete for the training set, the validation and test sets are evaluated without any updates to the network, to get an understanding of the current performance. The training and validation loss are stored, as well as the performance from the test set, before the process repeats. This process is continued until no more meaningful progress is achieved, by observing changes to the loss function. The result is a trained network, with some basic metrics calculated such as the correctness, completeness, and purity for each shower in the test set, as well as the training and validation loss. These values can be plotted per epoch to understand how the network learnt and which epoch should be taken and used. An example of these values can be seen in Figure 7.10, which shows the training and validation loss trending downwards over time, indicating the network is learning. However, it can be seen that there is also a point where only the training loss drops, indicating that in this training process it is possible that a feature specific to the training set has been learnt, which does not generalise to a similar performance drop in the validation set, which is undesirable. It should be noted that these training metrics are not directly comparable to the cheated

or later reconstruction metrics, as they lack the follow-up mop-up algorithms and similar, as they are not available to run outside of Pandora.

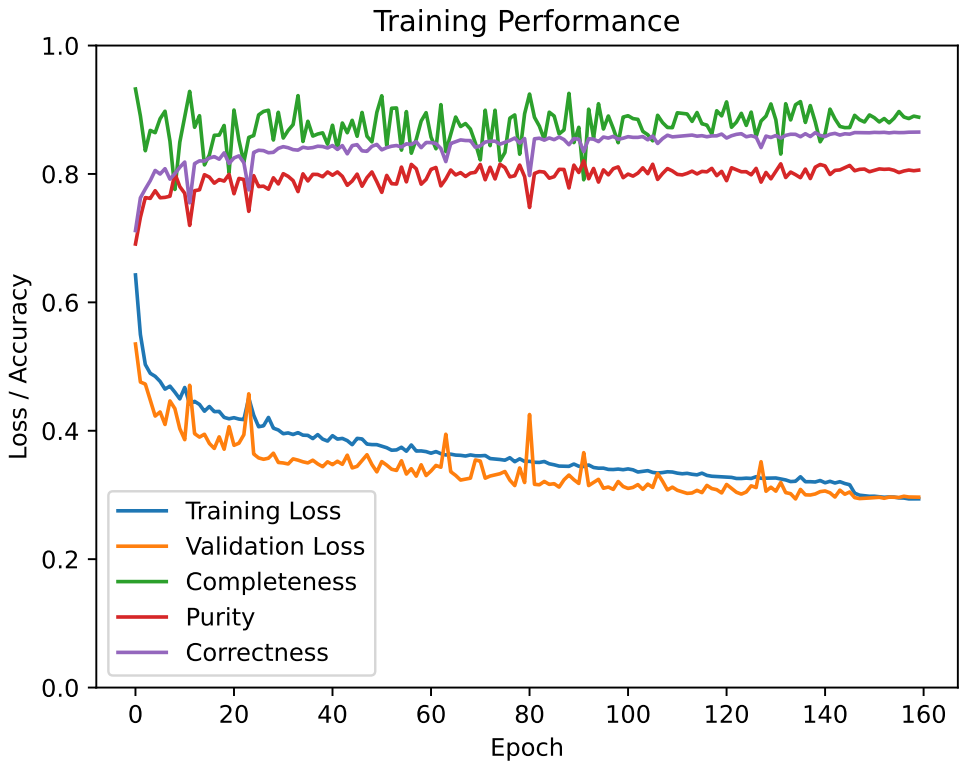


Figure 7.10: The training and validation loss against the epoch number, with associated test set performance numbers. The training and validation loss can be seen trending downwards over time, with a jump at around epoch 145. Similarly, the completeness and purity, as well as the overall correctness of the output (that is, the number of nodes that were predicted correctly), trends upwards and gets smoother over time. For this model, the epoch before the drop around 145 were chosen, as the drop at 145 is not accompanied by a similar drop in the validation loss, indicating that a training set only feature may be behind the drop. The validation loss being lower than the training loss is a somewhat common phenomenon, that can be potentially attributed to many different things, including the presence of dropout, the differences in the training and validation data, learning rate scheduler impact and more.

7.3.1 Hyperparameter Tuning

The selected network and graph architecture has many tunable parameters, including the various layer sizes, graph rounding figures, distance and angle thresholds, the chosen network order and activation functions and more. Each of

these parameters needed to be optimised, to achieve the best overall performance for neutrino events.

Performing this optimisation is made somewhat harder as the usage of the network and the training process do not align perfectly. That is, the training process uses a randomised input rather than a score-based selection, and the training process stops after a single iteration of growing, which means that the full event performance is not available. As a consequence, the network performance was analysed using per-shower results for completeness and purity calculated during training, and then networks that performed similarly in training were tested against each other inside Pandora. This provides a better understanding of their full performance in a more realistic setting, with multiple iterations of shower growing and an intelligent input cluster selection, rather than a randomised input.

This process was completed using a variety of tools, first using manual tests to get an understanding of the impact of the parameter changes, before swapping to Ray Tune [120] for more in-depth hyperparameter tuning. Ray Tune is a Python library that allows for intelligent hyperparameter tuning to be performed, allowing for parallelised tuning that also utilises intelligent early stopping and more, to speed up the tuning process compared to a more basic tuning algorithm such as trying every combination of values in full. Ray Tune was selected after a brief exploration into optimisation libraries, and was chosen for being the simplest to use, as well as having good compatibility with the various other packages and hardware in use for the training process. Additionally, features that were added to the graphs early on proved detrimental when running on full neutrino events, such that the final parameter tuning decision was to disable them. All the values listed here are listed together in Table 7.2.

Many of the tunable parameters are correlated, such that they were tuned together. For example, changing the graph rounding figure has an obvious impact on the number of nodes in the graph, reducing the information available to learn from. However, a perhaps less obvious change is that the lower number of nodes means that a cluster may be connected to more unique clusters now, as there are fewer nodes per cluster, so less rounding with the same number of edges per node means more connections to unique clusters. These additional connections can actually alter the performance, due to the flow of information in the message passing layer. Similarly, as there are fewer nodes the final "should merge" versus "should not merge" count is different, and is impacted. As there are fewer nodes

per cluster, the decision is made based on fewer network outputs. A similar effect happens with changing the value of K in KNN, the distance thresholds for edges or around vertices and more. This means many of the values have a suitable middle ground, both for performance against speed, but also enabling or inhibiting information flow.

Tuning the graph rounding figure showed that most values of rounding were viable, most likely due to the new edges offsetting the information loss from less structure in the graph. Tests were performed using different values of rounding, including no rounding, and the rounding set to one, two, five, and ten centimetres. The final tested models used a rounding value of one cm, which offers a small amount of event simplification, equal to rounding the decimal values to the nearest integer. This ended up giving the best performance, though even with aggressive rounding, the performance was still competitive. A rounding of one was chosen as it overall gave the best result, at least in terms of overall event completeness.

The edge KNN with distance cut-off were tuned simultaneously. Reducing K meant that less information was passed around the network, but also that connections would only be made to the most immediate neighbours, which are likely the most useful. Similarly, the distance cut-off is a balance between allowing sensible clusters to be joined, bridging small gaps in showers, whilst not allowing edges to be made across many meters of the detector. The final chosen values for this was $K = 8$, and no distance cut off. It was found that whilst the distance cut off helped in some artificial test cases, such as single-particle events, in neutrino events overall it led to fewer merges being made, which is not desirable. An artificial limit also means the network is not able to learn this cut off on its own, though this may only be possible with sufficient training.

As well as the edge-based parameters, there are further tunable parameters that enforce strict limits on the angular agreement between clusters near the vertex. The distance of this strict angular cut needs to be balanced to fit for larger events, without overwhelming small events by enforcing the limit everywhere. Similarly, the angular agreement needs to be set to a reasonable value, where clusters over the angle threshold are legitimately unlikely to be related to other clusters, whilst accounting for the wide range of angles that clusters can be relative to each other whilst all coming from the same parent particle. This was eventually set to a strict 0.2 radians agreement, but only for a small region around the vertex, of 0.1, which once scaled accounts to about 50 cm at a maximum. This allows strict rules

around the track-like region that high-energy showers have, but then relaxes it quickly after the shower starts to branch.

The final major tunable parameters were those of the network itself, not the input graph. This includes the learning rate, which was chosen to be low (0.0003) and then lowered further when learning stalled. The number of layers was initially tuned much lower, but eventually runtime performance testing showed that even when set to 24 layers with 64 hidden nodes in the MLP in each GNN layer, the overall network inference time was of order a few hundred milliseconds total per plane, which is acceptable compared to the existing growing and if compared to the full run time for the reconstruction chain as a whole, especially in ProtoDUNE-SP which has a much longer overall runtime.

Throughout this process, alongside the training-based metrics of completeness, purity, accuracy and more, an eye was kept on the underlying score distributions output by each different network. These help form part of the decision-making process for picking a model, as a well-trained model should produce a score landscape that covers a wide-range of values, with a badly trained model mostly returning values close to 50%, indicating that it is not confident in the decisions it is making. Figure 7.11 gives an example of how the "should merge" distribution can look like for an example network, with the "should not merge" result being the inverse.

As mentioned, the better performing of these tests were run inside of Pandora, as running inside of Pandora's full reconstruction chain gives a more accurate understanding of how the shower growing performs, as there can be multiple iterations of shower growing, as well as the follow-up mop-up algorithms. An example of some of the tested models are outlined in Table 7.3. Throughout this process, the reconstructed completeness and purity was used as the main indicator of model performance, as this metric is easily understood as part of the development process. As such, those models that achieved a balanced of high completeness and performance in the training process were chosen to be tested further inside of Pandora. However, as outlined previously using reconstruction-based metrics can potentially obscure deficiencies in a model if showers are missing from the event, so it is not always the most appropriate metric to use. Instead, using event efficiency as a second metric produces a different optimisation target compared to only using shower completeness and purity. For this reason, the final reconstruction metrics in Chapter 8 show the results for two models, one tuned

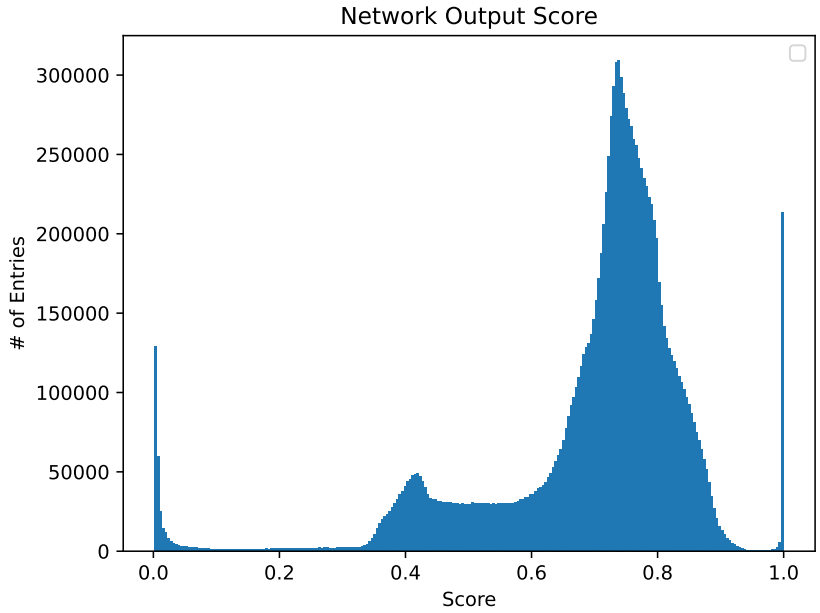


Figure 7.11: An example distribution of the output scores from the network, showing the "Should merge" score, with the "Should not merge" score being the inverse. Plots like this help understand how a certain training of the network is learning when to perform merges, as well as being a useful tool of where to put merge thresholds to have the largest impact.

for particle quality and a second for event efficiency.

The final models were chosen after tuning these hyperparameters and comparing the metrics that are discussed next. Using these two trained models, a full performance analysis can be performed, to get a deeper understanding of how the shower growing works in practice. When training the network, the results only consider a single pass of the network, and include no additional mop-up steps, which also have a large impact on the shower's completeness and purity. Getting an understanding of how these additional steps alter the observed results, as well as the final results that an analyser would see is important, and the next task.

	Completeness	**Purity**
Existing Growing	87.4%	87.1%
DL Growing	91.9%	81.0%
Simple model, 75% merge threshold	87.9%	81.4%
Simple model, 85% merge threshold	87.1%	84.2%
Simple model, drop small clusters	87.0%	84.5%
Increased vertex protection radius	86.9%	83.2%
Even rounding, low distance cut-off	86.9%	85.2%
Even rounding, high distance cut	86.9%	84.0%
Drop small clusters (<= 5)	86.9%	85.5%
Drop small and distance cut off	87.1%	86.3%

Table 7.3: Examples of different tested final model performance at the far detector. First, the baselines are given with the existing growing and the chosen DL growing configuration. Next, various tunes are given to give an idea of how the performance can change with different tuning. The numbers for the chosen models are explained in more detail later in the chapter.

8

Deep Learning Shower Growing Performance

C3PO: *"R2D2, you know better than to trust a strange computer!"*

Star Wars: Episode V – The Empire Strikes Back

With an optimal graph and network architecture found, the network needs to be tested against the existing shower growing, to understand if it can improve events, and if so, what impact it has. This can be achieved by looking at high statistics metrics, to get an understanding of how the shower's completeness, purity change with and without the new shower growing, as well as the overall event efficiency and if what topologies are most impacted by this new approach.

After optimising and training the network, the reconstruction performance was assessed using a higher-statistics and independent set of simulated electron neutrino data from the HD FD. Here we can get an understanding of how the new and existing shower growing algorithms compare, directly at the shower growing step, and as well later on at the end of the algorithm chain. Following this, there will be a first look at how the network performs on real data using ProtoDUNE-SP e^+ test beam data, comparing MC against real data. This gives an idea of how a network built and trained on DUNE far detector data can be applied to a different detector still using LArTPC technology, as well as a look at the agreement between simulated and real data, which is crucial as methods are developed for the far

detector using simulated data.

8.1 DUNE FD Performance

The cheated study performed on ν_e far detector events showed that there is a clear performance gap between the current growing and a perfect growing. The deep learning-based growing is built to help close this gap, but the actual achieved performance needs to be benchmarked to get an understanding of what gap exists after the changes to the growing, and if the new growing is actually beneficial when applied to full neutrino events. This performance study was performed on around 18,000 ν_e events in the full simulation of the DUNE horizontal drift far detector, with this chain outlined in full in Chapter 4.

As before, there are two useful areas to assess the shower reconstruction performance: directly around the shower growing itself and the final reconstruction performance at the end of the full Pandora algorithm chain. Benchmarking both before and after the growing gives the most direct comparison of what the new shower growing is doing to the shower tagged clusters, which is useful for understanding what overall impact it is having, as well as being a useful development aide. However, improving the mid-chain performance is not useful in of itself[1], if that improvement does not persist until the end of the reconstruction chain and make real improvements to the final reconstructed particles. This is where an assessment of the full reconstruction chain is useful instead, giving a more realistic impression of the shower growing and the follow-up algorithms which improve the clustering further. In both cases, the performance is evaluated for all reconstructed showers, and the performance metrics are also calculated separately for both the largest cluster per shower, and the overall largest shower in the event, with the size here measured based on the number of hits in the cluster or particle. The largest shower in the event is an important part of any neutrino selection, which is why it is an interesting shower to target, whilst also giving a clearer view of the impact of the shower growing in the before and after plots, which are otherwise clouded by the vast number of showers. The largest cluster per shower is used, rather than simply looking at every shower for a similar reason, as well as applying a small amount of normalisation to the results.

[1]Performance here meaning reconstruction performance such as completeness, purity or efficiency. Increasing the mid-chain runtime performance is always useful.

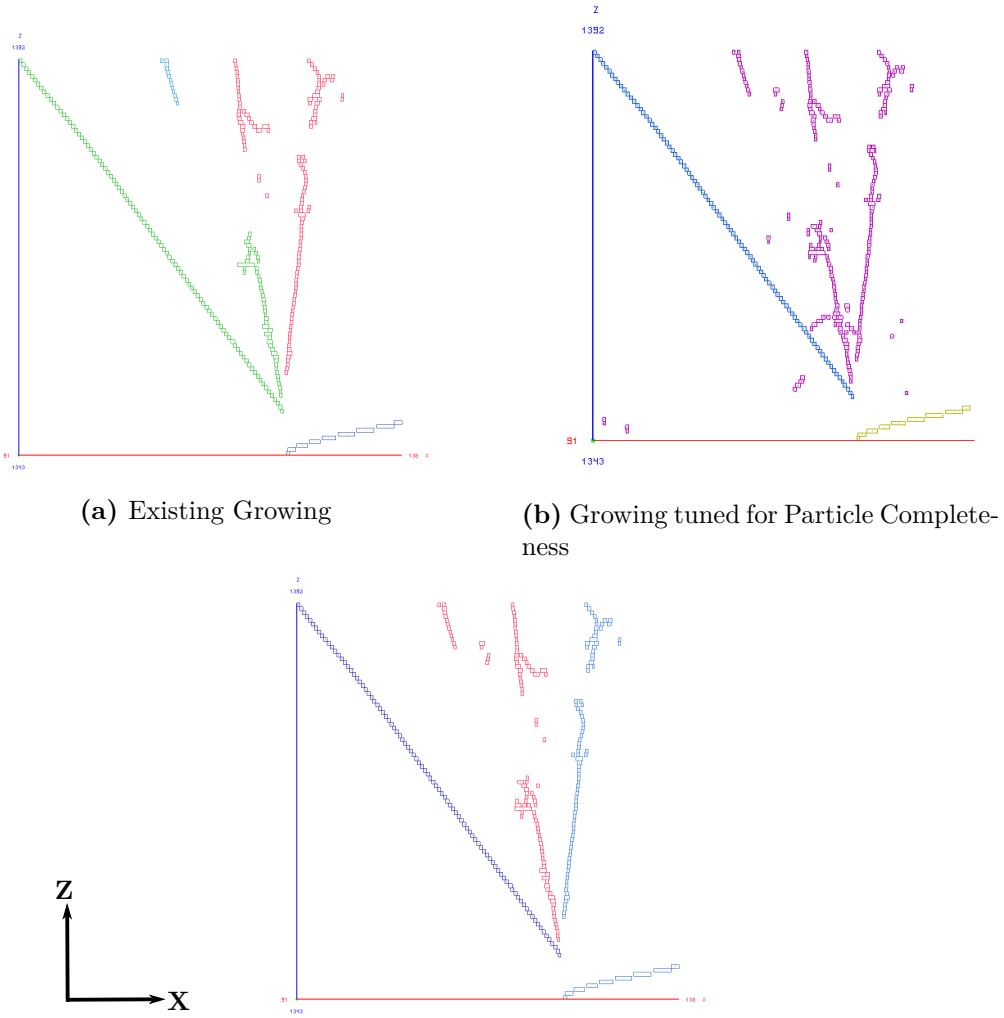
As well as including plots from around the shower growing itself and the end of the reconstruction chain, both reconstruction and Monte Carlo first plots are needed as each provide different information. The reconstruction first plots are a useful tool, informing how the reconstructed objects look, their completeness and purity. However, a reconstruction first approach does not include those particles that are completely missing from an event, if the particle has been swallowed by a larger shower in the event. Looking at both sets of plots gives an understanding of how the particles that are reconstructed look, whilst giving context of any missing particles. This can be useful when an algorithm is able to improve its reconstruction performance at the cost of removing other particles from the event. The impact of this depends on the analysis undertaken, and the number of hits in any missing particles in the event.

Two models are shown for the final reconstruction quality part of these results, a baseline model that prioritises the completeness of the clusters, and then a follow-up tuned model that utilises the same trained network, but with a greater emphasis on event efficiency[2]. The efficiency tuned models only difference is an increase in the required percentage to perform a merge from 75% to 85%. That is, when the network outputs two percentages that sum to 100% for "should merge" and "should not merge", the minimum required percentage for performing a merge can be tuned higher, meaning fewer merges are made overall. An example of how this can impact a real event is shown in Figure 8.1, where enforcing a higher confidence on the decision means that a photon is not lost. This value can be tuned further, alongside similar values for the cluster-based merge decision built from summing each node result, but these two tunings were chosen to show the difference in results when optimising for particle completeness or event efficiency.

8.1.1 Performance of 2D Shower Growing

An initial assessment of the DL-based shower growing algorithm shows the direct impact that the shower growing had on the initial clusters of the event, going from hundreds of very small, very pure clusters to fewer, larger clusters without introducing too much contamination from unrelated clusters. These plots show the direct impact of the shower growing, before the follow-up mop-up algorithms improve the clusters further. However, these numbers and plots do not give a

[2]Completeness, purity and efficiency are outlined in Section 4.3.7.



(a) Existing Growing

(b) Growing tuned for Particle Completeness

(c) Growing tuned for Event Efficiency

Figure 8.1: Example ν_e event containing two photons, with the two different growing tunes. The existing growing and completeness tuned DL growing produce merged photons, where the two distinct photons in the event are merged, either fully or partially. The existing growing also fails to pull in a fragment of the shower, instead producing a third shower. In the efficiency tuned DL growing, fewer merges are made, resulting in an overall more accurate reconstruction. Shared colours between the two showers and the tracks in the event are only due to the automated colour selection, no showers and tracks were merged in any of the configurations.

	Completeness		Purity	
	All Showers	Largest	All Showers	Largest
Before Growing	07.7%	28.5%	94.0%	93.2%
Existing Growing	17.1%	52.4%	93.1%	90.0%
DL Growing				
Efficiency Tuned	19.0%	61.2%	92.3%	81.1%
Completeness Tuned	31.9%	73.2%	87.7%	76.5%

Table 8.1: Comparison between the existing growing and the DL growing, for before and after the shower growing step. The equivalent numbers for the smaller cheated test ν_e dataset can be seen in Table 7.1. The new shower growing when tuned for particle quality is able to achieve a higher overall completeness for the largest shower in the event when compared to the cheated growing, which most likely indicates that smaller showers from a different parent MC particle are being merged in the DL shower growing case, which are instead ignored in the perfect, cheated algorithm. This is not the case when the model is instead tuned for event efficiency, which helps protect some of these smaller showers, at the expense of having lower gains compared to the existing growing.

realistic view of the full reconstruction, such that they need to be taken in context with the final reconstruction performance plots too.

Figure 8.2 shows that the performance of the shower growing on simulated far detector data can approximate the improvements made by the cheating in Section 7.1.2. In fact, the completeness of the cheated growing shown in Table 7.1 is slightly lower than the completeness tuned deep learning-based growing, indicating that when tuned for particle completeness, the deep learning growing is perhaps growing too aggressively and merging with some clusters that may not be optimal.

This can be seen by comparing the achieved purity, as listed in Table 8.1. In the largest cluster per shower case, the existing performance has a higher number of showers at 100% completeness, further indicating that some smaller showers may be missing in the newer growing. This can be investigated more directly at the end of the reconstruction chain by evaluating the reconstruction efficiency. It is, however, encouraging to see that the deep learning-based shower growing is able to achieve a performance improvement of around 15% to all showers and over 20% when looking at only the largest shower in the event, when compared to the existing reconstruction at the same point in the reconstruction chain. The efficiency tuned model performs as expected, lying between the two results, incorporating some of the improvements of the shower growing, whilst also being careful to avoid performing too many merges.

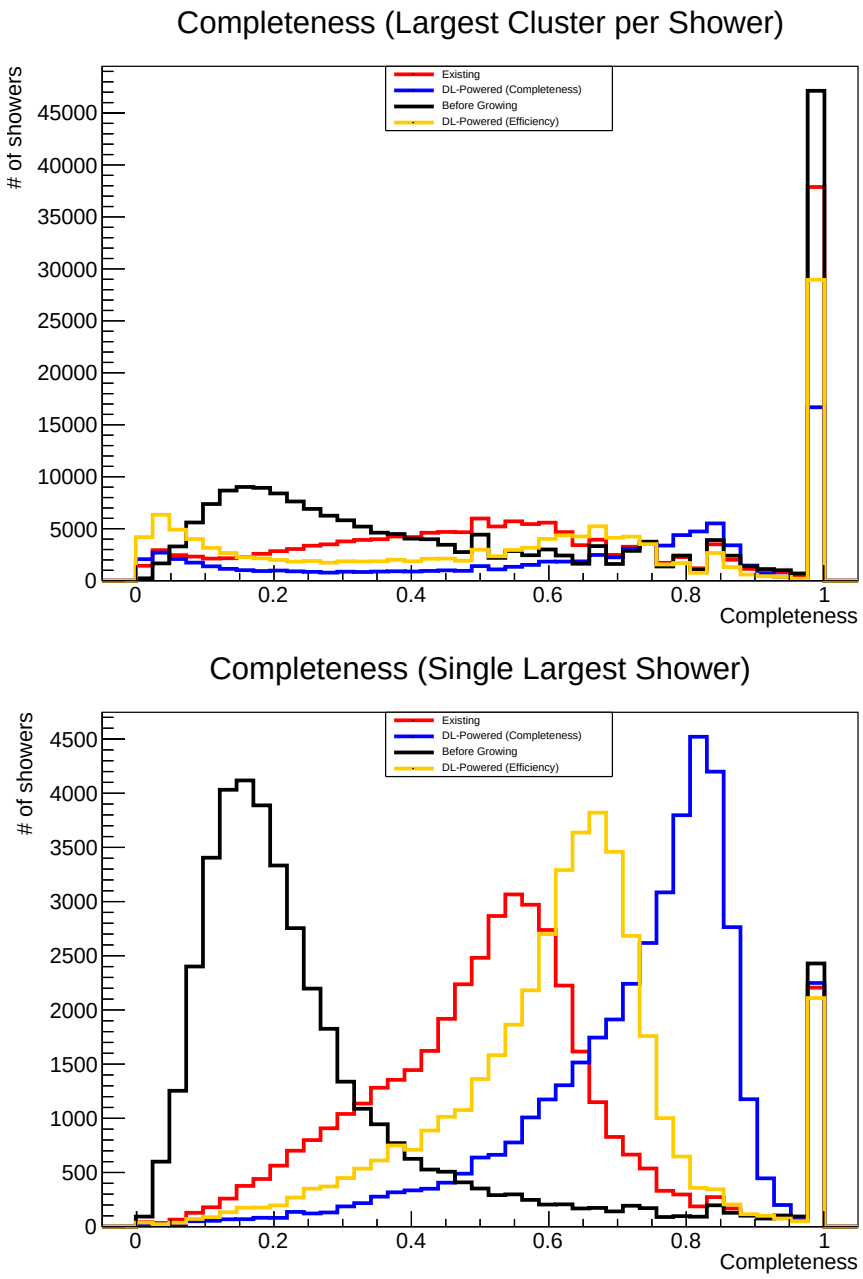


Figure 8.2: A comparison between the existing and both tunes of the DL-based shower growing, showing the difference in the initial cluster growing. The before growing stage is the same for every algorithm, as the initial starting state is identical for them. When tuned for particle completeness, the shower growing shows reductions in low completeness showers and increases at higher completeness. This is also true for the efficiency tuned result just will a less extreme impact, as it is more cautious when merging small showers, resulting in less overall merges. In both plots, the reduction in showers at 100% completeness is due to the largest shower changing from very small showers (around 5-15 hits total), to a much larger one.

8.1.2 Performance after Full Reconstruction

Whilst looking at plots before and after shower growing is a useful tool for development, it does not give an accurate representation of the final reconstruction performance. For that, the performance at the very end of Pandora is needed, after growing and every other mop-up algorithm that improves the 2D clustering. This will reflect the particles that will be used in analyses. Due to the distance between the shower growing and the end of the reconstruction (shower growing is performed very early), it is not as useful as a development tool, as it is difficult to parse how changes to the shower growing have impacted the final cluster metrics, except for relative comparisons against the existing shower growing.

Table 8.2 shows the final reconstruction performance achieved with both tunes of the deep learning shower growing, as compared to the existing shower growing. In both cases, both tunes of the DL shower growing are able to translate into real, end of reconstruction performance. The completeness tune is able to achieve an increase of over 16% for particle completeness when looking at all showers, whilst maintaining a high particle purity. This gap is reduced when looking at the largest shower in the event, but still pushes performance up by 4%. When instead tuned for event efficiency instead of particle completeness, it can be seen that the expected result is achieved, with a higher merge threshold reducing the number of merges, which in turn results in overall lower completeness. This results in a smaller increase of 7% in completeness for all showers, and just over 1% when looking at the largest shower. This is expected, and the performance of this tune should be reflected more in the event efficiency results.

Figure 8.3 shows the final completeness for the largest cluster per shower and the largest shower in the event at the very end of Pandora, to get an idea of how the completeness distributions change between each of the three results. As expected, the particle completeness tuned deep learning growing achieves the highest completeness in both cases, with the efficiency tuned result sitting between the completeness tune and the existing growing, offering a more restrained middle ground.

It should be noted, that like in the before and after plots, there is a larger number of showers in the existing distribution of the largest cluster per shower plot, indicating that fewer showers have been reconstructed using the DL-based shower growing, with this being a larger concern for the model tuned for particle completeness. This is shown in more detail in Figure 8.4 that shows the

	Completeness		Purity	
	All Showers	Largest	All Showers	Largest
Existing Growing	71.2%	87.4%	85.2%	87.1%
DL Growing				
Efficiency Tuned	79.0%	88.5%	82.4%	81.9%
Completeness Tuned	87.5%	91.9%	81.2%	81.0%

Table 8.2: Final reconstruction performance for the existing growing and the DL growing. The equivalent numbers for the smaller cheated test ν_e dataset can be seen in Table 7.1. The DL shower growing is able to make distinct changes to the shower clusters, that are not made up by the later mop-up algorithms, resulting in overall higher completeness and comparable purity, when tuned for event efficiency and particle completeness.

	Efficiency		
	All Showers	Electrons	Photons
Existing Growing	90.4%	95.8%	85.3%
DL Growing			
Efficiency Tuned	83.3%	91.0%	67.0%
Completeness Tuned	79.6%	89.1%	54.7%

Table 8.3: Final reconstruction efficiencies for the existing growing and the DL growing. Here it is more obvious where issues are occurring with the new growing approach, with photons being lost much more in the new algorithm. This loss of photons likely aides the improvements in the completeness, as small hard to reconstruct particles are absorbed into the larger showers of the event. It can be seen here that when tuning for event efficiency, a notable improvement is made across all three channels when compared to the shower completeness tune.

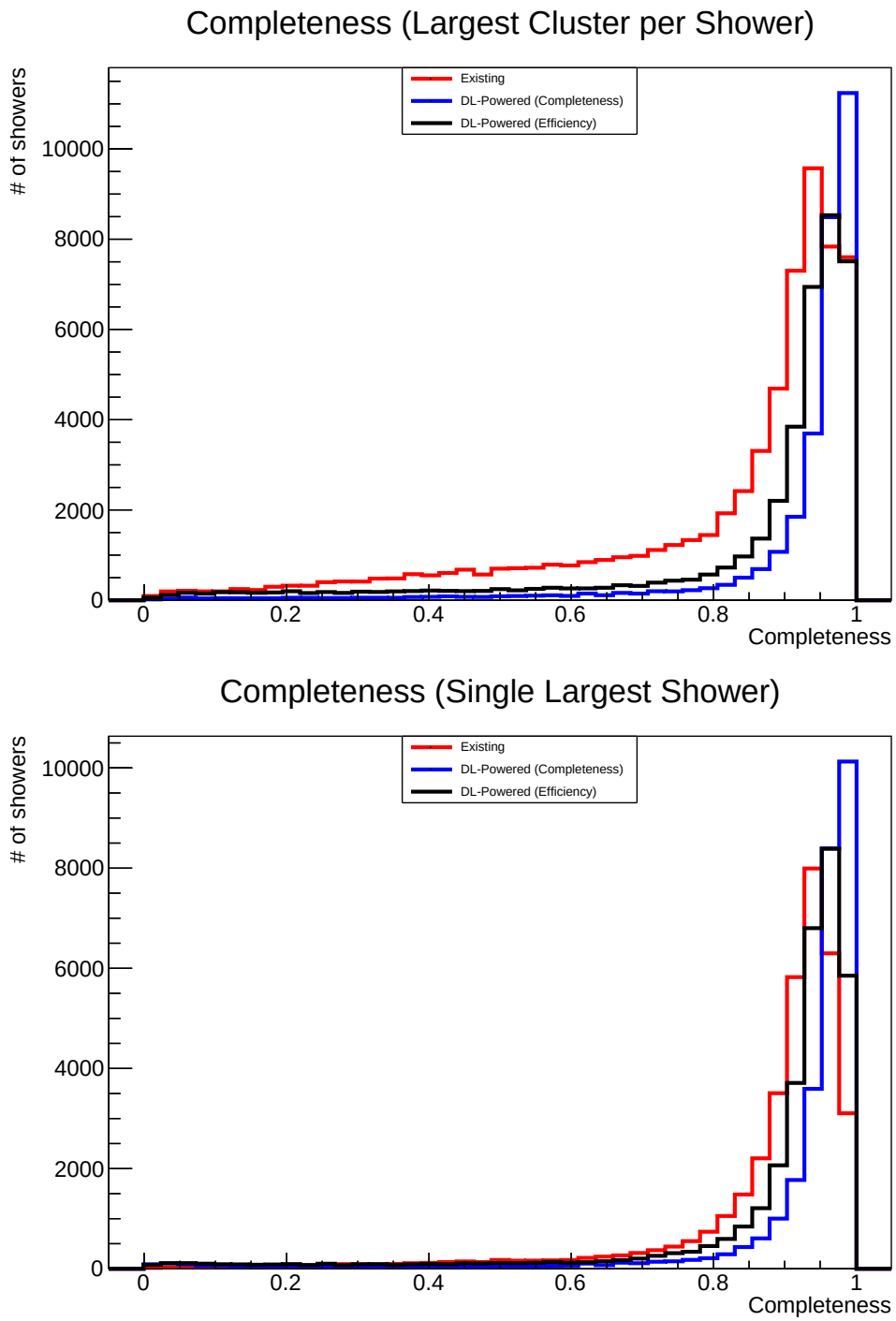


Figure 8.3: Final reconstruction performance for the existing and DL-based shower growing. In both cases, the deep learning-based growing improves the shower completeness, resulting in an overall more complete shower. As expected, when tuned for particle completeness, there is a greater emphasis applied to completeness such that an overall higher completeness is achieved, at the cost of event efficiency, as shown in Figure 8.4.

reconstruction efficiency. As defined in Section 4.3.7, reconstruction efficiency is the number of MC particles matched to at least one reconstructed particle over the total number of MC particles, with some quality cuts on the reconstructed particle match to ensure the matched particle is at least 15 hits, purity greater than 50% and completeness greater than 10%. This gives an overall event efficiency of 79.6% and 83.3% when looking at every shower for the completeness and efficiency tunes, respectively, compared with 90.4% for the existing growing. Focusing on the efficiency tuned growing, this can be split into 91.0% efficiency for electrons and 67.0% for photons, as compared to 95.8% and 85.3% for the existing growing. This shows that the main cause of the loss of efficiency is related to loss of photons in the event, which is the case for both tunes of the network. It does also show that tuning the model after training can impact the final results considerably, with a small threshold change resulting in a model with reasonably higher event efficiency across all ranges of hits.

However, the metrics do show that for the showers that are reconstructed, the completeness is notably higher, for all reconstructed showers, including the largest showers in the event, whilst maintaining a high purity. This implies that any showers that are being merged, reducing the number of showers in the events, must be small as they do not have a large overall impact on the reconstructed purity. This trade-off, between completeness and efficiency, is a difficult parameter to tune, with individual physics analyses preferring different balances depending on their physics goals.

8.1.3 Potential Improvements

There are many further adjustments, improvements, and extensions that could be made to this new shower growing. It has been shown with the previous sets of results how much impact the final level of tuning can make. Some of the more interesting or important tuning or network improvements are outlined here. Making these decisions also requires a deeper connection to the underlying physics, rather than only using reconstruction-based metrics. Ideally, the changes outlined here could be tested against a full physics analysis change, to better understand the final impact of any change to the physics, which would help inform the network and graph architecture based on physics results, rather than reconstruction results.

One easy change would be the addition of an extended training set, with specific neutrino interaction types in. For example, as Figure 8.4 shows the largest drop

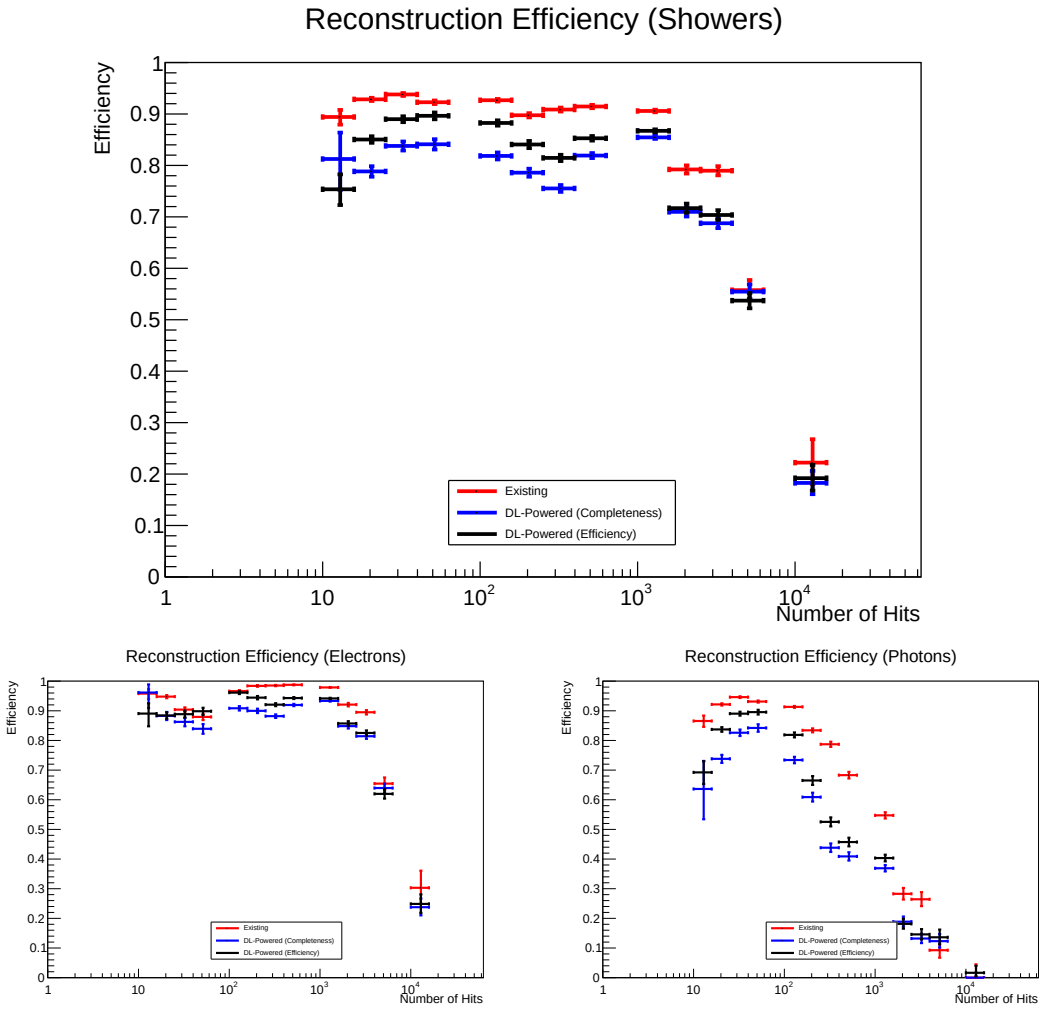


Figure 8.4: A comparison of the reconstruction efficiency between the existing and new shower growing. First, the total efficiency is shown, against the number of reconstructed hits. Then, split into electron and photon showers individually, to help show where the inefficiency is mostly coming from. In both cases, the efficiency of reconstructing mid-sized photons is lower overall, implying that they are being lost to merging or other issues. A low efficiency for photons with more hits can be seen in both the new and existing growing, though there is an even lower efficiency with this new algorithm. However, when tuned for event efficiency, there is an overall efficiency increase across the range of shower sizes, compared to the particle completeness tune.

in efficiency for photons, specific training data focused around photons could be used to augment the electron neutrino training data. This would help ensure the network is trained on a significant set of events containing photons, rather than only picking electron neutrino events with no further selection based on how the neutrino interacts in the detector. It is also possible that specific considerations need to be made for photon showers, which is a common balancing problem in neutrino event reconstruction; the growing process for a large, many hundred of hits primary electron shower may be very different compared to two small photon showers produced with a small opening angle from a π^0 decay. This is an area where the somewhat strict distinction into track-like and shower-like may benefit from further augmentation, as there is further distinction inside the shower-like category.

Alongside tuning the existing model, there is a whole suite of potential improvements, most of which require a greater availability of graphics cards to speed up the network inference, as otherwise the network's runtime would begin to become a significant percentage of the Pandora runtime. Some of these changes are simple extensions of the existing network, such as relaxing the rounding used or the value of K in the KNN edge algorithm. A larger change would be the inclusion of track-like clusters in the shower growing graph, which provides a lot of useful event-level information. For example, two showers with some angle between them may be ambiguous, but the inclusion of a track between them may make the problem much easier by providing a natural splitting point between the two showers. Similarly, leaving in the clusters that were grown in previous runs of the network provides additional context to the event, which could make subsequent growing steps much easier.

8.2 ProtoDUNE-SP Performance

Running on ProtoDUNE-SP data is an interesting test case, allowing an understanding with real LArTPC data. We can use the positron test-beam data that ProtoDUNE-SP has collected to validate how the DL shower growing performs, both in how it generalises to a new detector, but more importantly how work on simulated data can be applied to real data, utilising a source of real electromagnetic showers. For these tests, only a single tune of the deep learning shower growing was used, as initial testing showed essentially no difference between the models at

ProtoDUNE. The tune used was the model focused on shower completeness, to highlight the differences seen between the existing and new growing in a model that showed the largest differences on far detector simulated data.

However, it does introduce a number of differences compared to the target far detector data. These differences mean that running the same network may not be optimal, due to the differences between a surface-based test beam LArTPC and an underground neutrino beam LArTPC, but the broad approach should still work as the electromagnetic showers should look similar in the two detectors, as ProtoDUNE-SP does use the same components as the horizontal drift far detector will. This means that the broad approach, can be verified to work as expected at ProtoDUNE-SP even if a specific network and tuning may be required for real usage there. It should be noted that as the shower growing was built around growing electromagnetic showers resulting from neutrino interactions, the DL-based growing will only be applied to showers resulting from the test beam, rather than cosmic ray showers, as these showers would most likely require different features and graph structure. An example ProtoDUNE-SP event is shown in Figure 8.5, giving an example of how different the events look to far detector data.

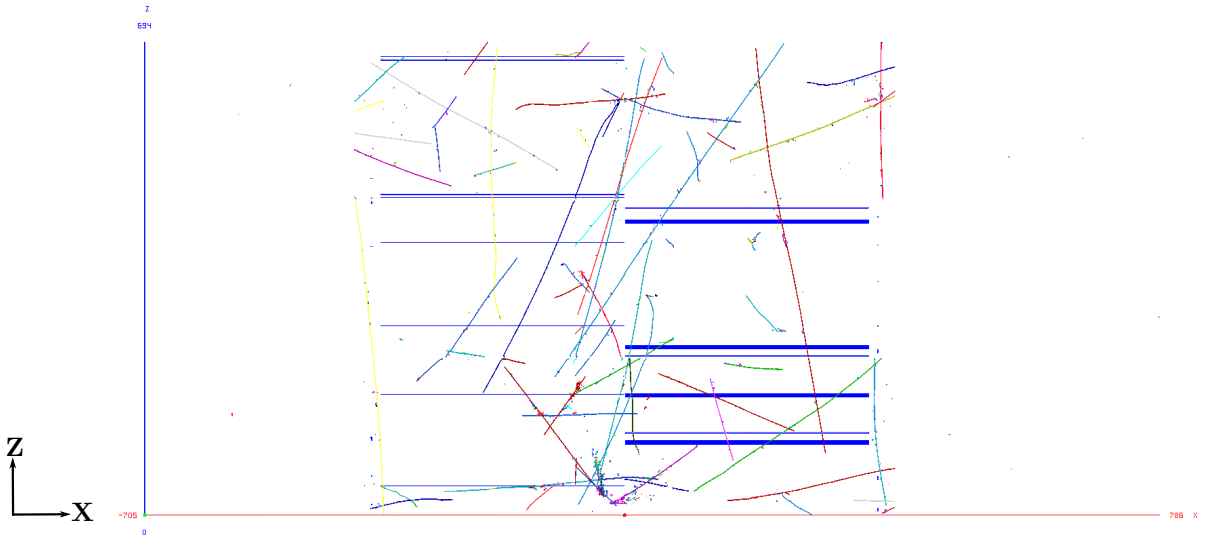


Figure 8.5: An example 1 GeV positron event at ProtoDUNE-SP, reconstructed with Pandora. The various blue horizontal lines relate to wires in the detector that have issues, so are excluded from the reconstruction. A number of cosmic rays can be seen throughout the detector, resulting in lots of overlap in 2D, complicating the reconstruction process. This is especially true at the bottom middle of the event display, where a shower can be seen intersected with many tracks.

The data used consists of around 18,000 MC 1 GeV e^+ simulated test beam particle events, and 1,000 1 GeV e^+ real test beam events. For the MC events, a pre-selection is applied that selects only events with a positron as the test beam particle based on the underlying simulation truth information. For the real data run 5809 was used, an electron-enhanced 1 GeV run from November 2018, with a pre-selection to select positrons. In the real data case, beamline instrumentation data is used to tag beam particles, based on time-of-flight measurements and reconstructed momenta taken from trigger profiles and profile monitors that sit in the beamline. This information can be used as a LArSoft filter to restrict the dataset to only positron events. By using data only from runs that have been validated and tagged by the collaboration as a 'good run', we know the detector was operating in a sensible and stable manner[3]. This ensures that overall, there is a high purity sample of test beam particles to test the new shower growing on. These datasets were sufficient to show some indication of the new shower growing performance, whilst also being quick enough to iterate on, rather than trying to be representative across the entire spread of beam energies that ProtoDUNE-SP used. The 1 GeV sample was used as it is the sample with the largest available statistics, as well as being well-understood and processed by the collaboration already.

To assess the real data, plots of reconstructed properties such as the shapes and sizes of the showers provides an interesting data point to compare the real test beam data against the simulated data, to get an understanding of how it compares and if the results from applying the new shower growing to real data are similar to those achieved on MC data.

First, an understanding of both the ProtoDUNE-SP physical and software differences is explored, to provide some additional context for why and how the two experiments differ.

8.2.1 ProtoDUNE-SP Differences

The ProtoDUNE-SP detector is a very different detector to the final far detector modules. However, first it is easiest to talk about the things that are the same. To test the construction methods, as well as benchmark them, the APAs are the same and are prototypes for the real APAs that will be used in the far detector. Similarly, the DAQ which is responsible for the actual reading out of the detector is also based

[3]For example, this means data is not used from periods of known low LAr purity or from dates where there were issues with the detector instrumentation, beam line or more.

on prototypes of the far detector designs. The biggest physics difference here is that both ProtoDUNE experiments are surface-based test beam experiments. This has a few overall impacts, at least in the context of reconstruction. Being in a test beam, rather than a neutrino beam, means that the interactions being reconstructed are very different: We have broadly simpler interactions, with a much smaller range of energies, coming from a much more well-defined point, the beam plug. Because the sample used contains only interactions of 1 GeV, the events are simpler and much more similar, compared to the far detector which can have events varying by tens of GeV. There is also additional complications from the 'beam-halo' particles. These beam halo particles include particles from interactions in the beam line, other particles that were not focused by the beam line magnets, and additional particle decays and out of time particles. This is another form of additional complexity, causing additional interactions in the detector, which reconstruction algorithms must deal with. An even bigger difference is the fact that the detectors are surface-based, meaning they are continually being hit by cosmic ray interactions. These cosmic ray interactions are read out the same as any beam interactions, and provide a complicated background that must be removed or ignored by any reconstruction code, such that the target beam interaction can be identified[4]. Finally, there is the smaller differences that come about due to having a real detector. For example, rather than simulated noise models based on previous LArTPC experiments, you have real noise, as well as detector defects if readouts are malfunctioning in some way.

All these differences add up to produce a reconstruction problem that is very different to a far detector module. For this reason, a different approach is needed to target ProtoDUNE-SP events.

8.2.2 ProtoDUNE-SP Consolidated Reconstruction

As ProtoDUNE has cosmic ray interactions, Pandora runs in a mode where it first reconstructs the clear cosmic ray interactions in the event, and then reconstructs the rest under both a test beam and cosmic ray hypothesis, to find the most suitable reconstruction for each of the remaining particles. To achieve this, after the most obvious cosmic rays have been removed, the remaining particles are split

[4]Cosmic ray interactions do form a very useful part of the reconstruction, and can be used for countless calibration tasks and more, so accurate reconstruction is useful and required for them too, even when most physics analysis target only the beam particles.

up into 'slices', regions of wire number and time that should ideally contain only a single particle. We can then reconstruct each of these slices independently, testing them as both a test beam interaction and cosmic ray interactions. Each slice goes through a cosmic ray optimised reconstruction chain, and a test beam optimised reconstruction chain, before a final selection is performed to pick the most appropriate reconstruction hypothesis at the end.

For this analysis, the shower growing is being evaluated before, after and at the end of each slice reconstruction in the test beam hypothesis only, rather than the full event reconstruction, as this shows most clearly the impact of the shower growing. This does mean that potential cosmic rays can be reconstructed with the DL shower growing under a test beam hypothesis, then thrown out in favour of the better cosmic ray hypothesis. As there is no way to know the result of the selection in the middle of the reconstruction chain, these cosmic ray interactions end up forming a background of difficult to reconstruct interactions that the network was not trained on. As the before and after data is not easily accessible once the slice hypothesis has been chosen, it was deemed easier to include this small background, as it is realistic that any shower growing algorithm will need to work for both true test beam interactions, and any ambiguous cosmic ray interactions.

As a combination of this different running mode, and the aforementioned simpler interactions with a much tighter spread in energy, the actual shower cluster completeness and purity pre-growing is very different to that seen in the far detector. Shower-like clusters in ProtoDUNE-SP are created with much higher overall completeness and purity, most likely due to the lower overall energy compared to the far detector.

These compromises aside, the true value of having ProtoDUNE-SP data is that ideas can be tested on real data, so having an understanding that the algorithm performs the same, even if the running mode is not ideal, for data versus MC is still useful. Ensuring that the algorithm performs the same on simulation and data at ProtoDUNE-SP can start to give us confidence that the same will be true when running on DUNE far detector data in the future.

This analysis was performed outside LArSoft, due to the version of `libtorch` it bundles being too old to support the GNN library used for this work. Instead, the data was exported via LArSoft to a form suitable for Pandora to read in. This exporting of hit-level data does somewhat constrain the types of analysis possible, as the rest of the ProtoDUNE-SP analysis work is implemented inside LArSoft,

	Completeness		Purity	
	All Showers	Largest	All Showers	Largest
Before & After Shower Growing				
Before Growing	61.5%	68.9%	71.5%	75.5%
Existing Growing	70.5%	72.0%	71.2%	71.2%
DL Growing	73.4%	73.7%	69.0%	62.8%
End Of Pandora				
Existing Growing	69.3%	74.0%	69.4%	56.8%
DL Growing	70.0%	75.8%	69.0%	49.8%

Table 8.4: Completeness and purity of the DL and existing shower growing at ProtoDUNE, split into the before and after stages, then the final reconstruction performance. The most interesting part of this is in comparison with Tables 8.1 and 8.2, where it can be seen that the pre-growing completeness is much higher, indicating that a different method of shower growing may be needed at ProtoDUNE, if it is deemed that improving the shower growing at ProtoDUNE-SP is worth it through cheating studies. The mop-up algorithms also perform differently due to the presence of slicing in the ProtoDUNE configuration, which may explain the drops seen moving from the growing stage to the end of the reconstruction.

meaning an extended analysis of how the new shower growing impacts physics results is not easily possible. For a similar reason, it is not possible to easily study systematic uncertainties in this analysis outside LArSoft.

8.2.3 Performance of 2D Shower Growing at ProtoDUNE

Table 8.4 shows the overall performance of the new deep learning-based shower growing at ProtoDUNE-SP, on the 18,000 simulated ν_e events. If we first look at the before and after performance, when looking at all showers, there is a moderate 3% increase in completeness over all showers, and a minor increase of 1% when looking at the largest shower in the event. In the all shower case, this is at a similar level of purity, but there is a more significant drop in purity of around 8% when looking at the largest shower. As mentioned previously, it is important to note how different these numbers are compared to Table 8.1, with the before growing completeness being much higher, whilst also having much lower purity. This means that on average at ProtoDUNE-SP, showers are much more complete and overall should require less growing compared to events at the far detector.

These results can also be seen in Figure 8.6, comparing the existing growing

and the DL growing, with the before growing step as reference.

Figure 8.6 highlights a difference in the two detector configurations: whilst the far detector has very low completeness showers pre-growing, ProtoDUNE-SP has fairly higher completeness showers, even before growing. This may be a consequence of the different interactions, or because of the configuration the shower growing is running, both outlined in Section 8.2.5.

Moving to the final reconstruction performance, there is a much more minor increase overall, with around 1% for all showers, and focusing on the largest shower. Similarly to the before and after results, the purity when looking at all showers is comparable to the existing growing, but there is a similar level of purity loss when looking at the largest shower, around 7%.

Figure 8.7 shows more definitively that improving the shower growing is not a priority at ProtoDUNE, compared to the impact it has at the far detector. However, this is perhaps not surprising based on the larger differences between the two detectors, despite sharing the same technology and ProtoDUNE-SP operating as a test bed for the far detector. Subtle improvements are made to the largest shower in an event, but when looking at the largest cluster per shower, the difference is minimal, and gets even smaller when looking at every shower. This is also the case when looking at the overall event efficiency, where there is essentially no changes.

8.2.4 Reconstructed Shower Property Comparison

The main benefit of having data from a prototype experiment for reconstruction tasks, is being able to use real data, and perform MC versus data comparisons. For the shower growing, this means we can check how the shower growing performs on real and simulated data, checking that it is consistent between the two. This helps give confidence that algorithms produced on far detector simulation, but tested on ProtoDUNE data should work once real far detector data becomes available, though with some inevitable tuning.

This study is performed only on the very final results, rather than before and after shower growing, partially as the reconstructed properties for a partial shower is not intuitive, but also to ensure that we are only comparing true tagged test beam particles, rather than potential particles that are reconstructed under both a test beam and cosmic hypothesis. This ensures that the comparisons made here are only using fully built showers, with a decided test beam versus cosmic

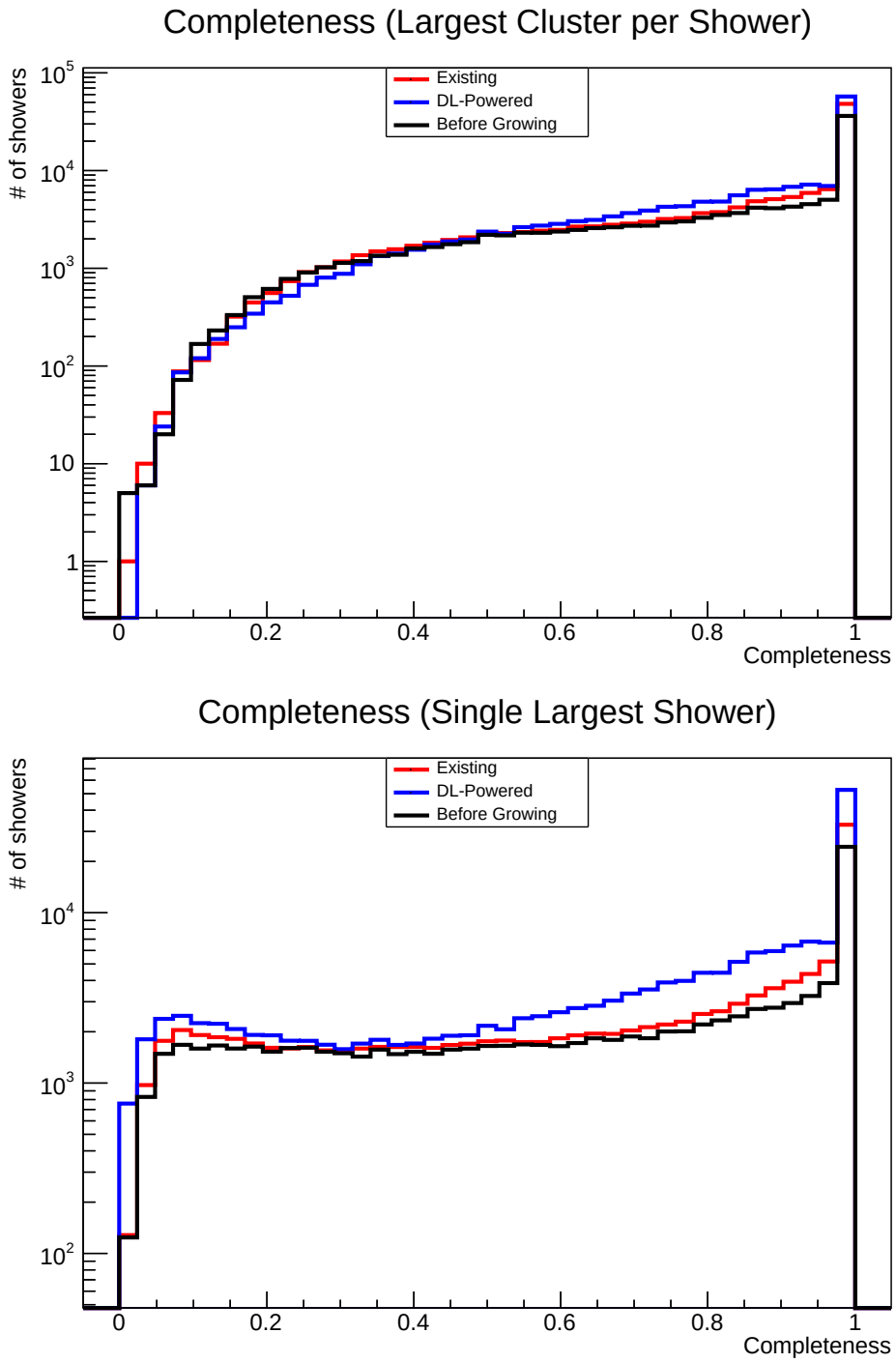


Figure 8.6: A comparison between the existing and DL-based shower growing at ProtoDUNE, before and after growing. Unlike Figure 8.2, the before growing contains a large number of high completeness showers, indicating that there may be larger clusters in this sample, pre-growing compared to the far detector. The new shower growing can improve the performance, but at the cost of introducing more low completeness showers. When looking at the largest cluster per shower, a smaller difference is seen.

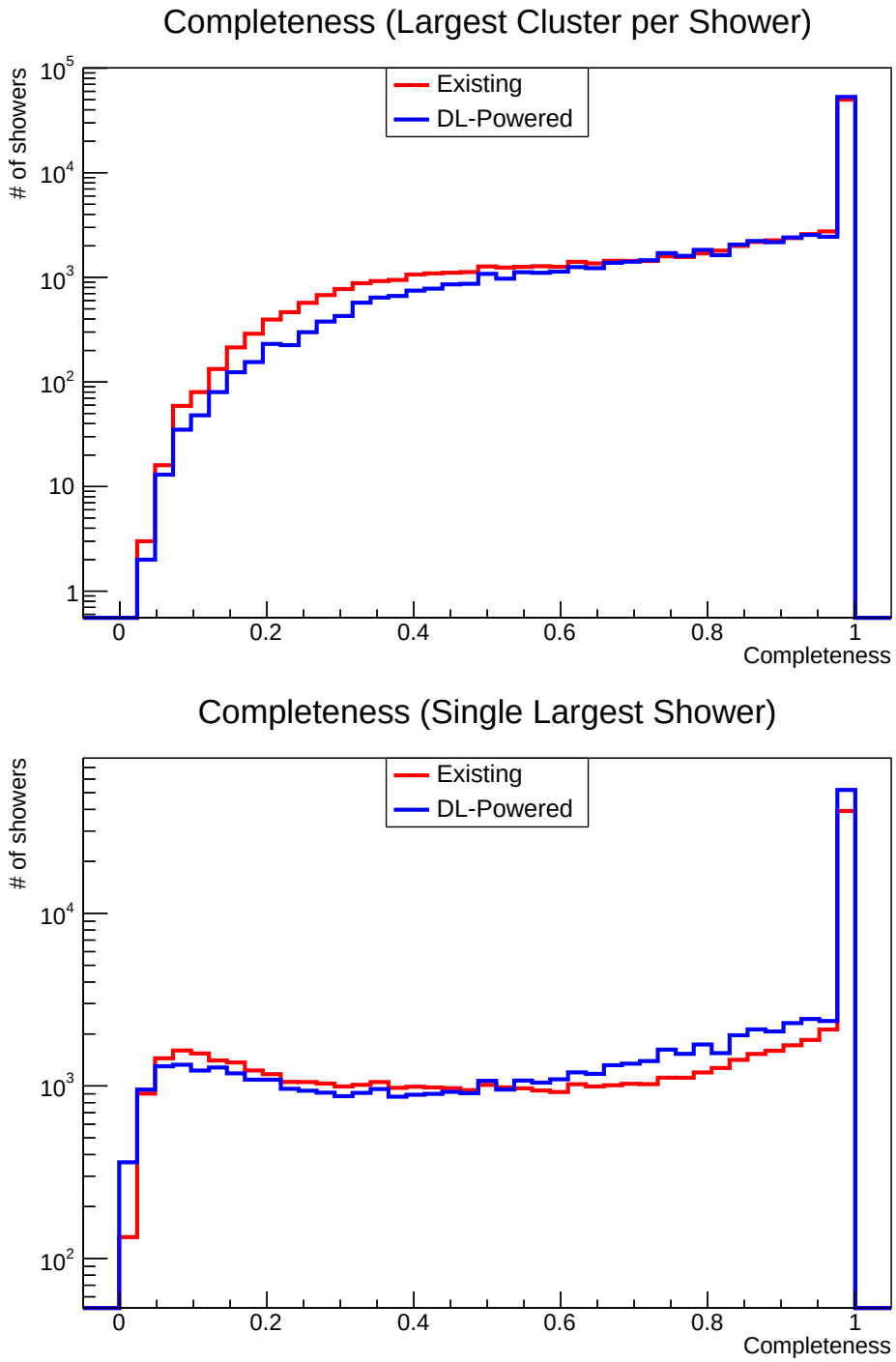


Figure 8.7: A comparison of the final reconstruction performance at ProtoDUNE. Here it can be seen more obviously that the inclusion of improved shower growing is not a big driver of performance at ProtoDUNE-SP, compared to the impact that it has at the HD FD. There is a small improvement for the largest shower in the event, but this reduces significantly when moving to the largest cluster per shower, and then closer again when looking at every shower-like cluster in the event.

hypothesis, with intuitive shower properties. Without this, looking at plots of calculated shower widths and shower primary directions would be muddied by non-obvious results for any non-shower cosmic rays that are being tested through the neutrino hypothesis. If instead, we wait until the end of Pandora, we are only looking at fully reconstructed and tagged test beam particles, which should produce metrics with more sensible shapes.

To assess the real test-beam data, we can compare reconstructed shower properties between the two datasets. The shower properties being analysed are the shower's size, based on the average cluster size, as well as features built using the final shower cluster. The size, shape, and direction of the fully reconstructed 3D shower is assessed by applying a PCA fit to their reconstructed 3D hits. This yields primary, secondary and tertiary axes for the shower. The shower properties are then calculated from these axes as followed:

- **Shower Length**: The shower length is simply calculated using the length of the PCA primary axis.
- **Opening Angle**: The opening angle is a ratio of the primary axis length to the secondary and tertiary axes:

$$\theta = \arctan \frac{\sqrt{\|\mathrm{PCA}_2\|^2 + \|\mathrm{PCA}_3\|^2}}{\|\mathrm{PCA}_1\|}$$

  with $\mathrm{PCA}_1, \mathrm{PCA}_2, \mathrm{PCA}_3$ referring to the primary, secondary or tertiary axes. When performed in 2D, the tertiary axis is set to zero.

This process is performed on both the simulated and real ProtoDUNE-SP data, and we can then compare the results. This is repeated for both the new, DL shower growing and the existing growing where interesting, to highlight cases where the shower growing is failing, or there may be more underlying issues in the simulation.

First, we compare the 2D shower growing on ProtoDUNE-SP MC against real data for the collection view, as shown in Figures 8.8 and 8.10, with the collection view chosen as it should be the view most free of noise. The results show a reassuring agreement between the two data sets. Here, we can see that broadly, there is reasonable agreement between the simulated and real data, such that the DL growing is performing similarly on both sets of data. The shower length and opening angle looks as expected, with most showers in both data and

simulation having an opening angle around 10 degrees and a length of around 160 cm. Similarly, the bias in the reconstructed X direction fits the expected distribution based on the entry point of the test beam at ProtoDUNE, and most showers go in the direction of the beam in the Z direction. The backwards going showers, with a Z direction closer to -1 are usually small showers that have a low number of hits, which causes the direction correction factor to be wrong, pointing the shower the wrong way. Similarly, the spread in length and angle can depend on when the shower begins to shower, as well as contamination from cosmic rays and background test beam interactions.

Figures 8.9 and 8.11 show the same properties but for the existing shower growing. It can be seen that there is good agreement across the board for the existing shower growing. If we compare the 2D shower length plots for the new and existing shower growing, we also see good agreement across all four distributions. However, when looking at the 2D opening angle, the new shower growing algorithm is producing a much larger range of opening angles, with a much less sharp peak around 10 degrees, as compared to the existing shower growing. It is also interesting to note that in both cases, with and without the new growing, the data sees a systematically larger opening angle compared to the simulation, perhaps indicating a deficiency in the simulated data. Finally, it can be noted that the reconstructed 2D X direction plots for both the new and existing growing show some small disagreement for values just below zero, perhaps indicating a small issue with the simulation, rather than a failure of the new growing causing differing results.

Following from this, we can also check that the final showers made in 3D have comparable reconstructed properties, by performing the same steps as in the 2D case, but on 3D clusters. Figures 8.12 and 8.14 show similar agreement as in 2D, even after the 3D reconstruction steps and further processing of the clusters. The remaining shape differences may be explainable due to missing effects in the simulation, with Appendix C giving some additional figures that show the agreement for the existing shower growing, rather than the new DL growing, which may explain further limitations.

Finally, figures 8.13 and 8.15 again show the same distributions, but when utilising the existing growing. These allow for some interesting comparisons, as there is certain features, seen in both 2D and 3D as well as in both versions of the growing. For example, the agreement between data and simulation for

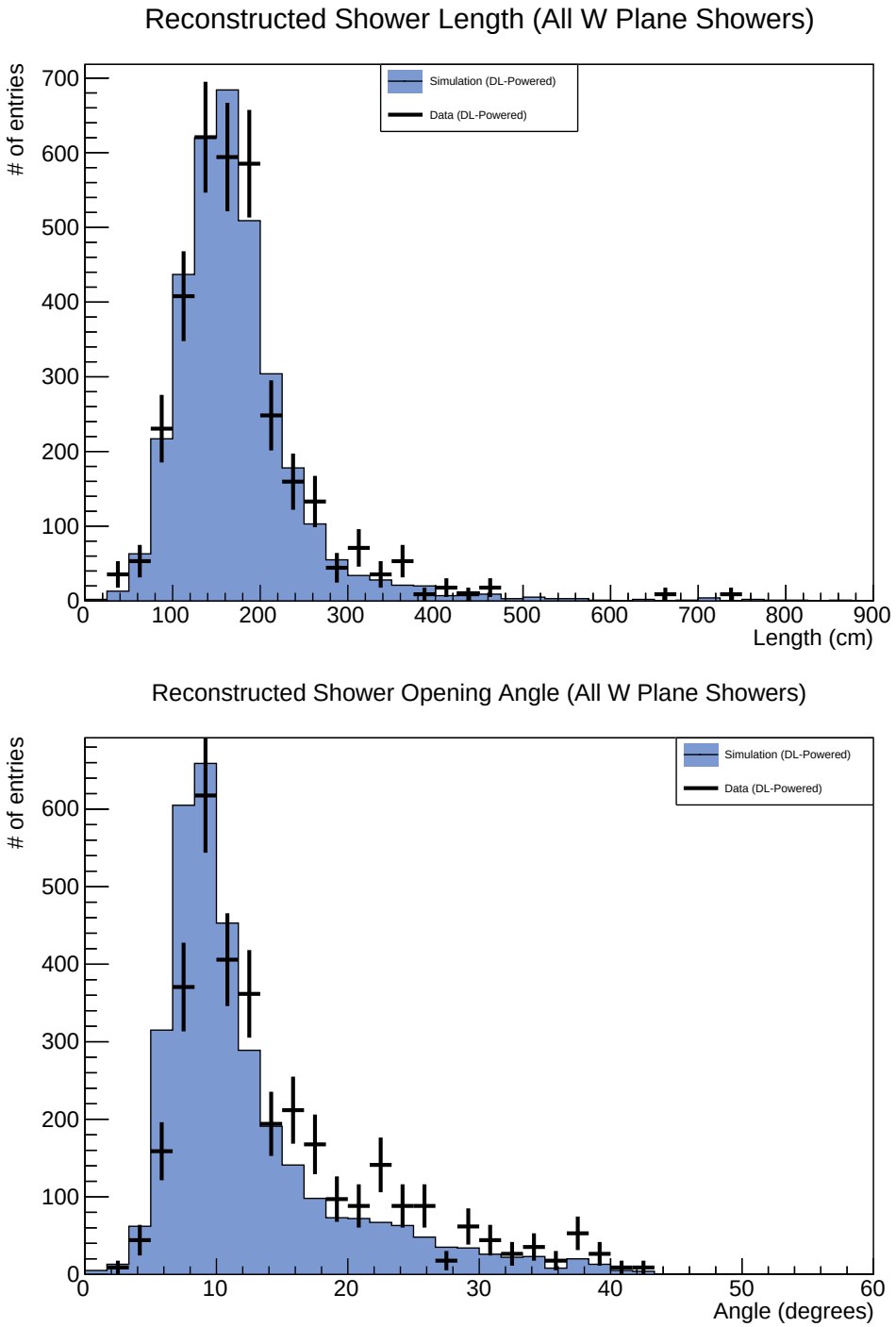


Figure 8.8: Comparison of the reconstructed 2D shower length and opening angle on ProtoDUNE simulation and data, using the new shower growing. We can see reasonable agreement between the two. Here, the length is the length of the PCA primary axis, and the opening angle is calculated using the inverse tan of the ratio between the secondary and primary axis. The DL growing is able to perform the same on both simulated and real data, even with new detector effects not encountered in training.

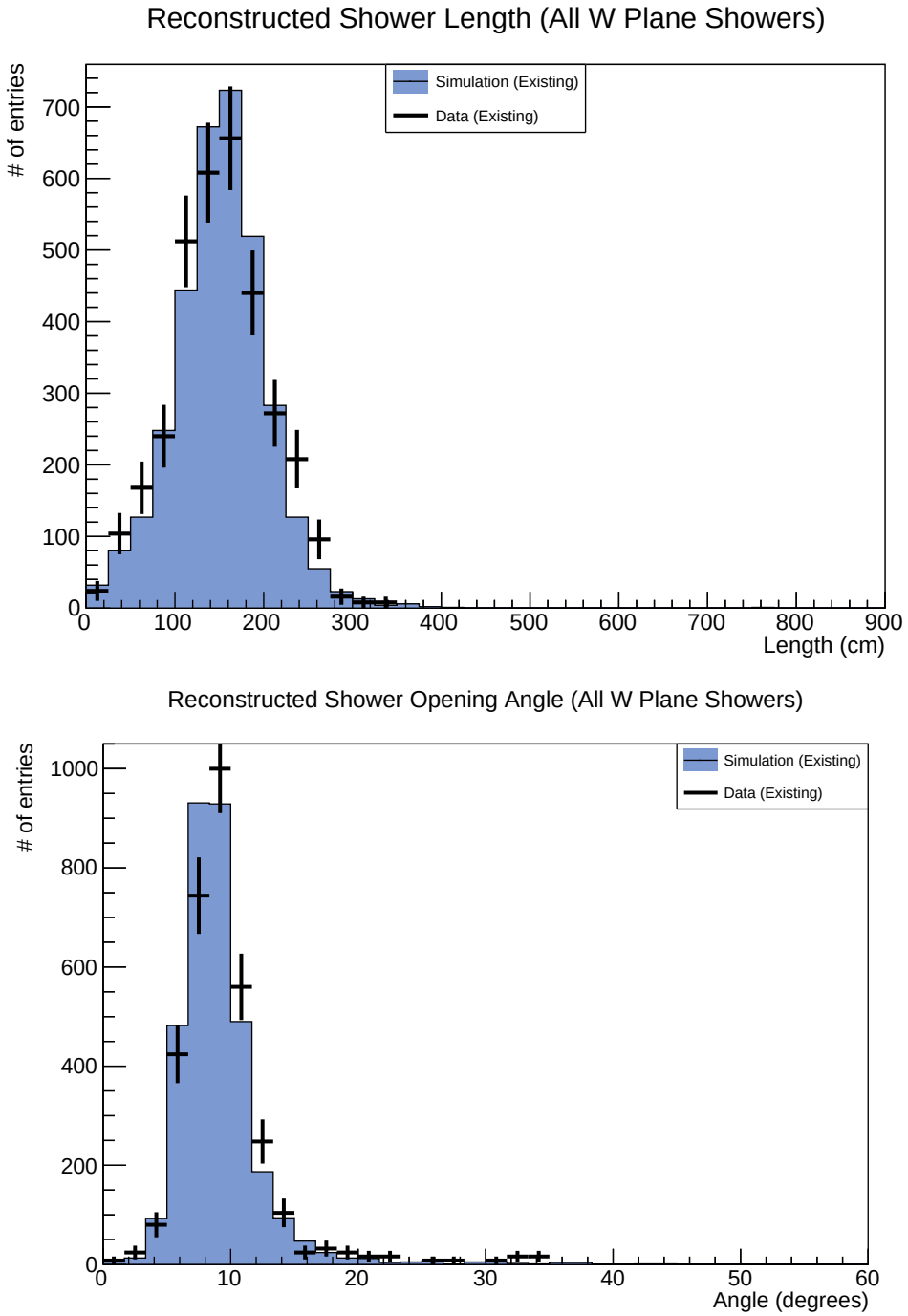


Figure 8.9: Comparison of the reconstructed 2D shower length and opening angle on ProtoDUNE simulation and data, using the existing shower growing. We can see good agreement between the two. Like the DL case, the length is the length of the PCA primary axis, and the opening angle is calculated using the inverse tan of the ratio between the secondary and primary axis. We see similar agreement between these two shapes, as we do in the DL case.

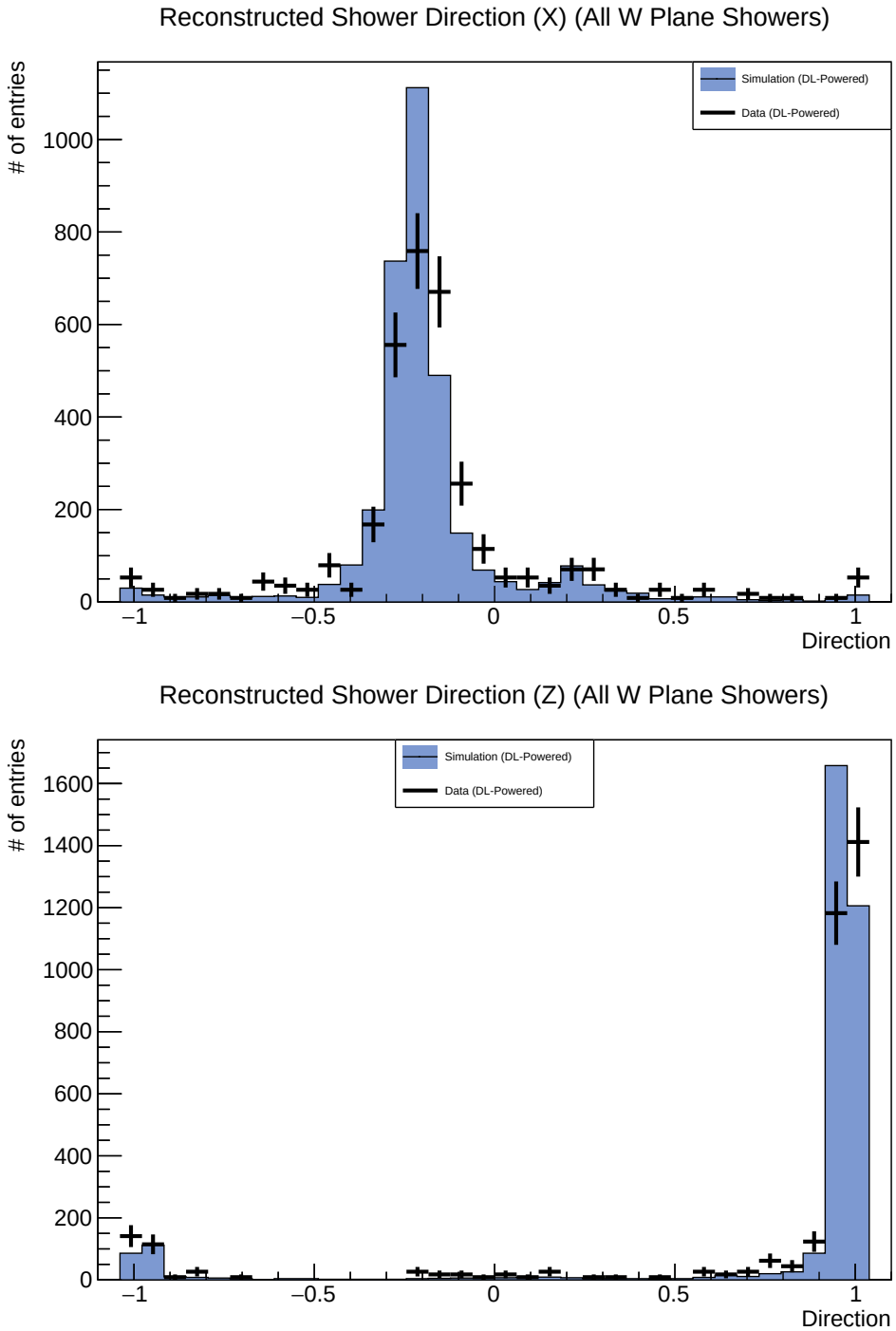


Figure 8.10: Data and simulation comparison of the reconstructed 2D shower direction at ProtoDUNE-SP after the GNN growing. There is also good agreement between both results here. The showers reconstructed going the wrong way in Z are usually due to the correction factor applied to the shower direction being incorrect, pointing the primary axis the wrong way, rather than the shower truly being reconstructed going the wrong way.

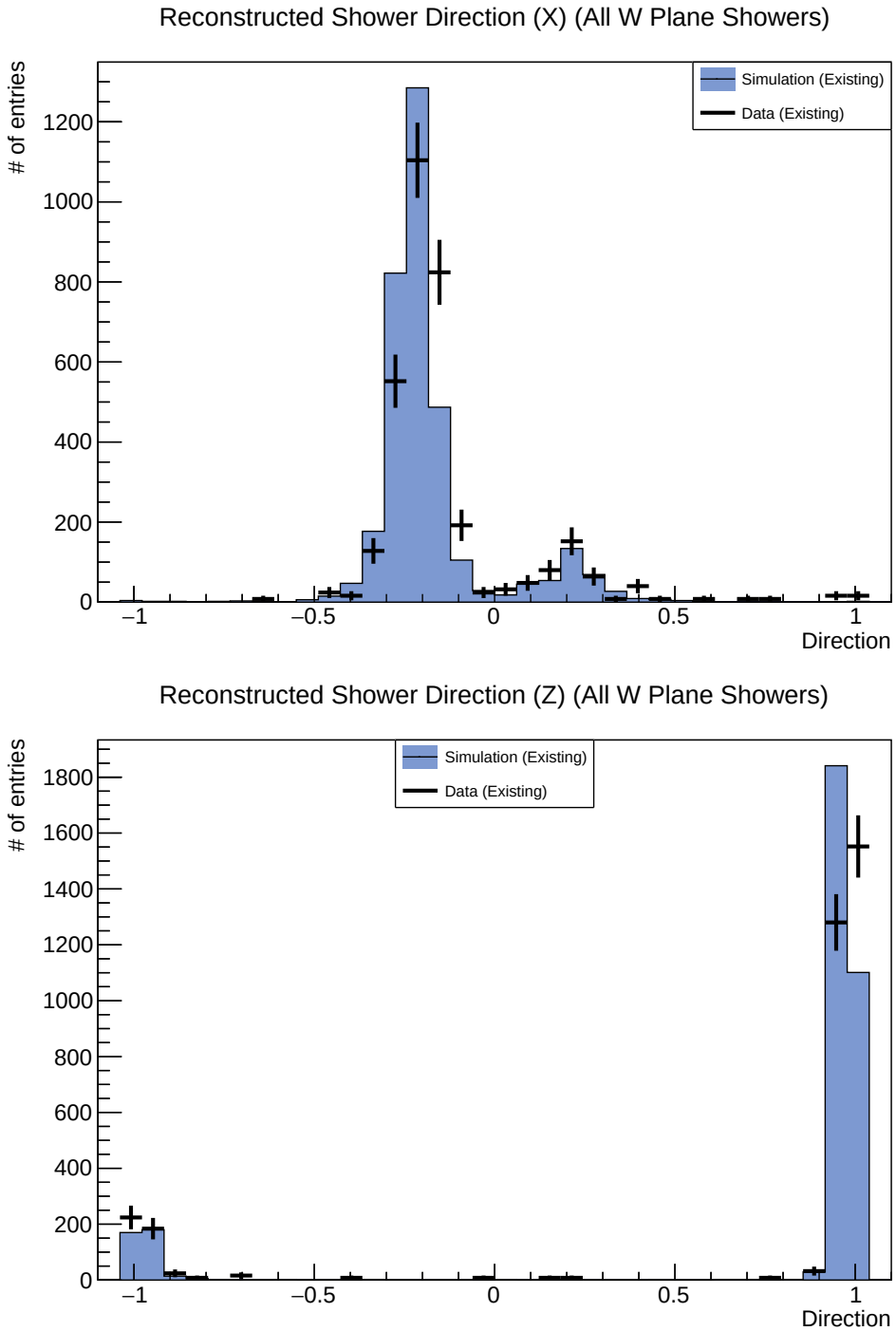


Figure 8.11: Data and simulation comparison of the reconstructed 2D shower direction at ProtoDUNE-SP after the existing growing. There is also good agreement between the two here for the existing growing. It is notable that there is a similar level of disagreement between the bins just below zero, as compared to Figure 8.10.

showers reconstructed with X directions below zero is poor in both plots. This could indicate an issue in the MC simulation, such that the simulated showers and the showers seen in data do not agree. What is perhaps more interesting is looking at how the reconstructed shower length changes when moving from 2D to 3D. In Figures 8.9 and 8.9, there is broadly an agreement between each of the four distributions, with both the old and new method producing a peak around 150. However, once moved into 3D, the DL method produces a peak around 160, compared to 120 in the existing method, with the peak shape also differing much more. In both cases, there is a reasonable agreement between the simulation and data, so it seems that the new shower growing is producing showers of slightly different length, compared to the old method.

Further comparisons of the reconstructed shower properties can be seen in Appendix C, including some plots showing the same comparisons with the new growing, but with cuts applied to show only 'small' and 'large' showers, based on some number of hits.

8.2.5 Summary of ProtoDUNE-SP Performance Study

The DL-based shower growing is able to achieve a very similar level of performance on both data and simulation at ProtoDUNE-SP, which is very encouraging. However, compared to the far detector performance, the difference in reconstruction quality when utilising the DL shower growing is minimal. This is likely due to the outlined differences in the shower-like clusters, where they already have a much higher completeness, rendering an improved shower growing much less useful, at least at the energy tested. Perhaps more importantly, then, is that it also validates that work that is performed on simulated data using deep learning is able to generalise to real data, as very good data and simulation agreement is seen for reconstructed shower properties when using the deep learning-based growing. This is a useful study to perform generally, to better understand how development on simulation can work and what considerations need to be made to then generalise to real data, once available.

As outlined previously, there are many distinct differences between ProtoDUNE and far detector data. Combined, these differences may mean that a different shower growing architecture is more optimal, or additional features should be included for use at ProtoDUNE. It is also important to look at the values quoted in Table 8.4, as the shower growing here is doing much less work than in the far

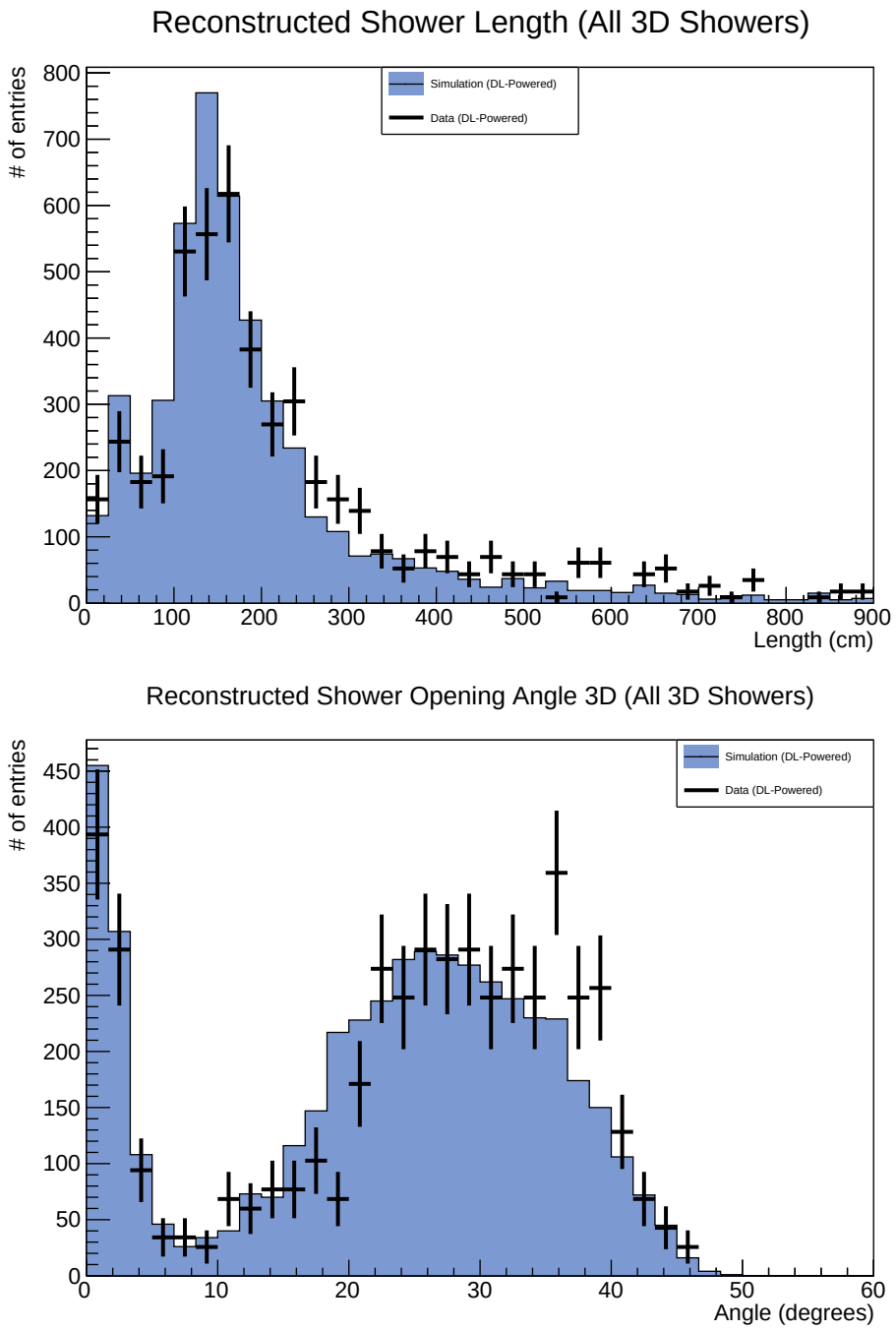


Figure 8.12: Comparison of the reconstructed 3D shower length and opening angle at ProtoDUNE-SP with the deep learning-based growing. This shows similar levels of agreement to Figure 8.8, with some variance in the actual distribution shape. This is likely because the calculation in 3D considers both the second and tertiary axis of the PCA fit to calculate the opening angle, rather than just the secondary. Rather than taking the ratio between the primary and secondary axis, the ratio between the primary and combined squared secondary and tertiary axes.

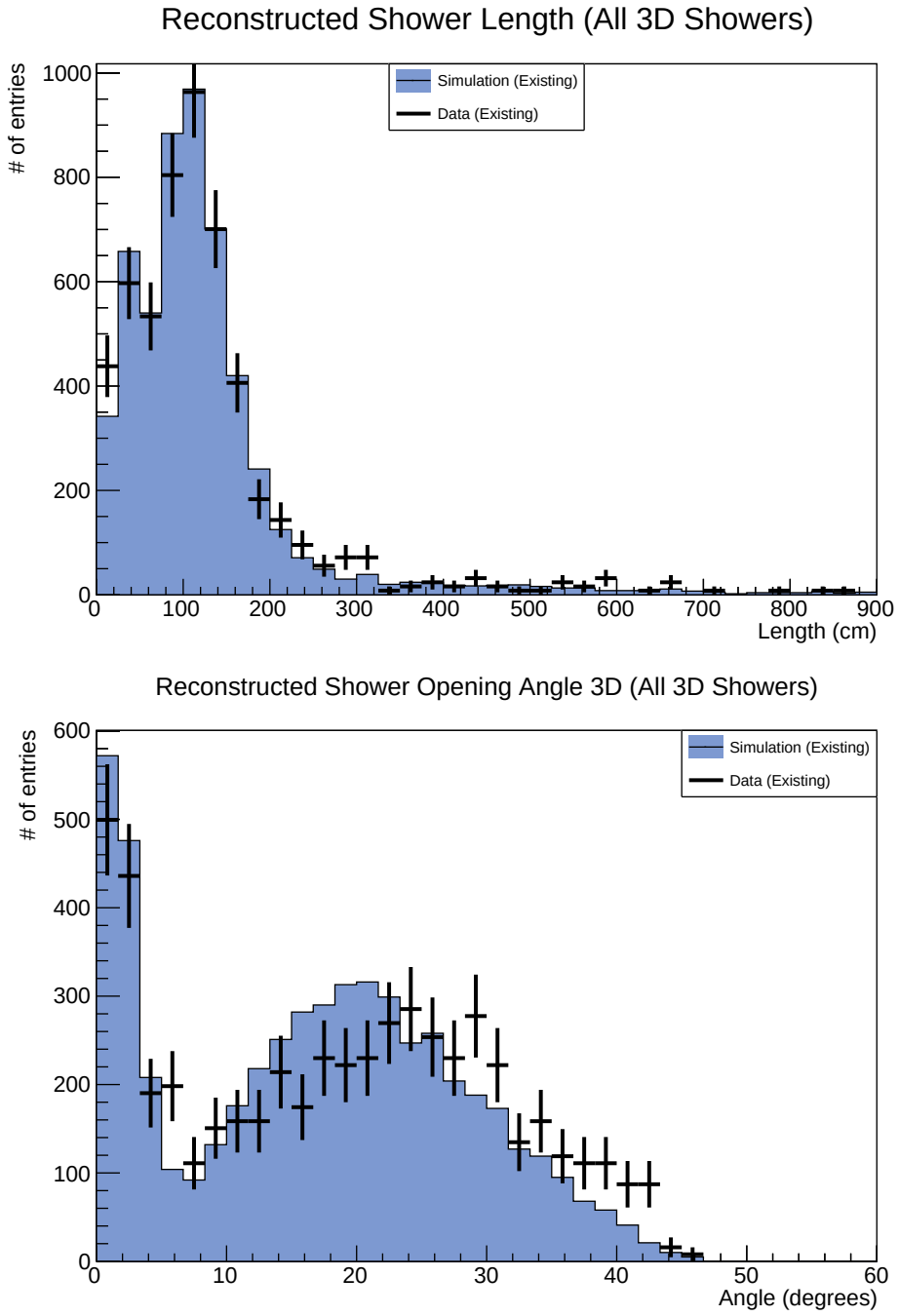


Figure 8.13: Comparison of the reconstructed 3D shower length and opening angle at ProtoDUNE-SP with the existing growing. It is interesting to note that whilst Figures 8.8 and 8.9 mostly agree on the length distributions, there is a disagreement once passed through the further 3D reconstruction algorithms. Similarly, the angular distribution in 3D disagrees even more than in 2D, with the peak moving from just under 30° to 20° when swapping between the existing and new growing techniques.

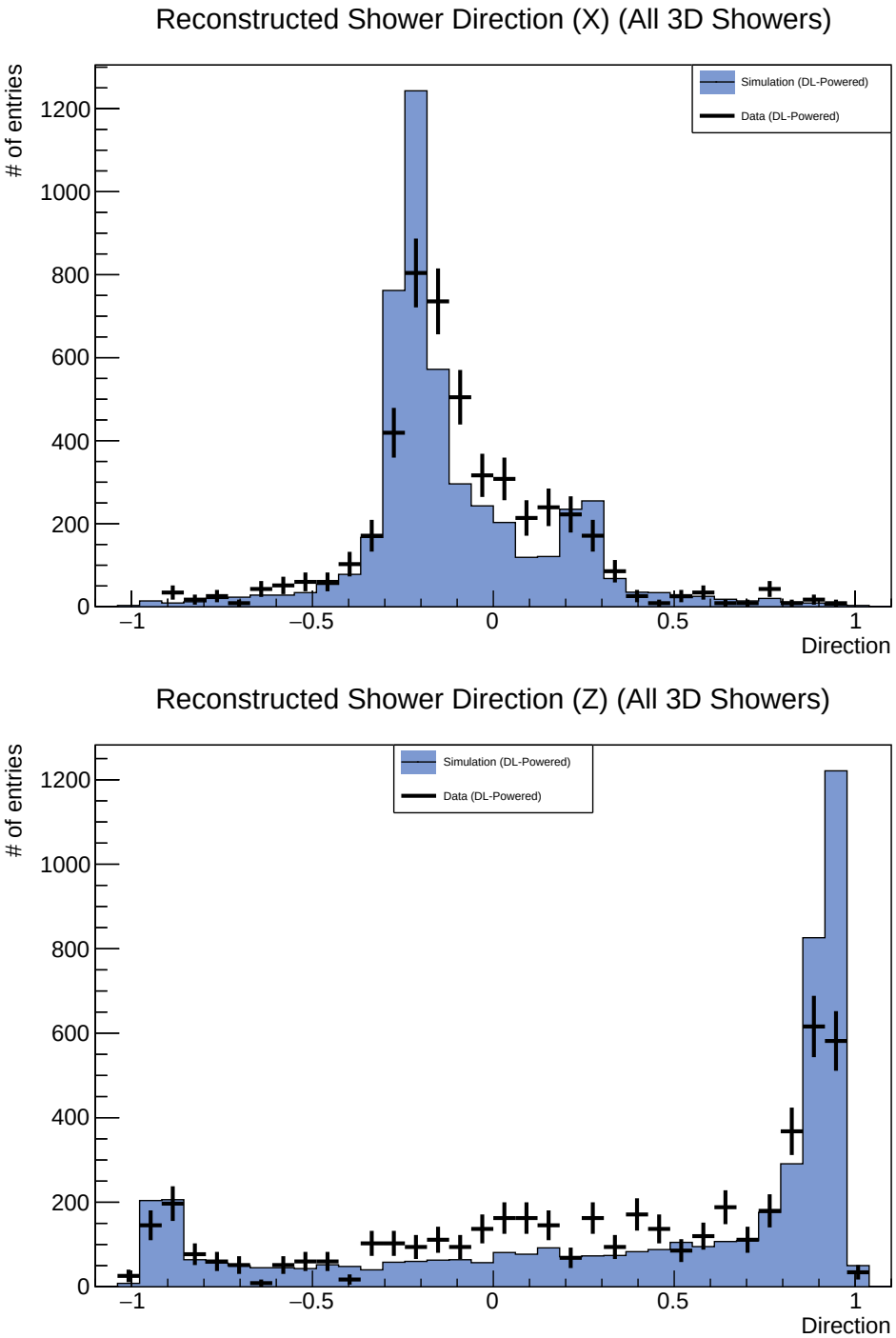


Figure 8.14: A comparison of the reconstructed 3D shower direction at ProtoDUNE, using the new growing. A similar shape in the Z distribution can as seen in Figure 8.10, where some showers have a correction applied to their primary axis that end up pointing it the wrong direction. The bias in the X direction here is due to the angle that the beam enters the detector, which is 30 cm away from the central cathode on the negative X side, pointing down 11° from the horizontal and 10° to the right of the Z direction.

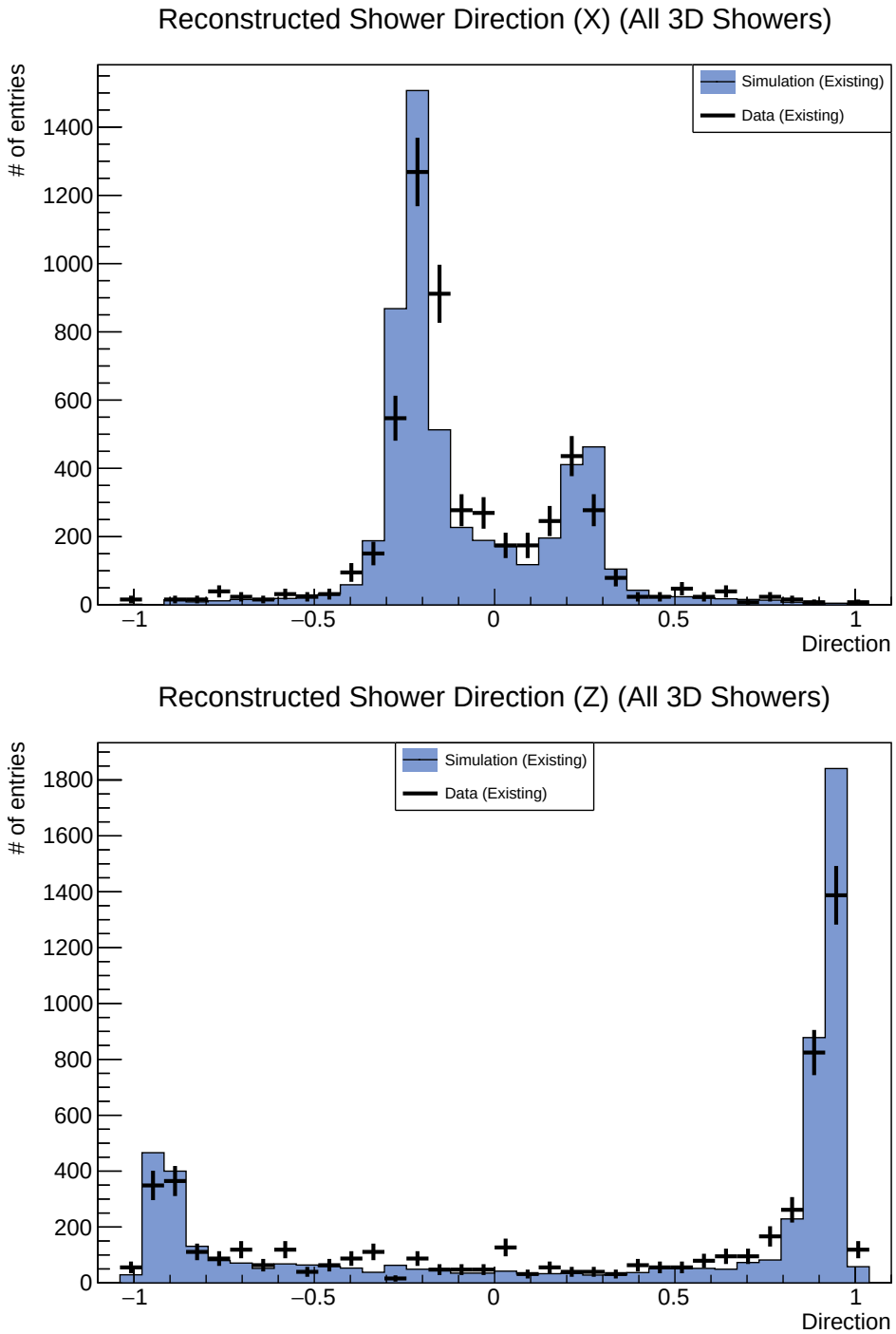


Figure 8.15: A comparison of the reconstructed 3D shower direction at ProtoDUNE, using the existing growing. Like in 2D, the X distribution in both the new and existing growing, shown in Figure 8.14, disagree more for values below zero, with there being both under and over-represented values in the X direction. However, when looking at the Z direction, it seems there is worse agreement in the new growing, when compared to the existing growing.

detector, as the difference before and after growing utilising the existing growing is only 3%, compared to a difference of over 23% at the far detector for the largest shower. The initial completeness before any growing is also very different at ProtoDUNE, with the completeness starting at 62% rather than only 7% at the far detector. In fact, the before growing completeness at ProtoDUNE is almost as high as the after growing completeness achieved with the DL growing at the far detector. This implies that changes to the shower growing are less impactful at ProtoDUNE-SP than the far detector, at least for this 1 GeV positron sample.

The beam positrons measured by ProtoDUNE-SP are different in energy, topology and complexity to the charged-current electron neutrino interactions that will be reconstructed in the DUNE FD. In ProtoDUNE-SP, the beam particles have a single energy, compared to the wide-band LBNF spectrum at the far detector. The final-state multiplicities are also significantly greater at the far detector. This is shown more clearly in Figure 8.16, comparing both the size and number of the clusters at ProtoDUNE-SP compared to the FD, where it is obvious that the shower growing has overall less work to do, such that a change in architecture may be required. Across every shower, at ProtoDUNE the average number of clusters before growing is 107 compared to 584 at the far detector. For ProtoDUNE, this number drops by only 17.6% after growing is complete, compared to a change of 29.2% at the far detector, with even more significant differences when looking at the largest cluster per shower, and the largest overall shower in the event. Similarly, the growing step at ProtoDUNE takes the largest cluster for a given shower from an average of 43 hits to 47 hits, whereas at the far detector the difference is instead from 57 hits to 197 hits. This drastic difference in the size and number of clusters may explain part of the reason the shower growing performs differently at ProtoDUNE-SP.

It is possible that improvements to the shower growing network to target test beam events in a surface-based detector may help improve the performance in the future. For example, as the beam particle enters the detector from a known position, the beam plug, the shower growing process could benefit from topological features that utilise this position. As well as the beam plug, being a test beam event means that there is the presence of additional beam halo particles. This is another form of additional complexity, causing additional interactions in the detector, which the current shower growing is not built to deal with.

Similarly, cosmic ray interactions at ProtoDUNE-SP introduce additional

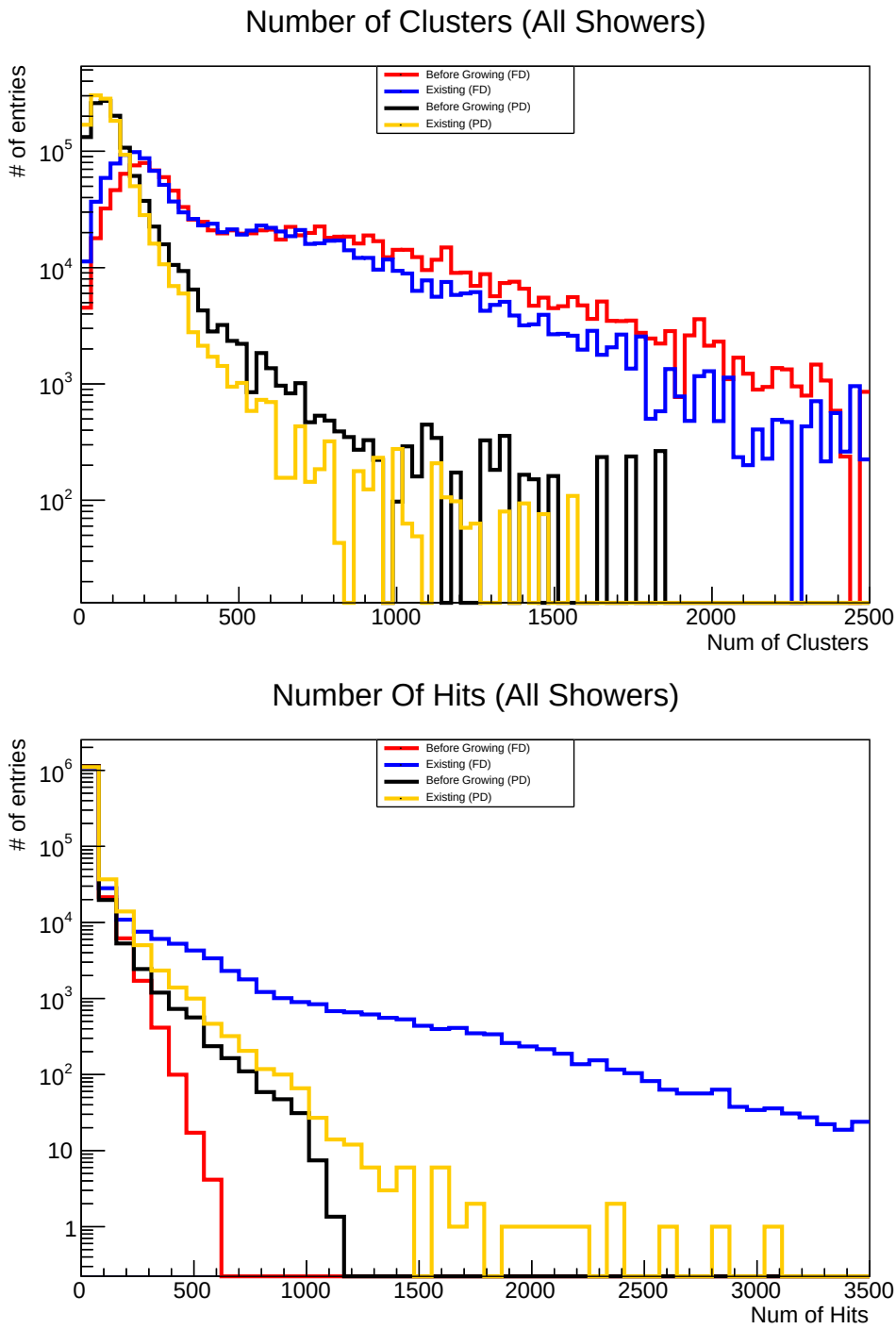


Figure 8.16: A comparison of the existing shower growing at the DUNE FD and ProtoDUNE-SP. First, the number of clusters in a given event is drastically different between the two detectors, both before and after growing, as well as the difference between them. Secondly, the size of the shower-like clusters in the event is much lower at ProtoDUNE, with a minimal change after growing. This may explain why the growing is performing differently on this event sample at ProtoDUNE.

complexity that the network may need improving to tackle. They introduce a large source of noise in the data, resulting in the need for slicing. Because of the cosmic rays, Pandora runs in a different mode, first removing obvious cosmic rays in a number of passes, then working through the remainder of the event using both a cosmic ray and beam reconstruction hypothesis, and choosing the result that makes the most sense. This mode of running, if there are errors, can mean that the final test-beam reconstruction that runs the shower growing for the identified test beam hits, may have either additional hits from cosmic rays, or missing hits lost to other slices. This style of sparse hits is not something the network was trained for, which makes it difficult for the network to perform inference on it, whilst also potentially being non-optimal for the graph structure; Another use of cosmic ray information would be using the identified cosmic ray vertices, such that clusters that are nearer to interaction vertices of the cosmic rays can be treated differently based on their position and direction relative to the other known vertices.

Alongside the potential for additional features to deal with the test beam and surface-based detector of ProtoDUNE-SP, there is usually per-detector tuning. The thresholds or training done for one detector may not be most optimal for another detector, with differences mostly being due to differences in energy spectrum, detector geometry (the size, the wire plane angles), or the presence of additional interactions. All of these small differences mean that it is usually optimal to re-tune, or in the case of deep learning-based approaches, re-train a network using new, detector-specific data. This step was not performed for the DL track/shower tagging network (outlined in Section 4.3.8), such that it can not achieve the same level of performance in ProtoDUNE, as the far detector. This can introduce a background of track-like clusters that make the shower growing more difficult, by either merging tracks into showers, or providing high scoring input clusters that are not suitable. Retraining the network could help alleviate this issue, but was not easily feasible. The track shower tagging network will suffer from the same issues as the shower growing as well, with a lack of understanding of the cosmic rays, the differences in noise and interaction differences.

Regardless, the performance differences aside, the reconstruction-based quantities show that the method is functional at ProtoDUNE-SP, and performs the same on both simulation and real data, even if the configuration of the detector means that an improvement to the growing is less impactful at ProtoDUNE-SP than the FD. This is encouraging, as it helps validate the use of deep learning

more generally in Pandora, for further work on other parts of the reconstruction chain. The performance achieved at the far detector indicates that the method as a whole is able to achieve encouraging performance on electromagnetic showers, with additional work required to ensure that the growing performance is consistent for both electrons and photons, as well as across a range of energies. However, as deep learning becomes more ubiquitous across DUNE and particle physics as a whole, with greater access to GPUs for accelerating deep learning workflows, this network and others like it in Pandora can grow in complexity and scope, to take advantage of the increase in available computing power, which should allow the technique to become even more sophisticated.

9

Conclusion

"Take what you have learned, and move on."

The Fox of Dreams - The Sandman

Deep Underground Neutrino Experiment (DUNE) is a next generation long-baseline neutrino oscillation experiment in the construction phase, with aims to make precision measurements of the neutrino oscillation parameters, in search of CP violation. To achieve these goals, accurate reconstruction of the DUNEs detector technology of choice, the liquid argon time projection chamber (LArTPC), is required. LArTPCs provide high resolution spatial and calorimetric outputs, which results in a complex particle topology to be reconstructed. To efficiently tackle this problem, DUNE will need a variety of reconstruction approaches to accurately reconstruct particle interactions and provide analysers with useful outputs. This thesis presents the results for two new techniques, implemented inside the Pandora multi-algorithm reconstruction framework, with verification of these techniques on both simulated and real data, utilising data from the sophisticated DUNE simulation chain, and test beam data from ProtoDUNE-SP at CERN.

In this thesis, the development of an improved 3D hit reconstruction for tracks was outlined in Chapter 6. This work can efficiently utilise the full power of the 3D hit reconstruction in Pandora, and fix a number of outlined issues in the process. First, simple constraints are enforced on the 3D hit creation, to ensure the detector geometry is considered as part of the hit creation process, to remove errors where a hit falls outside the detector. Next, and most powerfully, the algorithm chain is

split and parallelised, providing a comprehensive set of 3D hits to chose between, rather than a single fixed result. This superset of 3D hits containing many potential final hits is then sampled, first using random sample consensus (RANSAC) to find a consistent starting point, and then with a further fitting step to produce a single coherent result for the given track. This result in final 3D tracks that are more consistent, match their truth more accurately, and lack the reconstruction artefacts previously seen such as deviations in the 3D hits that looked like the track was bending. The 3D track creation was also extended further to include the addition of hit interpolation in a detector aware way.

Next, the shower growing step in Pandora was worked on in Chapter 7. This is a step early in the Pandora reconstruction chain that takes the tiny clusters (grouped together hits) that have been tagged as most likely coming from electromagnetic showers and merges them together to produce larger and larger clusters that more accurately reflect a single shower. Before the work started properly, a cheating study was performed. This study involved the use of truth level information to perform the shower growing, to get a better understanding of how improving the shower growing could impact the overall performance of Pandora. The cheating study showed that in a perfect shower growing case, there could be as much as 15% performance gain for the shower completeness, which is a significant improvement that provides scope for development. Looking at this, and with an understanding of the issues that the existing shower growing faces, a graph neural network (GNN) based approach was developed as part of this thesis. The GNN and its input graph were developed to take an input shower-like cluster and output merge candidates that could be used to grow the initial input cluster. This approach and the development of both the graph and network are also outlined in this chapter, including the decisions made around the graph and network features, and other technical details such as the input cluster selection and the implementation of a GNN inside of Pandora.

Chapter 8 then outlines all the results achieved with the newly developed GNN shower growing. After an explanation of how the network was trained on DUNE electron neutrino events, the performance on simulated data is analysed, both before and after shower growing to show the direct impact of the improved shower growing, as well as the final performance once Pandora has completed the full reconstruction chain. These two result types help show both the direct impact, and the actual usable impact once the full reconstruction chain has used the result

from the deep learning-based growing. Comparing the old growing against this work for the shower growing step only, an improvement of 15.0% is seen in the average completeness across all showers, at the cost of a 5.9% drop in purity. If instead the final reconstruction numbers are used, which includes the impact of other Pandora algorithms continuing to improve the shower growing output, a final improvement of 16.3% is seen, at a cost of 4.0% purity. However, these numbers and the model used were chosen to optimise the highest available completeness, rather than an overall result that optimises completeness, purity and takes shower losses into account. Chapter 8 also outlines some potential changes to the shower growing network developed as part of this work, if an alternative variable was optimised instead, offering a look at the potential tuning or additional features to improve the performance further.

Alongside simulation only results at the DUNE far detector, there is also an overview of results at ProtoDUNE-SP. ProtoDUNE-SP offers a unique opportunity to verify the results seen on DUNE simulation on real data, which is always a concern for deep learning-based approaches, to avoid learning features that are specific to the simulation, rather than real life features. First, a repeat of the simulation-based studies performed at the far detector is applied to ProtoDUNE-SP simulation data. Much smaller improvements are seen here, of the scale 1% or so, and some differences are outlined that potentially explains why the shower growing is much less important at ProtoDUNE-SP, at least for the dataset tested. This is followed by comparisons of the GNN shower growing on real and simulated data, comparing the size and shapes of reconstructed showers, giving a first look at the performance of the new shower growing on real data. Good agreement can be seen between the real and simulated data, giving confidence that deep learning approaches applied to simulated data can successfully be applied to real data without significant issue.

This thesis has presented the results of two major improvements to Pandora, a reconstruction framework in use at both DUNE and ProtoDUNE, including improvements to the 3D reconstruction and utilising deep learning to improve electromagnetic shower reconstruction, both of which are key steps in the reconstruction chain. Work is underway to validate this work at ProtoDUNE and other LArTPC experiments, as well as further improvements to Pandora to ensure DUNE will have an effective reconstruction framework ready when data taking starts.

References

[1] J. Chadwick, "The intensity distribution in the magnetic spectrum of beta particles from radium (B + C)", Verh. Phys. Gesell. **16**, 383–391 (1914).

[2] W. Pauli, "Pauli letter collection: letter to Lise Meitner", Typed copy, `https://cds.cern.ch/record/83282`.

[3] F. Reines and C. L. Cowan, "Detection of the free neutrino", Phys. Rev. **92**, 830–831 (1953).

[4] C. L. Cowan, F. Reines, F. B. Harrison, H. W. Kruse, and A. D. McGuire, "Detection of the free neutrino: A Confirmation", Science **124**, 103–104 (1956).

[5] G. Danby, J. M. Gaillard, K. A. Goulianos, L. M. Lederman, N. B. Mistry, *et al.*, "Observation of High-Energy Neutrino Reactions and the Existence of Two Kinds of Neutrinos", Phys. Rev. Lett. **9**, 36–44 (1962).

[6] M. L. Perl *et al.*, "Evidence for Anomalous Lepton Production in $e^+ - e^-$ Annihilation", Phys. Rev. Lett. **35**, 1489–1492 (1975).

[7] **DONUT** Collaboration, K. Kodama *et al.*, "Observation of tau neutrino interactions", Phys. Lett. B **504**, 218–224 (2001), arXiv:`hep-ex/0012035`.

[8] **ATLAS** Collaboration, G. Aad *et al.*, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC", Physics Letters B **716**, 1–29 (2012).

[9] **L3** Collaboration, M. Acciarri *et al.*, "Determination of the number of light neutrino species from single photon production at LEP", Phys. Lett. B **431**, 199–208 (1998).

[10] B. Pontecorvo, "Mesonium and antimesonium", Soviet Journal of Experimental and Theoretical Physics **6**, 429 (1958).

[11] B. Pontecorvo, "Neutrino experiments and the problem of conservation of leptonic charge", Sov. Phys. JETP **26**, 165 (1968).

[12] R. Davis Jr., D. S. Harmer, and K. C. Hoffman, "Search for neutrinos from the sun", Phys. Rev. Lett. **20**, 1205–1209 (1968).

[13] J. N. Bahcall, N. A. Bahcall, and G. Shaviv, "Present status of the theoretical predictions for the Cl-36 solar neutrino experiment", Phys. Rev. Lett. **20**, 1209–1212 (1968).

[14] **SNO** Collaboration, Q. R. Ahmad *et al.*, "Direct evidence for neutrino flavor transformation from neutral current interactions in the Sudbury Neutrino Observatory", Phys. Rev. Lett. **89**, 011301 (2002), arXiv:nucl-ex/0204008.

[15] **Super-Kamiokande** Collaboration, Y. Fukuda *et al.*, "Evidence for oscillation of atmospheric neutrinos", Phys. Rev. Lett. **81**, 1562–1567 (1998), arXiv:hep-ex/9807003.

[16] **Super-Kamiokande** Collaboration, Z. Li *et al.*, "Measurement of the tau neutrino cross section in atmospheric neutrino oscillations with Super-Kamiokande", Physical Review D **98**, 10.1103/physrevd.98.052006 (2018).

[17] R. Beuselinck, "High energy neutrino and antineutrino interactions in a neon filled bubble chamber", PhD thesis (Imperial Coll., London, 1979).

[18] R. Bock, R. Brun, O. Couet, J. C. Marin, R. Nierhaus, *et al.*, "PAW: TOWARDS A PHYSICS ANALYSIS WORKSTATION", Comput. Phys. Commun. **45**, edited by W. Ash, 181–190 (1987).

[19] Y. Takeuchi, "Measurement of solar neutrinos from 1000 days of data at Kamiokande-III", PhD thesis (Tokyo Inst. Tech., 1995).

[20] M. Bergevin, "Search for Neutron Anti-neutron Oscillation at the Sudbury Neutrino Observatory", PhD thesis (Guelph U., 2010).

[21] S. Kasuga, "Observation of a Small Muon-neutrino / Electron-neutrino Ratio of Atmospheric Neutrinos in Super-Kamiokande by the Method of Particle Identification", PhD thesis (Tokyo U., 1998).

[22] T. J. Irvine, "Development of Neutron-Tagging Techniques and Application to Atmospheric Neutrino Oscillation Analysis in Super-Kamiokande", PhD thesis (U. Tokyo (main), 2014).

[23] H.-J. Yang, B. P. Roe, and J. Zhu, "Studies of boosted decision trees for MiniBooNE particle identification", Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **555**, 370–385 (2005).

[24] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. D. Messier, *et al.*, "A Convolutional Neural Network Neutrino Event Classifier", JINST **11**, P09001 (2016), arXiv:1604.01444 [hep-ex].

[25] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman, and A. Schwartzman, "Jet-images —deep learning edition", Journal of High Energy Physics **2016**, 10.1007/jhep07(2016)069 (2016).

[26] E. Racah, S. Ko, P. Sadowski, W. Bhimji, C. Tull, *et al.*, "Revealing Fundamental Physics from the Daya Bay Neutrino Experiment Using Deep Neural Networks", in 2016 15th IEEE international conference on machine learning and applications (ICMLA) (Dec. 2016).

[27] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, *et al.*, "The open science grid", in J. phys. conf. ser. Vol. 78, 78 (2007), p. 012057.

[28] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, *et al.*, "The pilot way to grid resources using glideinWMS", in 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 2, 2 (2009), pp. 428–432.

[29] O. S. Grid, *Open Science Grid*, https://gracc.opensciencegrid.org/.

[30] J. A. Formaggio and G. P. Zeller, "From eV to EeV: Neutrino cross sections across energy scales", Reviews of Modern Physics **84**, 1307–1341 (2012).

[31] **DUNE** Collaboration, B. Abi *et al.*, "Deep Underground Neutrino Experiment (DUNE), Far Detector Technical Design Report, Volume II: DUNE Physics", (2020), arXiv:2002.03005 [hep-ex].

[32] **DUNE** Collaboration, B. Abi *et al.*, "Deep Underground Neutrino Experiment (DUNE), Far Detector Technical Design Report, Volume I Introduction to DUNE", JINST **15**, T08008 (2020), arXiv:2002.02967 [physics.ins-det].

[33] I. Esteban, M. Gonzalez-Garcia, M. Maltoni, T. Schwetz, and A. Zhou, "The fate of hints: updated global analysis of three-flavor neutrino oscillations", Journal of High Energy Physics **2020**, 10.1007/jhep09(2020)178 (2020).

[34] Particle Physics Project Prioritization Panel, *Building for Discovery; Strategic Plan for U.S. Particle Physics in the Global Context*, (2014) http://science.energy.gov/~/media/hep/hepap/pdf/May%5C%202014/FINAL%5C%5FP5%5C%5FReport%5C%5FInteractive%5C%5F060214.pdf.

[35] A. D. Sakharov, "Violation of CP-invariance, C-asymmetry, and baryon asymmetry of the Universe", in *In The Intermissions... Collected Works on Research into the Essentials of Theoretical Physics in Russian Federal Nuclear Center, Arzamas-16* (World Scientific, 1998), pp. 84–87.

[36] P. Derwent *et al.*, *Proton Improvement Plan-II*, `https://projectx-docdb.fnal.gov/cgi-bin/ShowDocument?docid=1232`.

[37] D. R. Nygren, "The Time Projection Chamber: A New 4 pi Detector for Charged Particles", eConf **C740805**, edited by J. Kadyk, D. Nygren, W. Wenzel, and F. Winkelmann, 58 (1974).

[38] J. N. Marx and D. R. Nygren, "The Time Projection Chamber", Physics Today **31**, 46–53 (1978), eprint: `https://doi.org/10.1063/1.2994775`.

[39] C. Rubbia, "The Liquid Argon Time Projection Chamber: A New Concept for Neutrino Detectors", (1977).

[40] **MicroBooNE** Collaboration, R. Acciarri *et al.*, "Design and Construction of the MicroBooNE Detector", JINST **12**, P02017 (2017), arXiv:`1612.05824 [physics.ins-det]`.

[41] **DUNE** Collaboration, B. Abi *et al.*, "Deep Underground Neutrino Experiment (DUNE), Far Detector Technical Design Report, Volume IV: Far Detector Single-phase Technology", JINST **15**, T08010 (2020), arXiv:`2002.03010 [physics.ins-det]`.

[42] P. A. Machado, O. Palamara, and D. W. Schmitz, "The Short-Baseline Neutrino Program at Fermilab", Ann. Rev. Nucl. Part. Sci. **69**, 363–387 (2019), arXiv:`1903.04608 [hep-ex]`.

[43] A. A. Machado and E. Segreto, "ARAPUCA a new device for liquid argon scintillation light detection", JINST **11**, C02004 (2016).

[44] H. da Motta *et al.*, "ARAPUCA light trap for large liquid argon time projection chambers", J. Phys. Conf. Ser. **1143**, edited by R. Ochoa, E. Salinas, and H. Blas, 012003 (2018).

[45] E. Segreto, A. A. Machado, A. Fauth, R. R. Ramos, G. de Souza, *et al.*, "First liquid argon test of the X-ARAPUCA", JINST **15**, C05045 (2020).

[46] **ArgonCube** Collaboration, C. Amsler *et al.*, *ArgonCube: a novel, fully-modular approach for the realization of large-mass liquid argon TPC neutrino detectors*, tech. rep. (CERN, Geneva, Feb. 2015).

[47] **MINERvA** Collaboration, L. Aliaga *et al.*, "Design, calibration, and performance of the MINERvA detector", Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **743**, 130–159 (2014).

[48] **DUNE** Collaboration, A. Abed Abud *et al.*, "Deep Underground Neutrino Experiment (DUNE) Near Detector Conceptual Design Report", Instruments **5**, 31 (2021), arXiv:2103.13910 [physics.ins-det].

[49] S. Charley, *DUNE publishes first physics results from prototype detector*, https://news.fnal.gov/2020/12/dune-publishes-first-physics-results-from-prototype-detector/.

[50] **DUNE** Collaboration, B. Abi *et al.*, "First results on ProtoDUNE-SP liquid argon time projection chamber performance from a beam test at the CERN Neutrino Platform", JINST **15**, P12004. 104 p (2020), arXiv:2007.06722.

[51] **DUNE** Collaboration, F. Cavanna *et al.*, *NP-04/ProtoDUNE-SP Report 2020*, tech. rep. (CERN, Geneva, Mar. 2020).

[52] C. Green, J. Kowalkowski, M. Paterno, M. Fischler, L. Garren, *et al.*, "The Art Framework", J. Phys. Conf. Ser. **396**, edited by M. Ernst, D. Düllmann, O. Rind, and T. Wong, 022020 (2012).

[53] *LArSoft Website*, https://larsoft.org/.

[54] E. D. Church, "LArSoft: A Software Package for Liquid Argon Time Projection Drift Chambers", (2013), arXiv:1311.6774 [physics.ins-det].

[55] **GEANT4** Collaboration, S. Agostinelli *et al.*, "GEANT4–a simulation toolkit", Nucl. Instrum. Meth. A **506**, 250–303 (2003).

[56] *Wire-Cell Toolkit*, https://github.com/WireCell.

[57] B. Viren, *Full TPC Signal Simulation*, tech. rep., https://indico.fnal.gov/event/10641/session/16/material/slides/0?contribId=31 ().

[58] **LBNF/DUNE** Collaboration, P. Adamson *et al.*, *Long-Baseline Neutrino Facility (LBNF)/DUNE Conceptual Design Report: Annex 3A_opt*, DUNE doc 4559, http://docs.dunescience.org/cgi-bin/ShowDocument?docid=4559&asof=2019-7-15 (2017).

[59] C. Andreopoulos *et al.*, "The GENIE Neutrino Monte Carlo Generator", Nucl. Instrum. Meth. A **614**, 87–104 (2010), arXiv:0905.2517 [hep-ph].

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

[61] *SNOwGLoBES*, `http://www.phy.duke.edu/~schol/snowglobes`.

[62] J. Gava, J. Kneller, C. Volpe, and G. C. McLaughlin, "A Dynamical collective calculation of supernova neutrino signals", Phys. Rev. Lett. **103**, 071101 (2009), arXiv:`0902.0317 [hep-ph]`.

[63] T. Golan, J. T. Sobczyk, and J. Zmuda, "NuWro: the Wrocław Monte Carlo Generator of Neutrino Interactions", Nucl. Phys. B Proc. Suppl. **229-232**, edited by G. S. Tzanakos, 499–499 (2012).

[64] B. Baller, "Liquid argon TPC signal formation, signal processing and reconstruction techniques", JINST **12**, P07010 (2017), arXiv:`1703.04024 [physics.ins-det]`.

[65] **MicroBooNE** Collaboration, C. Adams *et al.*, "Ionization electron signal processing in single phase LArTPCs. Part I. Algorithm Description and quantitative evaluation with MicroBooNE simulation", Journal of Instrumentation **13**, P07006–P07006 (2018).

[66] **MicroBooNE** Collaboration, C. Adams *et al.*, "Ionization electron signal processing in single phase LArTPCs. Part II. Data/simulation comparison and performance in MicroBooNE", Journal of Instrumentation **13**, P07007–P07007 (2018).

[67] **MicroBooNE** Collaboration, P. Abratenko *et al.*, "Wire-Cell 3D Pattern Recognition Techniques for Neutrino Event Reconstruction in Large LArTPCs: Algorithm Description and Quantitative Evaluation with MicroBooNE Simulation", (2021), arXiv:`2110.13961 [physics.ins-det]`.

[68] **DeepLearnPhysics** Collaboration, L. Dominé *et al.*, "Scalable deep convolutional neural networks for sparse, locally dense liquid argon time projection chamber data", Phys. Rev. D **102**, 012005 (2020), arXiv:`1903.05663 [hep-ex]`.

[69] **DUNE** Collaboration, B. Abi *et al.*, "Neutrino interaction classification with a convolutional neural network in the DUNE far detector", Phys. Rev. D **102**, 092003 (2020), arXiv:`2006.15052 [physics.ins-det]`.

[70] J. S. Marshall and M. A. Thomson, "The Pandora Software Development Kit for Pattern Recognition", Eur. Phys. J. C **75**, 439 (2015), arXiv:1506.05348 [physics.data-an].

[71] **MicroBooNE** Collaboration, R. Acciarri *et al.*, "The Pandora multi-algorithm approach to automated pattern recognition of cosmic-ray muon and neutrino events in the MicroBooNE detector", Eur. Phys. J. C **78**, 82 (2018), arXiv:1708.03135 [hep-ex].

[72] **DUNE** Collaboration, A. A. Abud *et al.*, *Reconstruction of interactions in the ProtoDUNE-SP detector with Pandora*, tech. rep. arXiv:2206.14521 (arXiv, June 2022).

[73] M. A. Thomson, "Particle Flow Calorimetry and the PandoraPFA Algorithm", Nucl. Instrum. Meth. A **611**, 25–40 (2009), arXiv:0907.3577 [physics.ins-det].

[74] J. S. Marshall, A. Münnich, and M. A. Thomson, "Performance of Particle Flow Calorimetry at CLIC", Nucl. Instrum. Meth. A **700**, 153–162 (2013), arXiv:1209.4039 [physics.ins-det].

[75] A. Reynolds, "Evaluating the low-energy response of the ProtoDUNE-SP detector using Michel electrons", PhD thesis (University of Oxford, 2020).

[76] R. E. Schapire *et al.*, "A brief introduction to boosting", in Ijcai, Vol. 99, 999 (Citeseer, 1999), pp. 1401–1406.

[77] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035.

[78] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like Environment for Machine Learning", in BigLearn, NIPS workshop, CONF (2011).

[79] **DUNE** Collaboration, A. Chappell, "A deep neural network to direct the Pandora multi-algorithm LArTPC event reconstruction", Poster presented at Neutrino 2020, https://indico.fnal.gov/event/19348/contributions/186187/.

[80] A. Lenail, *NN-SVG: Publication-ready NN-architecture schematics*, `https://alexlenail.me/NN-SVG`.

[81] G. Cybenko, "Approximation by superpositions of a sigmoidal function", en, Mathematics of Control, Signals, and Systems **2**, 303–314 (1989).

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", en, Nature **323**, 533–536 (1986).

[83] T. Zia and U. Zahid, "Long short-term memory recurrent neural network architectures for Urdu acoustic modeling", en, International Journal of Speech Technology **22**, 21–30 (2019).

[84] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, *et al.*, "Generative Adversarial Networks", arXiv:1406.2661 [cs, stat], arXiv: 1406.2661 (2014).

[85] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE **86**, 2278–2324 (1998).

[86] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition", IEEE Transactions on Systems, Man, and Cybernetics **SMC-13**, 826–834 (1983).

[87] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex", en, The Journal of Physiology **195**, 215–243 (1968).

[88] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision", in Proceedings of 2010 ieee international symposium on circuits and systems (2010), pp. 253–256.

[89] W. W. Zachary, "An Information Flow Model for Conflict and Fission in Small Groups", en, Journal of Anthropological Research **33**, 452–473 (1977).

[90] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX", in Proceedings of the 7th python in science conference, edited by G. Varoquaux, T. Vaught, and J. Millman (2008), pp. 11–15.

[91] W. L. Hamilton, *Graph Representation Learning*, eng, Synthesis Lectures on Artificial Intelligence and Machine Learning #46 (Morgan & Claypool Publishers, San Rafael, California, 2020).

[92] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry", arXiv:1704.01212 [cs], arXiv: 1704.01212 (2017).

[93] Z. Jin, Y. Wang, Q. Wang, Y. Ming, T. Ma, *et al.*, "GNNLens: A Visual Analytics Approach for Prediction Error Diagnosis of Graph Neural Networks", arXiv:2011.11048 [cs], arXiv: 2011.11048 (2021).

[94] C. Merkwirth and T. Lengauer, "Automatic Generation of Complementary Descriptors with Molecular Graph Networks", Journal of Chemical Information and Modeling **45**, PMID: 16180893, 1159–1168 (2005), eprint: `https://doi.org/10.1021/ci049613b`.

[95] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model", Trans. Neur. Netw. **20**, 61–80 (2009).

[96] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks", arXiv:1609.02907 [cs, stat], arXiv: 1609.02907 (2017).

[97] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs", arXiv:1706.02216 [cs, stat], arXiv: 1706.02216 (2018).

[98] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?", arXiv:1810.00826 [cs, stat], arXiv: 1810.00826 (2019).

[99] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A Review of Relational Machine Learning for Knowledge Graphs", Proceedings of the IEEE **104**, 11–33 (2016).

[100] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph Matching Networks for Learning the Similarity of Graph Structured Objects", arXiv:1904.12787 [cs, stat], arXiv: 1904.12787 (2019).

[101] B. Graham, M. Engelcke, and L. van der Maaten, "3D Semantic Segmentation with Submanifold Sparse Convolutional Networks", CVPR (2018).

[102] M. Paganini, L. de Oliveira, and B. Nachman, "CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks", Phys. Rev. D **97**, 014021 (2018), arXiv:`1712.10321 [hep-ex]`.

[103] S. Alonso-Monsalve and L. H. Whitehead, "Image-based model parameter optimization using Model-Assisted Generative Adversarial Networks", IEEE Trans. Neural Networks **31**, 5645–5650 (2020), arXiv:1812.00879 [cs.CV].

[104] S. Alonso-Monsalve, "Novel usage of deep learning and high-performance computing in long-baseline neutrino oscillation experiments", Presented 09 Feb 2021 (2020).

[105] H. Yu *et al.*, "Augmented signal processing in Liquid Argon Time Projection Chambers with a deep neural network", JINST **16**, P01036 (2021), arXiv:2007.12743 [physics.ins-det].

[106] **MicroBooNE** Collaboration, K. Mason, "Using a Convolutional Neural Network to Reconstruct Dead Channels in MicroBooNE", (2019).

[107] X. Ju *et al.*, "Performance of a geometric deep learning pipeline for HL-LHC particle tracking", Eur. Phys. J. C **81**, 876 (2021), arXiv:2103.06995 [physics.data-an].

[108] **D0** Collaboration, V. M. Abazov *et al.*, "Observation of Single Top Quark Production", Phys. Rev. Lett. **103**, 092001 (2009), arXiv:0903 . 0850 [hep-ex].

[109] **CDF** Collaboration, T. Aaltonen *et al.*, "First Observation of Electroweak Single Top Quark Production", Phys. Rev. Lett. **103**, 092002 (2009), arXiv:0903.0885 [hep-ex].

[110] HEP ML Community, *A Living Review of Machine Learning for Particle Physics*, https://iml-wg.github.io/HEPML-LivingReview/.

[111] W. R. Inc., *Mathematica*, Champaign, IL, 2021.

[112] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", Commun. ACM **24**, 381–395 (1981).

[113] **DUNE** Collaboration, I. Mawby, *Development of the Pandora LArTPC event reconstruction to optimise the sensitivity to CP violation at DUNE*, June 2022.

[114] M. Fey and J. E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric*, May 2019.

[115] G. Li, C. Xiong, A. Thabet, and B. Ghanem, *DeeperGCN: All You Need to Train Deeper GCNs*, 2020.

[116] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in Proceedings of the ieee conference on computer vision and pattern recognition (2016), pp. 770–778.

[117] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks", in European conference on computer vision (Springer, 2016), pp. 630–645.

[118] M. Wang, T. Yang, M. A. Flechas, P. Harris, B. Hawks, *et al.*, "GPU-Accelerated Machine Learning Inference as a Service for Computing in Neutrino Experiments", Frontiers in Big Data **3**, 10.3389/fdata.2020.604083 (2021).

[119] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2014.

[120] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, *et al.*, "Tune: A Research Platform for Distributed Model Selection and Training", arXiv preprint arXiv:1807.05118 (2018).

[121] K. Zuber, *Neutrino physics*, eng, 2nd ed., Series in high energy physics, cosmology and gravitation ; v. 2 (Taylor & Francis, New York ; London, 2010).

[122] **Particle Data Group** Collaboration, P. Zyla *et al.*, "Review of Particle Physics", PTEP **2020**, 083C01 (2020).

[123] S. F. King, "Neutrino Mass and Mixing in the Seesaw Playground", Nucl. Phys. B **908**, 456–466 (2016), arXiv:1511.03831 [hep-ph].

[124] H. Nunokawa, S. Parke, and J. W. Valle, "CP violation and neutrino oscillations", Progress in Particle and Nuclear Physics **60**, 338–402 (2008).

[125] **SNO** Collaboration, Q. R. Ahmad *et al.*, "Measurement of the rate of $\nu_e + d \to p + p + e^-$ interactions produced by ^{8}B solar neutrinos at the Sudbury Neutrino Observatory", Phys. Rev. Lett. **87**, 071301 (2001), arXiv:nucl-ex/0106015.

[126] **KamLAND** Collaboration, T. Araki *et al.*, "Measurement of neutrino oscillation with KamLAND: Evidence of spectral distortion", Phys. Rev. Lett. **94**, 081801 (2005), arXiv:hep-ex/0406035.

[127] **Daya Bay** Collaboration, F. P. An *et al.*, "Observation of Electron-Antineutrino Disappearance at Daya Bay", Physical Review Letters **108**, 10.1103/physrevlett.108.171803 (2012).

[128] **Double Chooz** Collaboration, Y. Abe *et al.*, "Indication of Reactor $\bar{\nu}_e$ Disappearance in the Double Chooz Experiment", Physical Review Letters **108**, 10.1103/physrevlett.108.131801 (2012).

[129] **RENO** Collaboration, J. K. Ahn *et al.*, "Observation of Reactor Electron Antineutrinos Disappearance in the RENO Experiment", Physical Review Letters **108**, 10.1103/physrevlett.108.191802 (2012).

[130] **T2K** Collaboration, K. Abe *et al.*, "Measurement of neutrino and antineutrino oscillations by the T2K experiment including a new additional sample of ν_e interactions at the far detector", Phys. Rev. D **96**, 092006 (2017).

[131] **NOvA** Collaboration, M. A. Acero *et al.*, "First measurement of neutrino oscillation parameters using neutrinos and antineutrinos by NOvA", Phys. Rev. Lett. **123**, 151803 (2019).

[132] **MINOS** Collaboration, P. Adamson *et al.*, "Combined Analysis of ν_μ Disappearance and $\nu_\mu \to \nu_e$ Appearance in MINOS Using Accelerator and Atmospheric Neutrinos", Phys. Rev. Lett. **112**, 191801 (2014).

[133] **NOvA** Collaboration, M. A. Acero *et al.*, "An Improved Measurement of Neutrino Oscillation Parameters by the NOvA Experiment", (2021), arXiv:2108.08219 [hep-ex].

[134] **T2K** Collaboration, K. Abe *et al.*, "Constraint on the matter–antimatter symmetry-violating phase in neutrino oscillations", Nature **580**, 339–344 (2020).

[135] **Hyper-Kamiokande** Collaboration, K. Abe *et al.*, "Hyper-Kamiokande Design Report", (2018), arXiv:1805.04163 [physics.ins-det].

[136] K. J. Kelly, P. A. N. Machado, S. J. Parke, Y. F. Perez-Gonzalez, and R. Z. Funchal, "Neutrino mass ordering in light of recent data", Physical Review D **103**, 10.1103/physrevd.103.013004 (2021).

Appendix

A

Neutrino Oscillations

Neutrino oscillations arise due to the quantum mechanical mixing of the 3 different neutrino mass eigenstates. We can express the 3 flavour states, electron, muon and tau, ν_e, ν_μ, ν_τ respectively, as a superposition of the mass eigenstates

$$|\nu_\alpha\rangle = \sum_k U^*_{\alpha k} |\nu_k\rangle \tag{A.0.1}$$

where ν_k are the neutrino mass eigenstates, and U is a unitary mixing matrix. The mixing matrix is analogous to the Cabibbo-Kobayashi-Maskawa (CKM) mixing matrix of the quark sector, but for neutrinos it is the Pontecorvo-Maki-Nakagawa-Sakata (PMNS) matrix. This is a 3×3 unitary matrix, but it can be parameterised using three weak mixing angles, $\theta_{12}, \theta_{13}, \theta_{23}$ and one phase δ_{CP}. The U_{PMNS} matrix can be written as [121]:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta_{\mathrm{CP}}} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta_{CP}} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & e^{i\beta} & 0 \\ 0 & 0 & e^{i\gamma} \end{pmatrix} \tag{A.0.2}$$

where $c_{ij} = \cos\theta_{ij}$ and $s_{ij} = \sin\theta_{ij}$. In this parametrisation, each sub-matrix contains a different mixing angle and is primarily associated with a different mode of oscillation. As discussed in Section B, the θ_{23} mixing angle drives the long-baseline oscillations that are observed in atmospheric and accelerator neutrinos; θ_{13} describes the short-baseline oscillations that occur in reactor neutrinos; and θ_{12} manifests itself in solar neutrinos and long-baseline reactor neutrino experiments. If the phase δ_{CP} is not a multiple of π, then U_{PMNS} will have complex components

in non-diagonal elements, leading to CP violation. The forth sub-matrix includes the two so-called Majorana phases, $e^{i\beta}$ and $e^{i\gamma}$. This matrix plays no part in the description of neutrino flavour oscillations, so will not be considered further here, but does play a key role in neutrinoless double beta decay.

A.1 Oscillation Formalism

This formalism draws mostly from *Neutrino Physics Second Edition* by K. Zuber [121].

First, assuming a plane wave solution to the time independent Schrödinger equation, the mass eigenstates of the neutrino $|\nu_i\rangle$ have a time dependence

$$|\nu_i(\vec{x},t)\rangle = e^{-i\phi_i}\,|\nu_i(\vec{0},0)\rangle \tag{A.1.1}$$

where $\phi_i = E_i t - \vec{p}\cdot\vec{x}$. Now, since we know that at some given time and place, the neutrinos flavour state $|\nu_\beta(\vec{x},t)\rangle$ will be measured, we can write it as

$$|\nu_\beta(\vec{x},t)\rangle = \sum_i U_{\beta i}\,|\nu_i(\vec{x},t)\rangle \tag{A.1.2}$$

which arises from Equation A.0.1. When this is combined with Equation A.1.1 gives

$$|\nu_\beta(\vec{x},t)\rangle = \sum_i U_{\beta i} e^{-i\phi_i}\,|\nu_i(\vec{0},0)\rangle \tag{A.1.3}$$

If we assume that the initial neutrino state is purely $|\nu_\alpha\rangle$, then the transition amplitude $\nu_\alpha \to \nu_\beta$ is given by

$$A(\alpha\to\beta)(t) = \langle\nu_\beta(\vec{x},t)|\nu_\alpha(\vec{x},t)\rangle = \sum_i\sum_j U^*_{\beta i} e^{i\phi_i} U_{\alpha j}\,\langle\nu_i(\vec{0},0)|\nu_j(\vec{0},0)\rangle \tag{A.1.4}$$

Now, because the mass eigenstates are orthogonal, we can reduce Equation A.1.4 to:

$$A(\alpha\to\beta)(t) = \sum_i U^*_{\beta i} e^{i\phi_i} U_{\alpha i} \tag{A.1.5}$$

If we make the further assumption that the neutrino travels in the x-direction, ϕ_i becomes

$$\phi_i = E_i t - \vec{p} \cdot \vec{x} = E_i t - p_i x \tag{A.1.6}$$

and under the assumption that each mass state has the same momenta, and the neutrinos are ultra-relativistic (i.e. $m << p$):

$$p \approx E - \frac{m^2}{2E} \tag{A.1.7}$$

which when put back into Equation A.1.6 and used with the relativistic assumption that $L = x = t$, can be simplified to

$$\phi \approx \frac{m^2 L}{2E} \tag{A.1.8}$$

This can be combined with Equation A.1.5 to obtain

$$A(\alpha \to \beta)(t) = \sum_i U^*_{\beta i} e^{i\frac{m_i^2 L}{2E}} U_{\alpha i} \tag{A.1.9}$$

The transition probability $P(\alpha \to \beta)(t)$ can then be obtained from the transition amplitude A

$$P(\alpha \to \beta)(t) = |A(\alpha \to \beta)(t)|^2 = \sum_i \sum_j U^*_{\beta i} U_{\beta j} e^{i\frac{(m_i^2 - m_j^2)L}{2E}} U_{\alpha i} U^*_{\alpha j} \tag{A.1.10}$$

which can be expanded further, to split it into the real and imaginary components, which helps emphasise the charge-parity violation possibilities in neutrino oscillations

$$\begin{aligned} P(\alpha \to \beta)(t) = &\sum_i U_{\alpha i} U^*_{\beta i} \sum_j U^*_{\alpha j} U_{\beta j} \\ &+ 2 \text{ Re } \sum_{i>j} U^*_{\beta i} U_{\beta j} U_{\alpha i} U^*_{\alpha j} \left[e^{i\frac{(m_i^2 - m_j^2)L}{2E}} - 1 \right] \end{aligned} \tag{A.1.11}$$

Finally, we can split the exponential into components, to separate the real observables (energy, length, and mass) from the complex parts

$$\mathrm{Re}\left(e^{i\frac{\Delta m^2_{ij}L}{2E}} - 1\right) = -2\sin^2\left[\frac{\Delta m^2_{ij}L}{4E}\right] \tag{A.1.12}$$

$$\mathrm{Im}\left(e^{i\frac{\Delta m^2_{ij}L}{2E}}\right) = \sin\left[\frac{\Delta m^2_{ij}L}{2E}\right] \tag{A.1.13}$$

where $\Delta m^2_{ij} = m^2_i - m^2_j$. Now, because U is unitary, the first term in Equation A.1.11 becomes $\delta_{\alpha\beta}$. So, if we insert Equation A.1.13 and Equation A.1.11, we can obtain a final oscillation probability of

$$\begin{aligned} P(\alpha \to \beta)(t) = \delta_{\alpha\beta} \\ &- 4\ \mathrm{Re}\ \sum_{i>j} U^*_{\beta i}U_{\beta j}U_{\alpha i}U^*_{\alpha j}\sin^2\left[\frac{\Delta m^2_{ij}L}{4E}\right] \\ &+ 2\ \mathrm{Im}\ \sum_{i>j} U^*_{\beta i}U_{\beta j}U_{\alpha i}U^*_{\alpha j}\sin\left[\frac{\Delta m^2_{ij}L}{2E}\right] \end{aligned} \tag{A.1.14}$$

A.2 Mass Ordering

A complication in measuring neutrino oscillation parameters in a real experiment is that the neutrinos that are detected have travelled through matter, for example the Earth's crust. Neutrino oscillation experiments are only sensitive to the square of the neutrino mass, as seen in Equation A.1.14. This means that with an oscillation experiment, it is not possible to make a direct mass measurement. It is, however, possible to make measurement of the mass-squared differences: $\Delta m^2_{ij} \equiv m^2_i - m^2_j$.

The smallest mass-squared difference, Δm^2_{21} has been determined to greater than 0 [122] by considering the oscillations of neutrinos in the sun, with a magnitude of order 10^{-4} eV^2. The remaining mass-squared difference, Δm^2_{32} is unknown. This gives two possibilities for the neutrino mass ordering. The normal ordering (NO), where $m_3 > m_2 > m_1$, or the inverted ordering (IO), where $m_2 > m_1 > m_3$. These two possibilities are shown in Figure A.1.

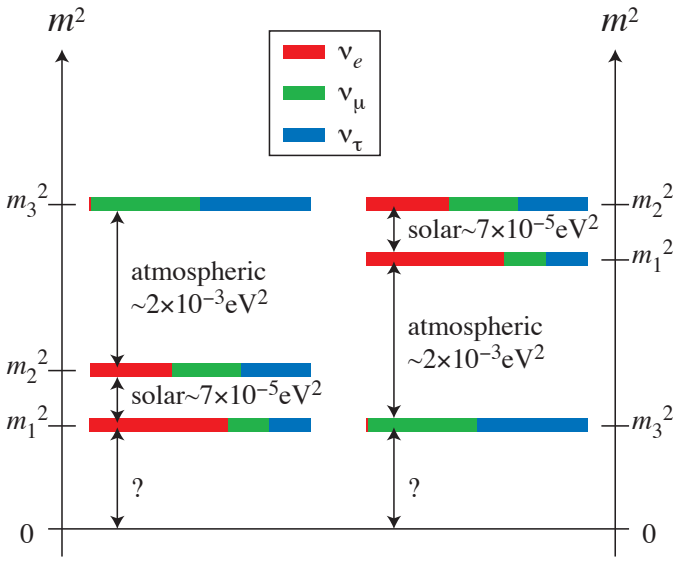


Figure A.1: The two possible neutrino mass orderings. The left is the normal ordering, and on the right the inverted ordering. Each mass eigenstate is split into its fractional flavour content, as shown by the colour. Figure from [123].

A.3 CP Violation

When considering most models of the Big Bang, it is predicted that the universe was created with equal parts of both matter and anti-matter. However, as evidenced by our existence, the universe appears to actually be dominated by matter. From this, we can assume that it is possible for some asymmetry to have occurred, resulting in this matter / anti-matter imbalance.

We believe that part of this asymmetry lies in the breaking of the charge-parity (CP) symmetry. Here, the charge operator is the conjugation of a particle for it's anti-particle, or vice versa. The parity operator, on the other hand, flips the sign of all spatial coordinates for a particle. Combined, they form the CP operator.

If Equation A.1.14 is expanded to the three-flavour paradigm and considered alongside all parts of the PMNS mixing matrix, we can parametrise the CP violating behaviour in terms of a single CP violating phase, δ_{CP}. We can see this if we look at an example oscillation probability between two flavour states, such as the first order approximation for $\nu_\mu \rightarrow \nu_e$ oscillations [124]:

$$
\begin{aligned}
P(\nu_\mu \to \nu_e) \simeq\ & \sin^2\theta_{23}\sin^2 2\theta_{13}\frac{\sin^2(\Delta_{31}-aL)}{(\Delta_{31}-aL)^2}\Delta_{31}^2 \\
& + \Bigg[\sin 2\theta_{23}\sin 2\theta_{13}\sin 2\theta_{12} \\
& \quad \frac{\sin(\Delta_{31}-aL)}{(\Delta_{31}-aL)}\Delta_{31}\frac{\sin(aL)}{(aL)}\Delta_{21}\cos(\Delta_{31}+\delta_{\mathrm{CP}})\Bigg] \\
& + \cos^2\theta_{23}\sin^2 2\theta_{12}\frac{\sin^2(aL)}{(aL)^2}\Delta_{21}^2
\end{aligned}
\tag{A.3.1}
$$

where $\Delta_{ij} = \Delta m^2_{ij} L/4E_\nu$, $a = G_F N_e/\sqrt{2}$, G_F is the Fermi constant, N_e is the number density of electrons in the Earth, L is the baseline of the experiment in km, and E_ν is the energy of the neutrino in GeV.

B

Current Knowledge

Neutrino physics is a rich section of experimental particle physics, with many neutrino oscillation measurements now, made across solar, atmospheric, reactor, and accelerator neutrino experiments. These results can be combined, to give the best estimate of the neutrino oscillation parameters. The latest results from the Nu-Fit global neutrino oscillation analysis [33] are summarised below, giving the values for both the normal and inverted hierarchy, and highlighting the major contributing experiment or experiments.

$\boldsymbol{\theta_{12}}$

The measurement and constraints on θ_{12}, the solar mixing angle, are predominantly set by solar neutrino experiments, such as SNO [125], Super Kamiokande [15] and additional data from the KamLAND experiment [126]. The current combined measurement for θ_{12} is,

$$\begin{aligned} \sin^2(\theta_{12}) &= 0.304\ ^{+0.013}_{-0.012} \qquad \text{(NO)} \\ &= 0.304\ ^{+0.013}_{-0.012} \qquad \text{(IO)} \end{aligned}$$

$\boldsymbol{\Delta m^2_{21}}$

The best measurement of Δm^2_{21} is also predominantly set by the KamLAND experiment. The current measured value is,

$$\begin{aligned} \Delta m^2_{21} &= 7.42\ ^{+0.21}_{-0.20} \times 10^{-5}\text{eV}^2 \qquad \text{(NO)} \\ &= 7.42\ ^{+0.21}_{-0.20} \times 10^{-5}\text{eV}^2 \qquad \text{(IO)} \end{aligned}$$

$\boldsymbol{\theta_{13}}$

The measurement of θ_{13}, primarily comes from reactor neutrino experiments such as Daya Bay [127], Double Chooz [128] and RENO [129], though there are additional contributions from some long baseline oscillation experiments, such as T2K [130] and NOvA [131]. The Nu-Fit global fit gives,

$$\begin{aligned} \sin^2(\theta_{13}) &= 0.02246\ ^{+0.00062}_{-0.00062} \qquad \text{(NO)} \\ &= 0.02241\ ^{+0.00074}_{-0.00062} \qquad \text{(IO)} \end{aligned}$$

$\boldsymbol{\theta_{23}}$ and $\boldsymbol{\Delta m^2_{32}}$

θ_{23} and Δm^2_{23} are strongly correlated, so it is common for their measurements to be presented together, as a two-dimensional contour. Figure B.1 shows a comparison between the contours from a variety of neutrino experiments, with the best results coming from T2K, MINOS [132] and NOvA. The current best combined three oscillation fit gives a constraint of,

$$\begin{aligned} \Delta m^2_{32} &= (+2.510\ ^{+0.027}_{-0.027}) \times 10^{-3}\text{eV}^2 \qquad \text{(NO)} \\ &= (-2.490\ ^{+0.026}_{-0.028}) \times 10^{-3}\text{eV}^2 \qquad \text{(IO)} \end{aligned}$$

and

$$\begin{aligned} \sin^2(\theta_{23}) &= 0.450\ ^{+0.019}_{-0.016} \qquad \text{(NO)} \\ &= 0.570\ ^{+0.016}_{-0.022} \qquad \text{(IO)} \end{aligned}$$

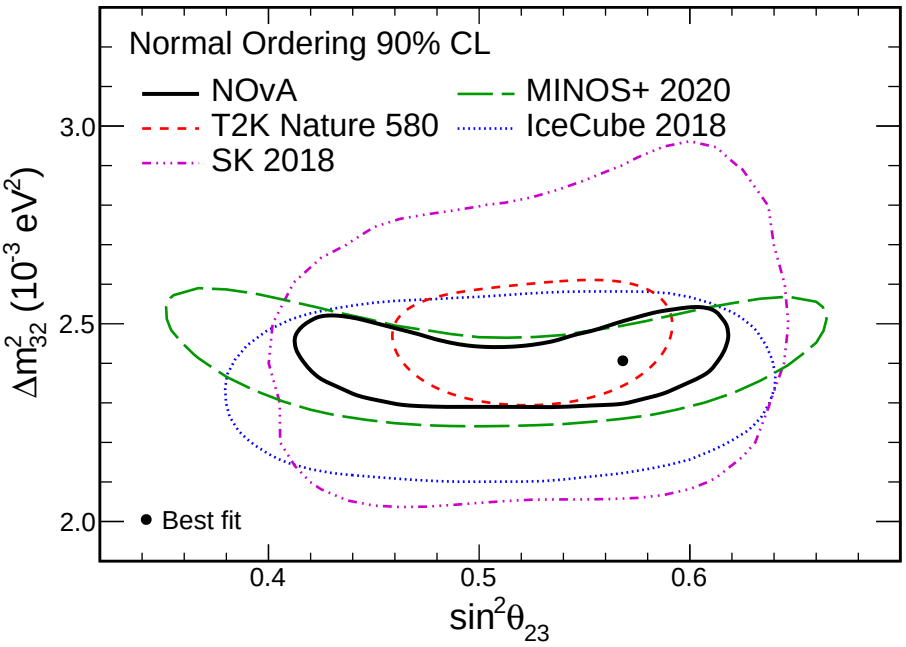


Figure B.1: The 90% confidence region contours for $\sin^2(\theta_{23})$-Δm^2_{23} from some of the leading neutrino oscillation experiments [130–132]. Figure is from [133] where the black point shows the best fit for the NOvA experiment.

Mass Ordering

Both NOvA and T2K and shown a preference for normal mass ordering [130, 131], and the combined result from the combined Nu-Fit oscillation analysis puts the inverted ordering disfavoured with a $\Delta\chi^2$ of 7.0.

δ_{CP}

There are hints from the long baseline neutrino experiments T2K and NOvA towards the value of δ_{CP} being non-zero. The limits are set based on joint fits over four distinct data samples: ν_e appearance, $\bar{\nu}_e$ appearance, ν_μ disappearance and $\bar{\nu}_\mu$ disappearance.

The NOvA experiment has completed joined fits to its neutrino and anti-neutrino oscillation data, with the results shown in Figure B.2 [133]. The normal mass ordering is preferred and the full range of possible values for δ_{CP} is allowed at 3σ.

Similarly, the result of the T2K experiments joint fit can be seen in Figure B.3 [134]. T2K sees an excess of electron neutrino events and a deficit of

anti-electron neutrino events, when compared to a baseline of $\delta_{\text{CP}} = 0$. This results in a preference for a negative value at a 3σ confidence interval of $[-3.41, -0.03]$, again in the case of normal mass ordering.

Both of these results show that there is a need for further study into CP-violation in neutrino oscillations, and this is what the next generation of long baseline accelerator neutrino experiments should be able to probe more thoroughly. The effects that different values of δ_{CP} have on the oscillation probability at both T2K and NOvA can be seen in Figure B.4.

There is a suggestion that $\delta_{\text{CP}} = 0$ and $\delta_{\text{CP}} = \pi$ are ruled out, thus hinting towards CP being violated in the lepton sector [134]. However, the 3σ confidence level does not currently exclude the two CP conserving points. So whilst there are strong hints towards CP violation (CPV), a definitive result will require the next generation oscillation experiments, such as DUNE or Hyper-Kamiokande [135].

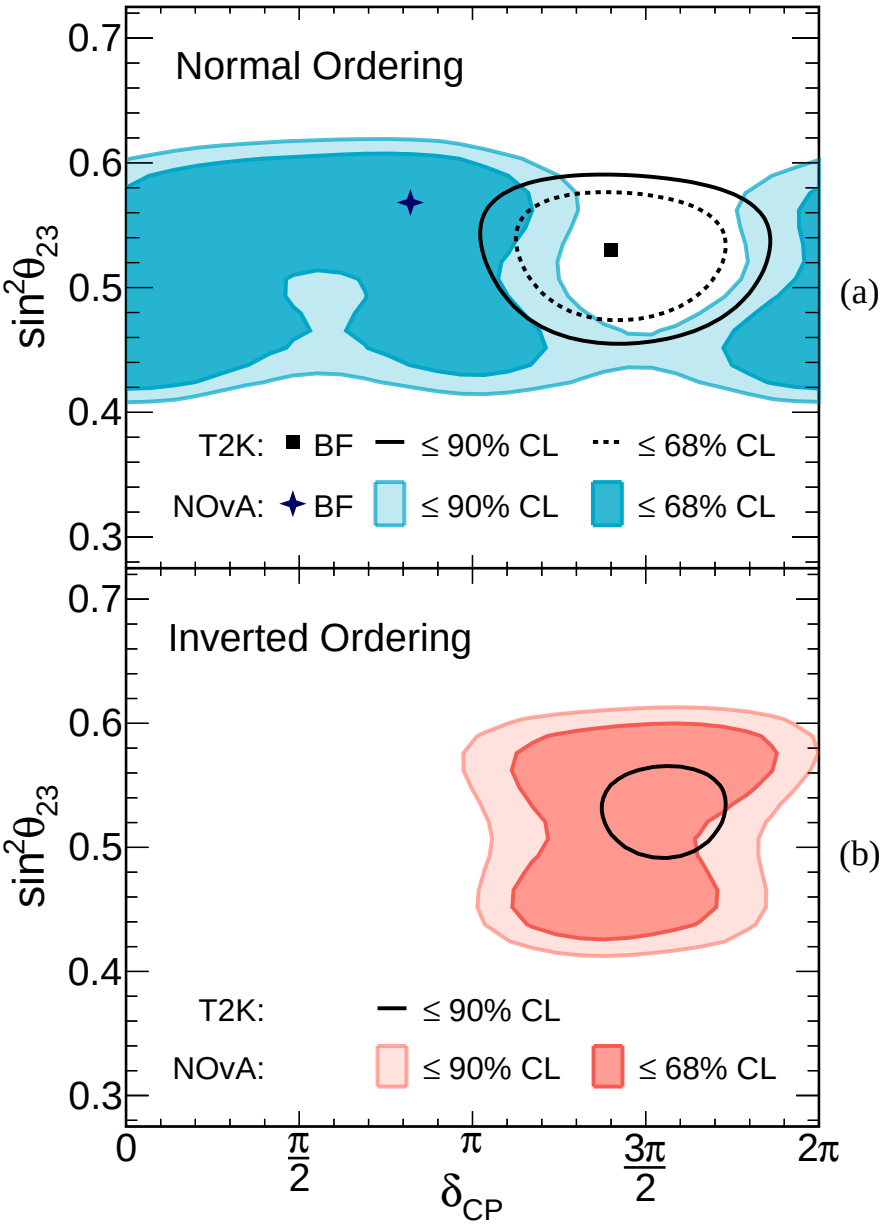


Figure B.2: Latest results from the NOvA collaboration. Figure from [133].
Figure a shows the confidence interval for δ_{CP} vs $\sin^2(\theta_{23})$ under the assumption of normal ordering. The best fit points for both NOvA and T2K are show, along with their confidence level contours.
Figure b shows the confidence interval for δ_{CP} vs $\sin^2(\theta_{23})$ under the assumption of the inverted ordering.

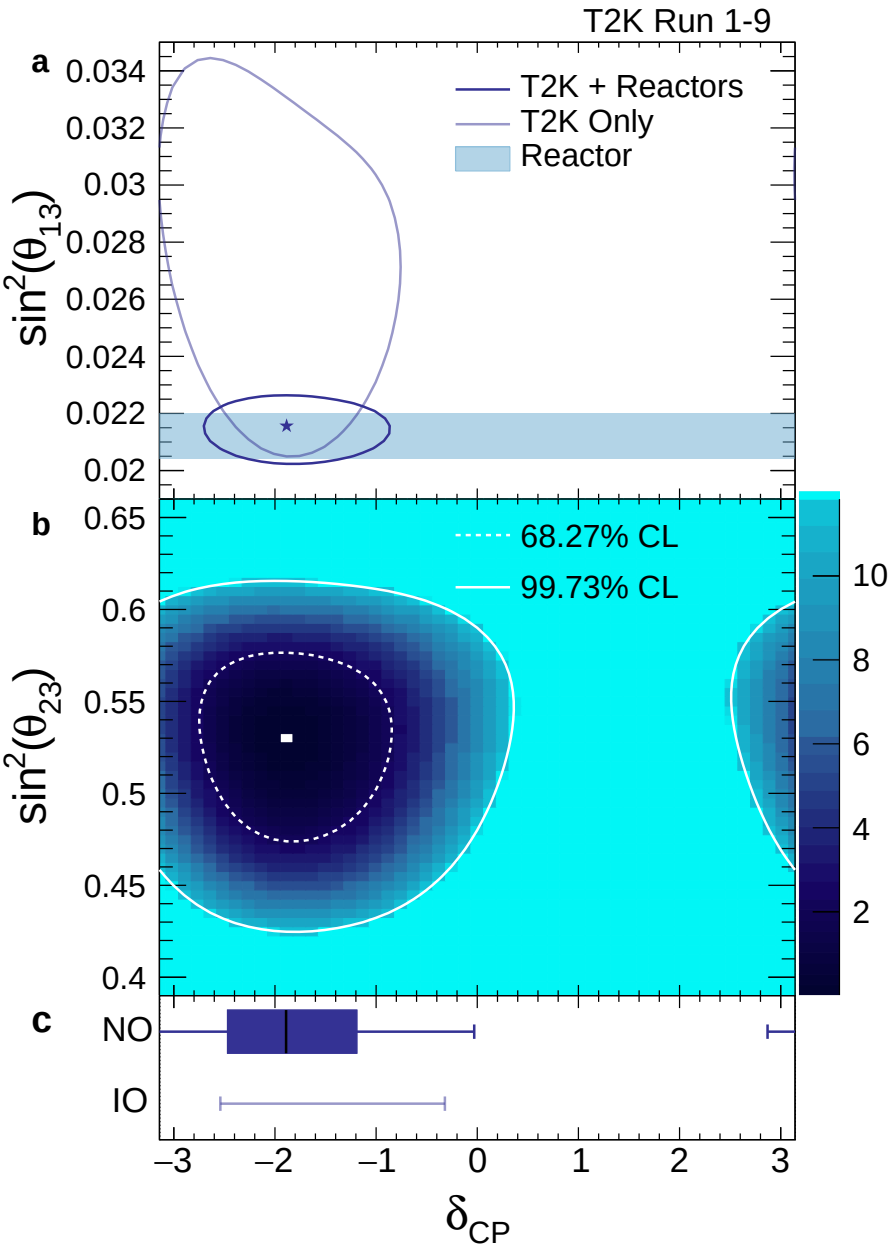


Figure B.3: The latest results from the T2K Collaboration. Figure from [134].
Figure a shows the 68.27% confidence level contours for δ_{CP} vs $\sin^2(\theta_{13})$ as measured by T2K, under the assumption of normal ordering. The star shows the best fit point for T2K plus reactor data.
Figure b shows the 68.27% and 99.73% confidence level for δ_{CP} vs $\sin^2(\theta_{23})$, under the assumption of normal ordering.
Figure c shows the confidence intervals for δ_{CP} from a fit to both T2K and reactor data, for both normal and inverted ordering. The line shows the best fit point for the value of δ_{CP}, where the shaded box and error bars show the 67.27% and 99.73% confidence intervals, respectively.

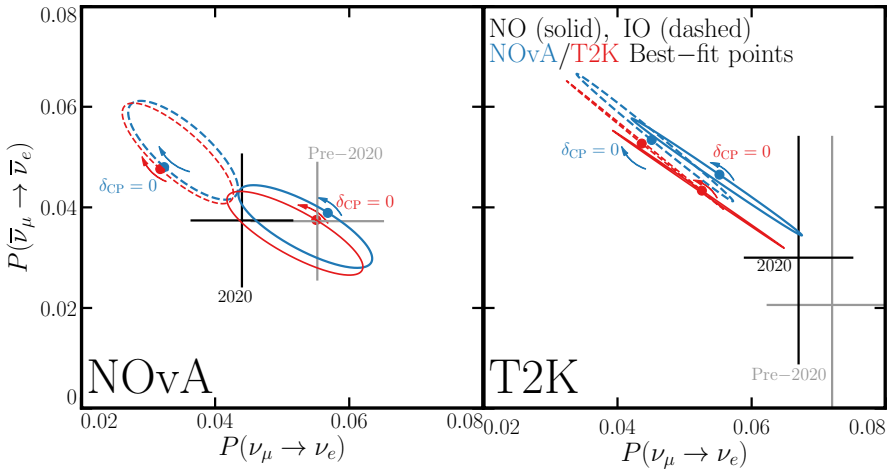


Figure B.4: Bi-probability plots of oscillation probabilities for neutrinos vs anti-neutrinos, for the L and E values of both NOvA (left panel) and T2K (right panel), whilst varying δ_{CP}. The ellipses correspond to best-fit points according to both NOvA and T2K in blue and red respectively. Normal ordering is shown as a solid line, whereas the inverted ordering is the dashed lines. The labelled points show a value of $\delta_{CP} = 0$ and arrows to indicate increasing the value of δ_{CP}. Figure is from [136].

C

Shower Growing Simulation-Data Comparison

C.1 DL growing

This shows the few remaining plots from the ProtoDUNE-SP reconstructed shower simulation and data comparison.

Additionally, some plots are included that show the results when focusing on showers with less than and greater than 450 hits, to show the difference between the two distributions. This threshold was chosen to show the differences between the two peaks seen in the following 2D and 3D number of hits plots, splitting the 'large' and 'small' showers up.

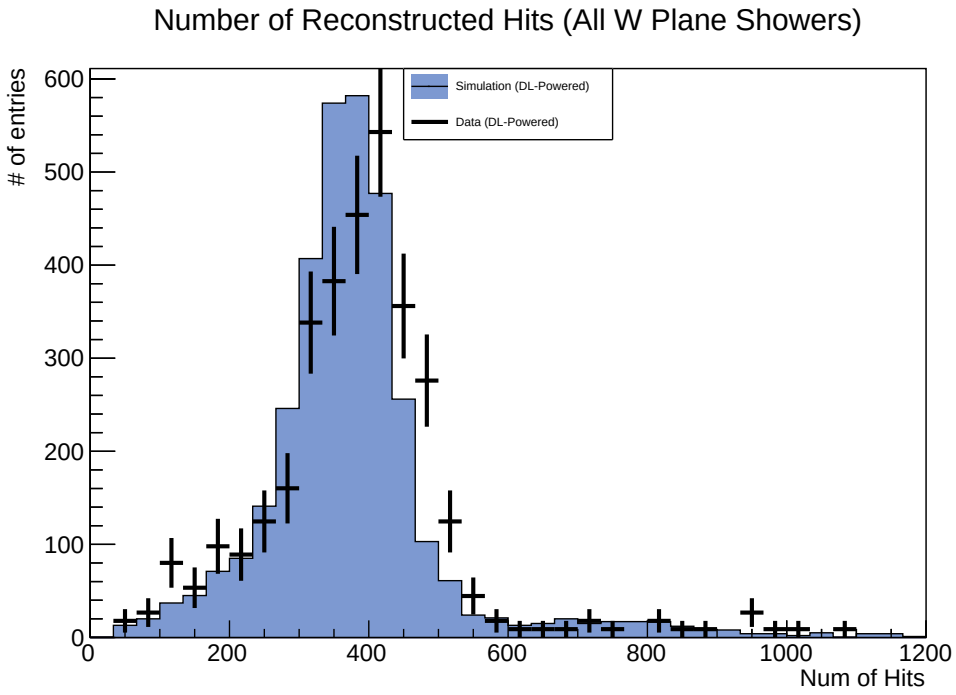


Figure C.1: Comparison of the reconstructed number of 2D hits in a shower at ProtoDUNE-SP, for the DL growing.

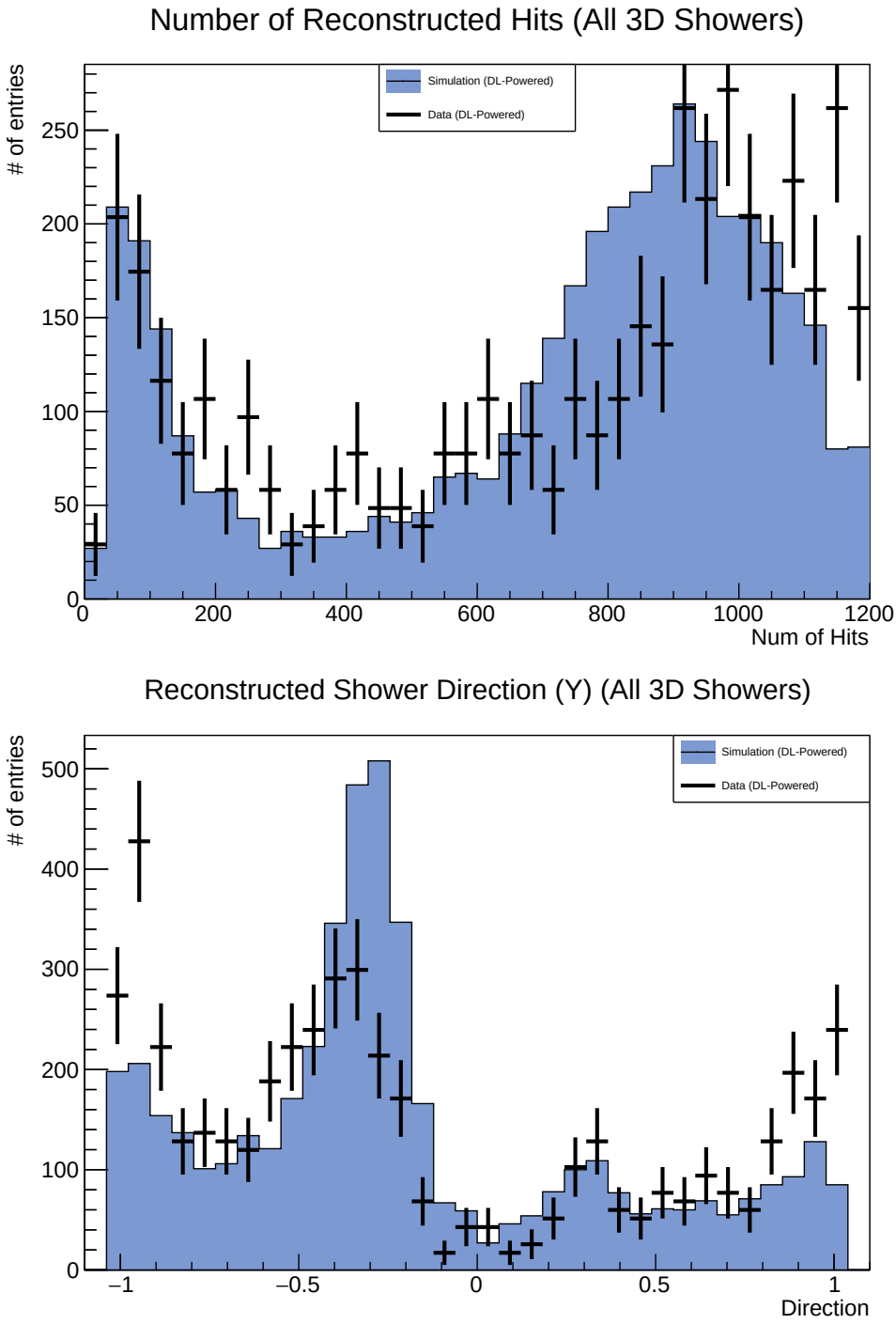


Figure C.2: Comparison of the number of 3D hits and the reconstructed 3D direction with deep learning-based growing.

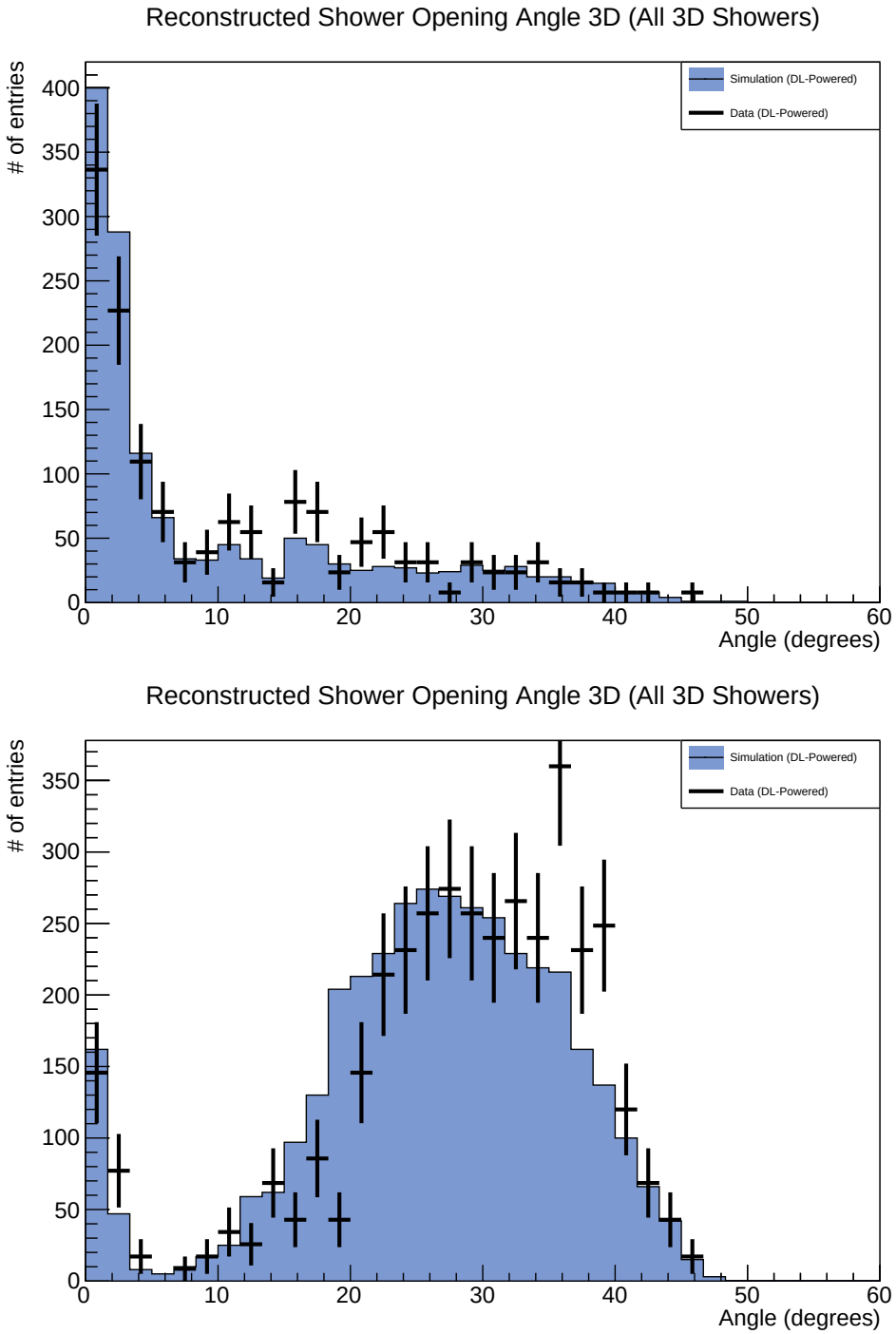


Figure C.3: Comparison between simulation and data for the reconstructed shower opening angle at ProtoDUNE-SP with the deep learning shower growing, split into small and large showers. Small here is showers with less than 450 3D hits, with large being anything above that. This threshold was chosen based on the peaks seen in Figures C.2 and C.1.

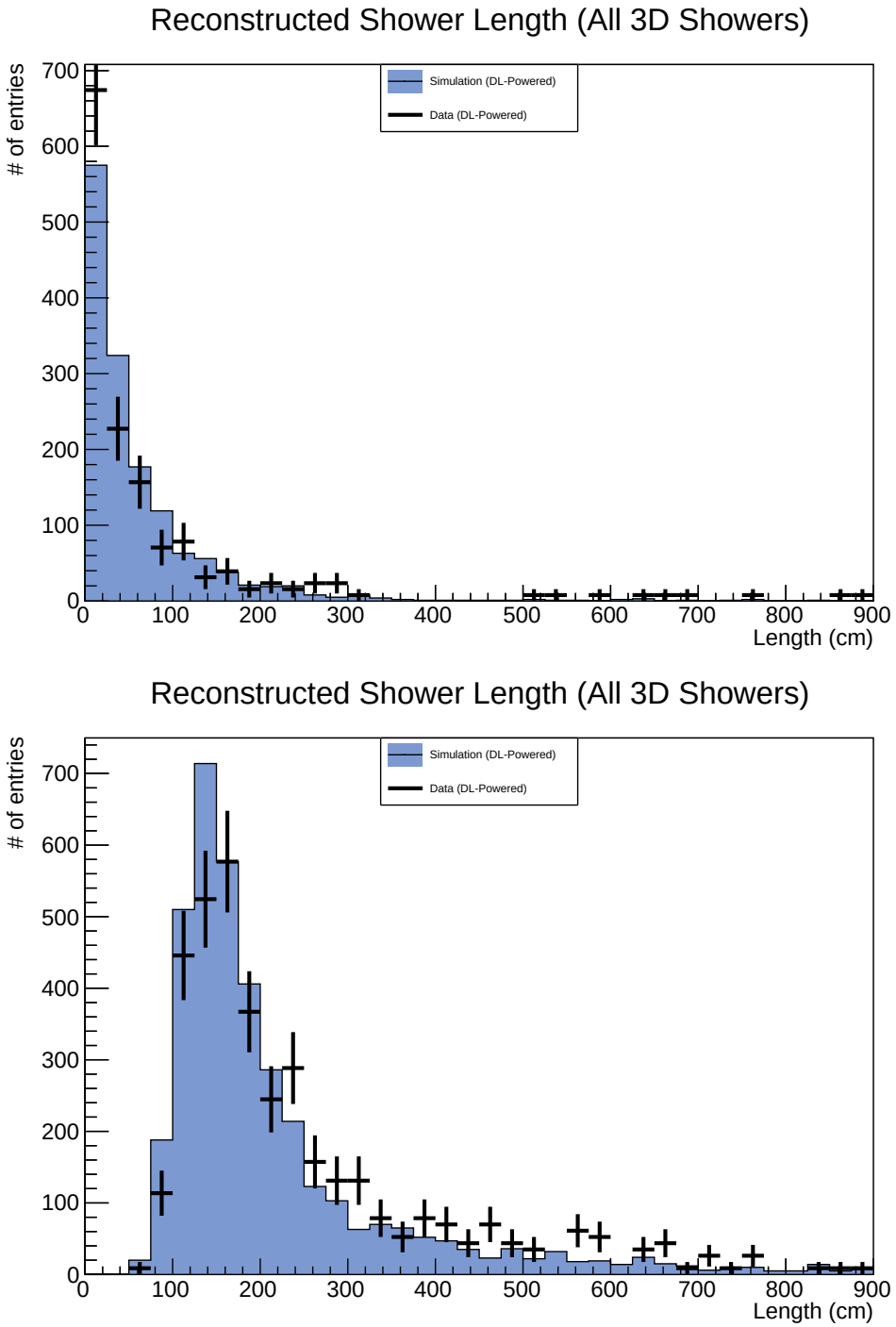


Figure C.4: Comparison between simulation and data for the reconstructed shower length at ProtoDUNE-SP with the deep learning shower growing, split into small and large showers. Small here is showers with less than 450 3D hits, with large being anything above that. This threshold was chosen based on the peaks seen in Figures C.2 and C.1.

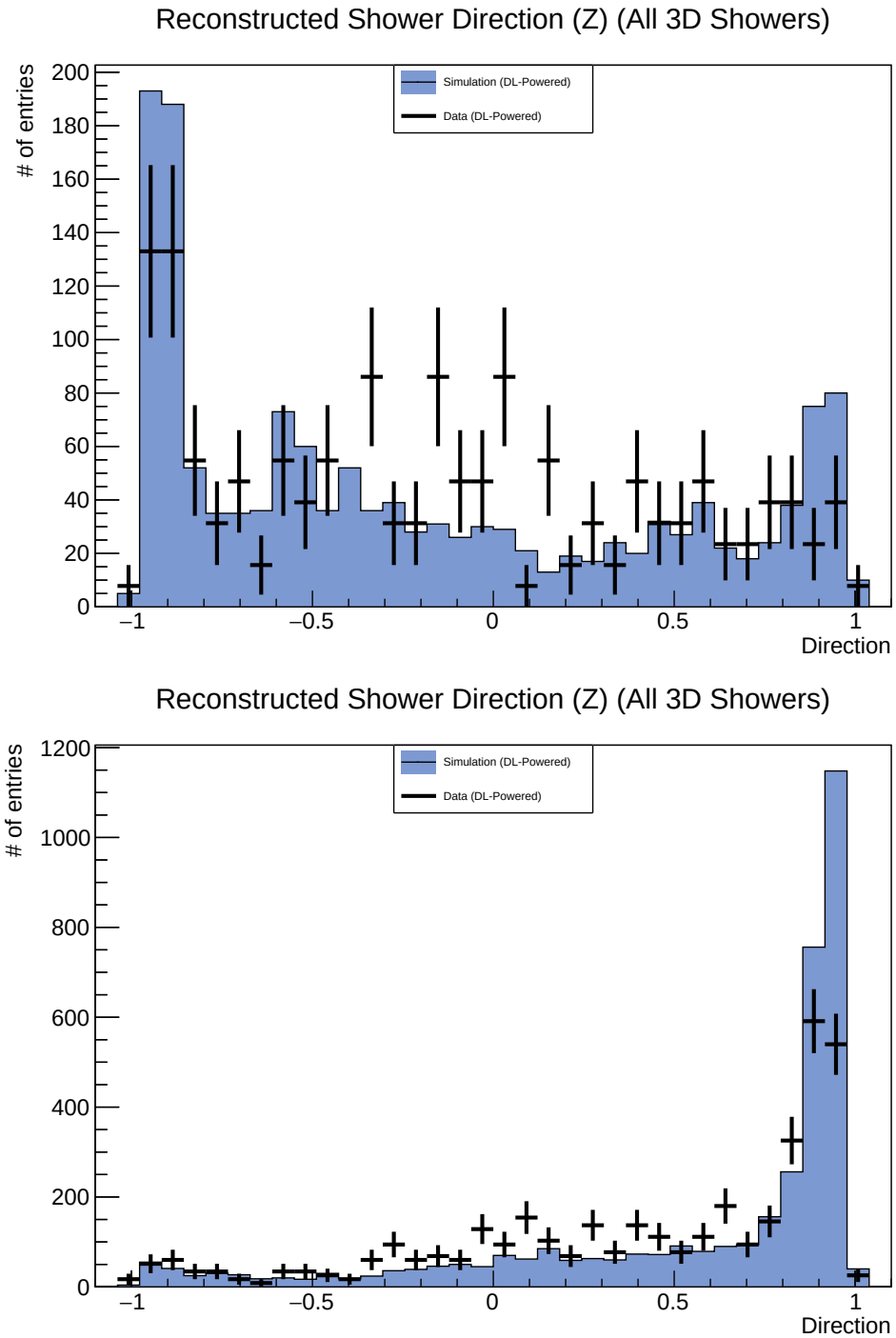


Figure C.5: A data versus simulation comparison between the small and large showers reconstructed shower direction. The first plot shows showers with less than 450 3D hits, whereas the second plot shows showers with more than 450 hits. It can be seen that the small showers contribute the most to the incorrect reconstructed shower direction.

C.2 Existing Growing

This just shows the same remaining plots as Section C.1, but for the existing shower growing, rather than the DL based shower growing. These are included to help show any differences that are down to the simulation itself, rather than any algorithm performing differently on simulation versus real data.

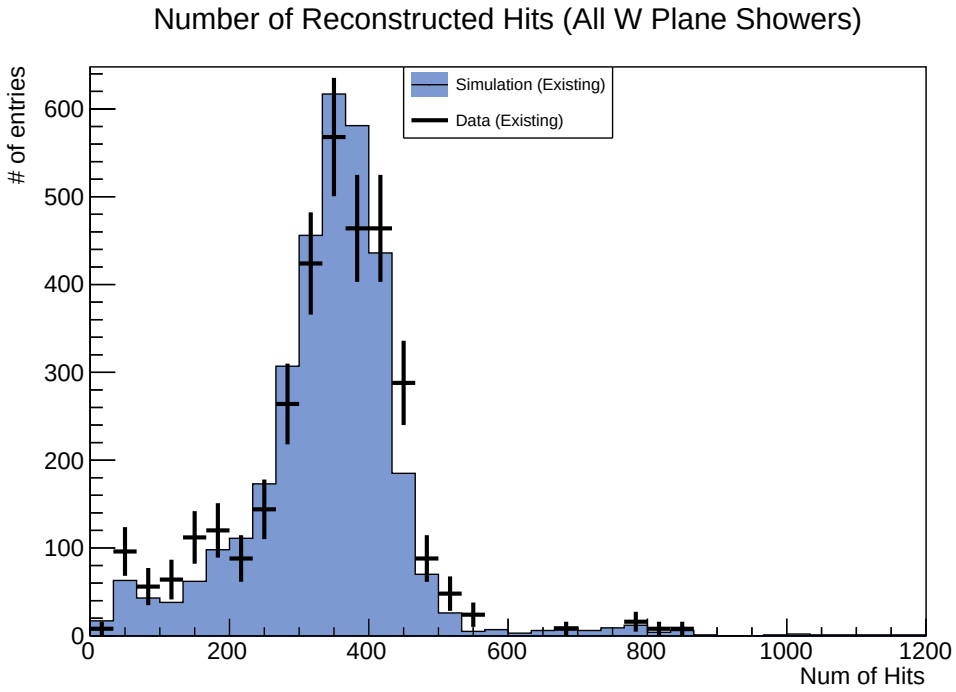


Figure C.6: Comparison of the reconstructed number of 2D hits in a shower at ProtoDUNE-SP, for the existing growing.

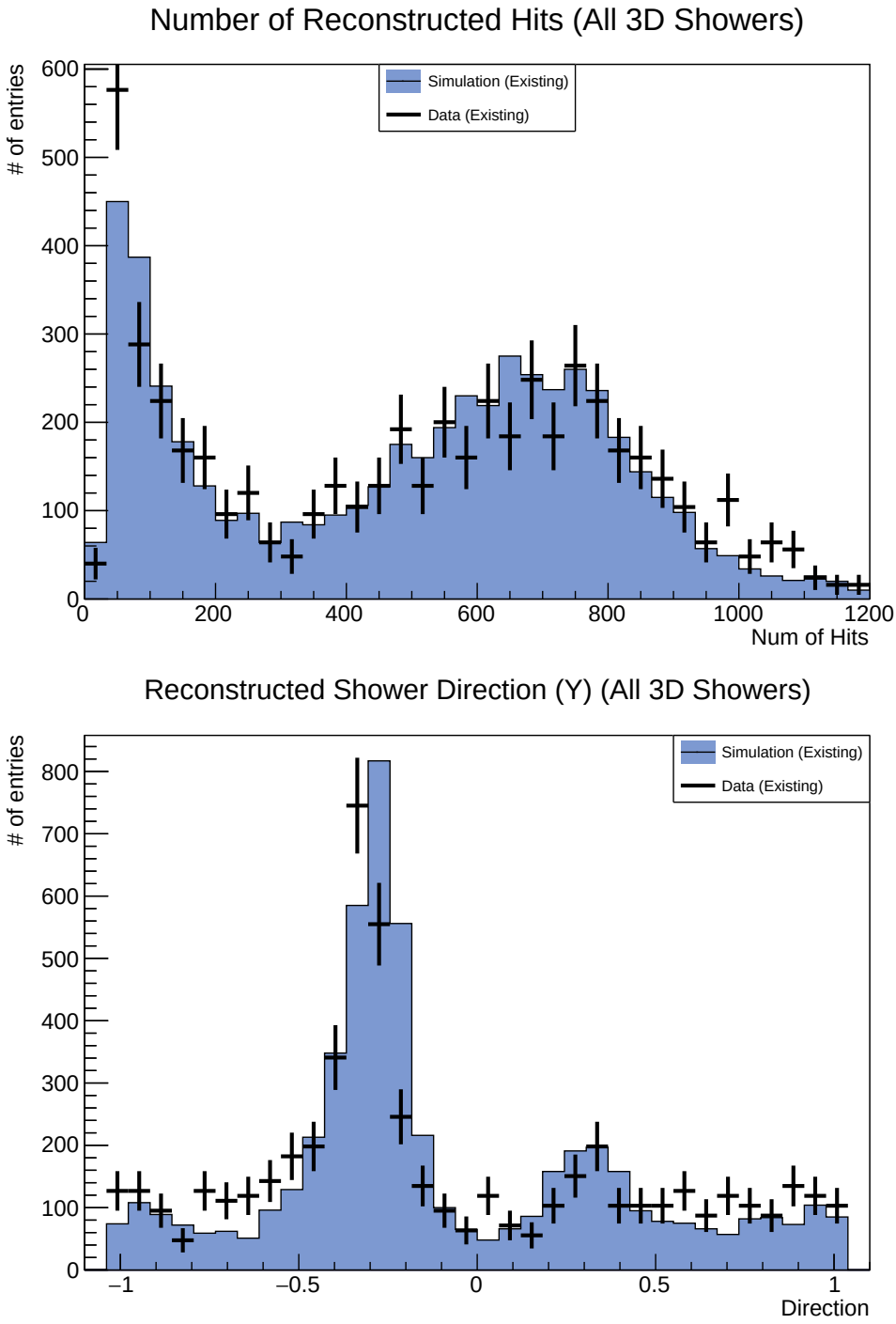


Figure C.7: Comparison of the number of 3D hits and the reconstructed 3D direction with the existing growing.

C.3 Vertex Position

A comparison of the vertex position for both the existing and new shower growing. This step should only be minimally impacted by the shower growing, so no larger change is expected.

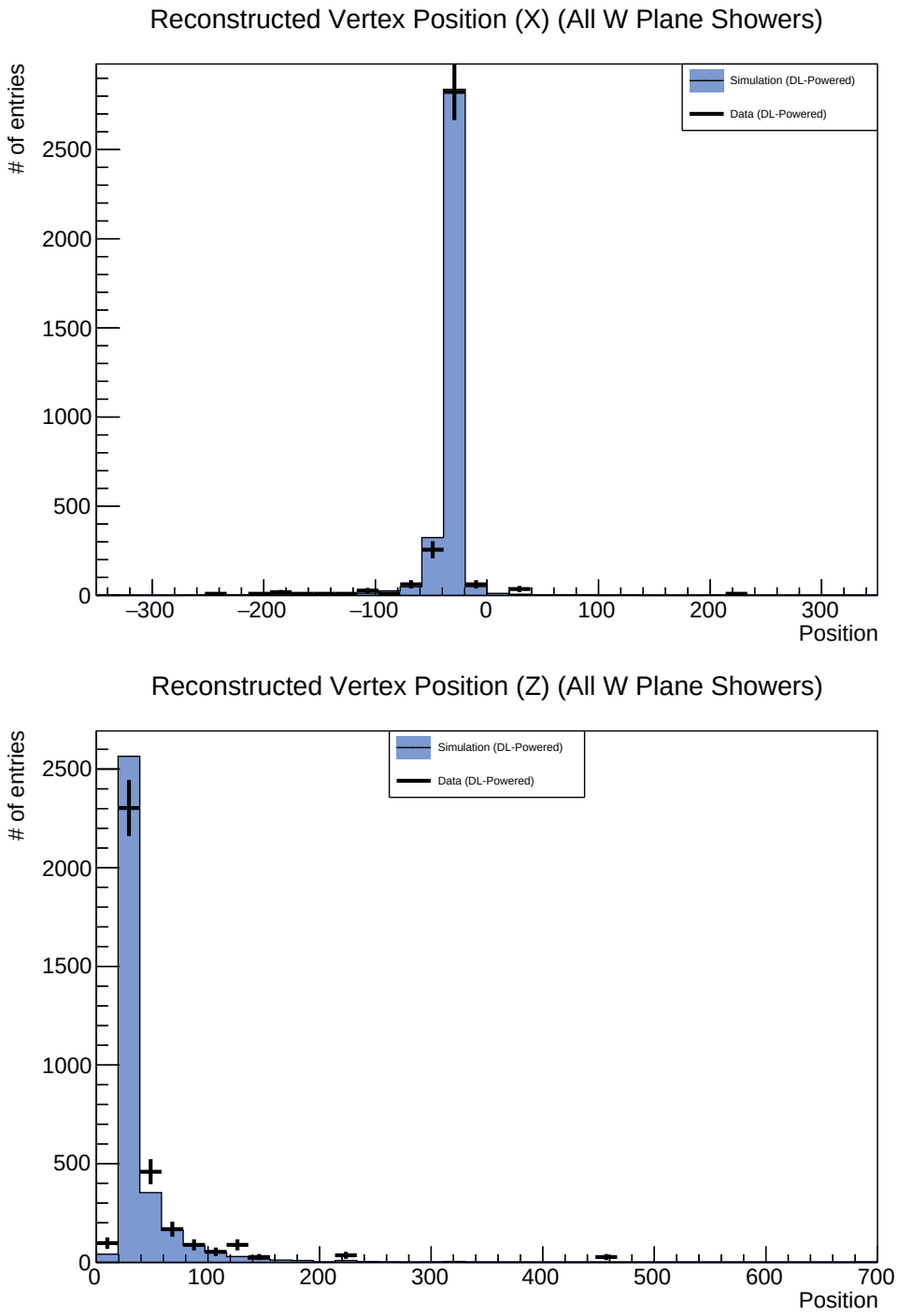


Figure C.8: Data and simulation comparison of the reconstructed 2D vertex position at ProtoDUNE-SP for the DL growing.

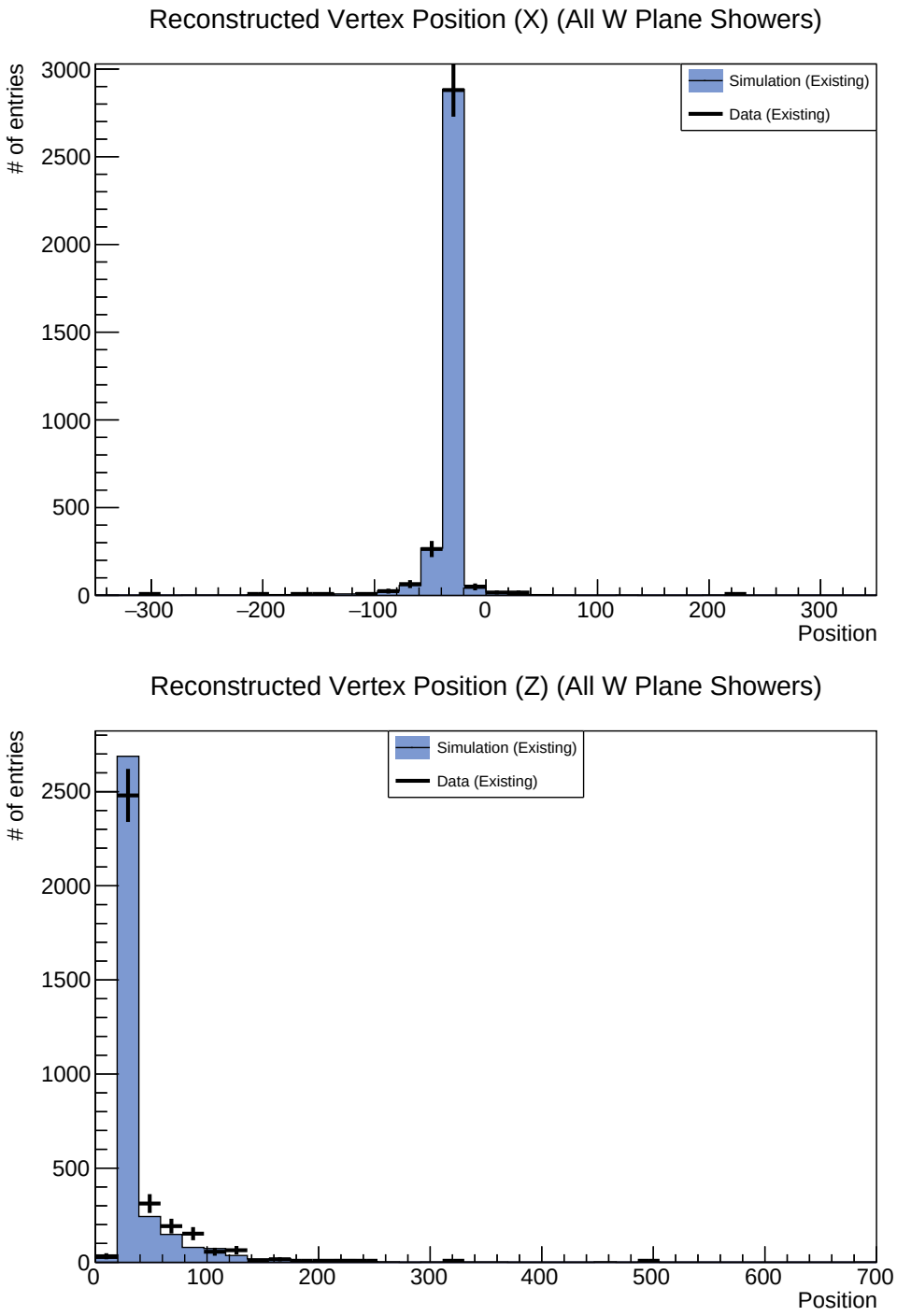


Figure C.9: Data and simulation comparison of the reconstructed 2D vertex position at ProtoDUNE-SP for the existing growing.

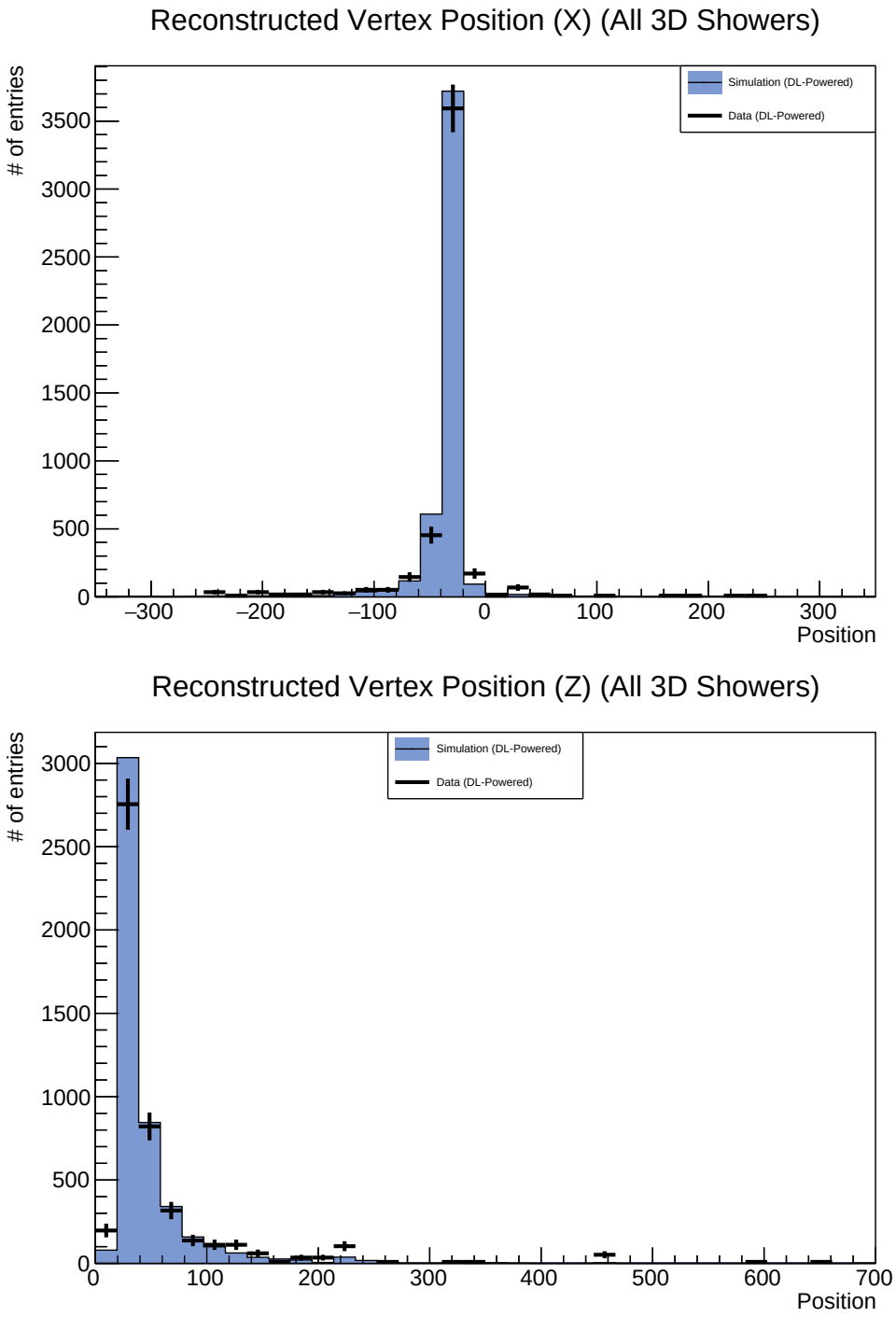


Figure C.10: Data and simulation comparison of the reconstructed XZ 3D vertex position at ProtoDUNE-SP for the DL growing.

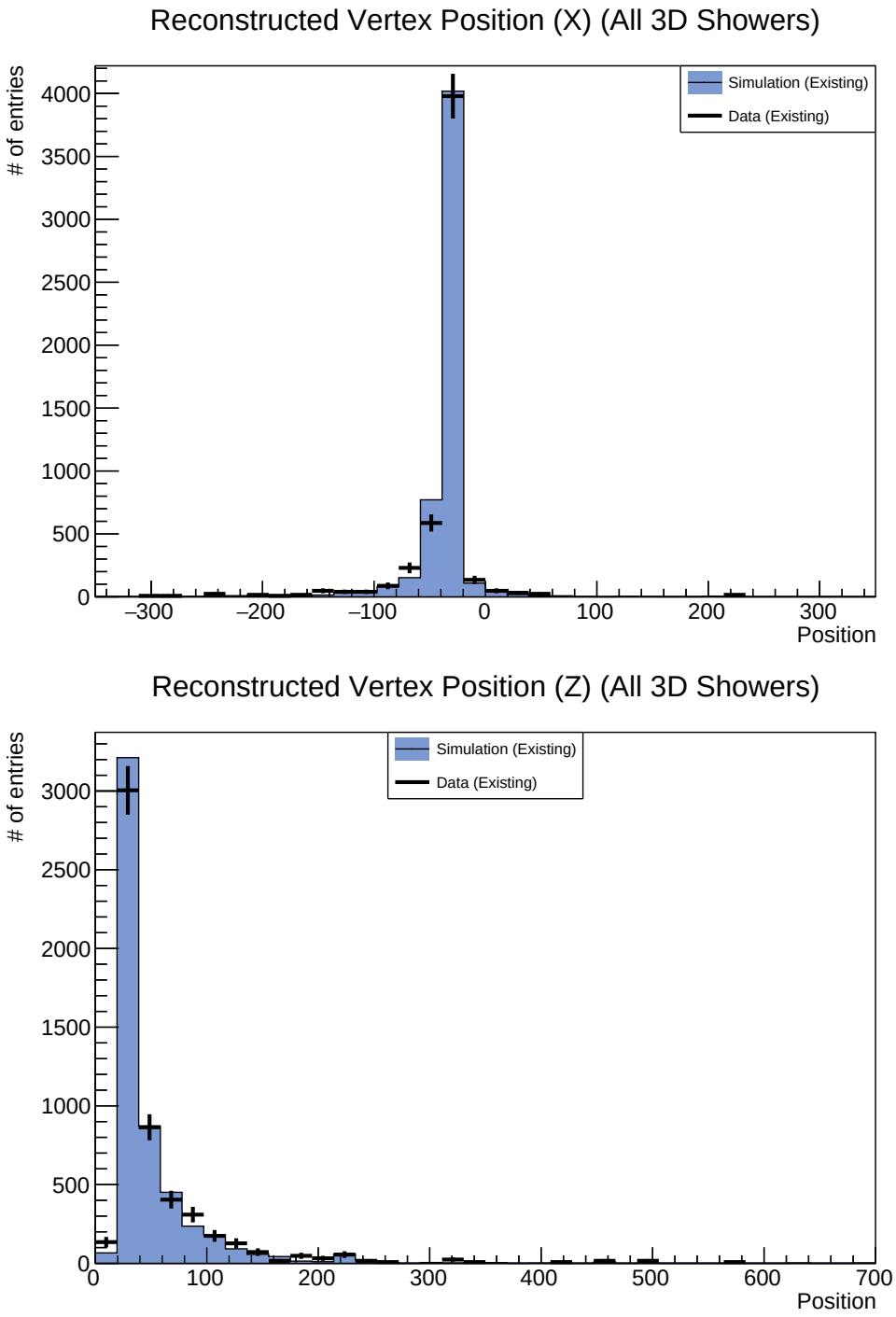


Figure C.11: Data and simulation comparison of the reconstructed XZ 3D vertex position at ProtoDUNE-SP for the existing growing.

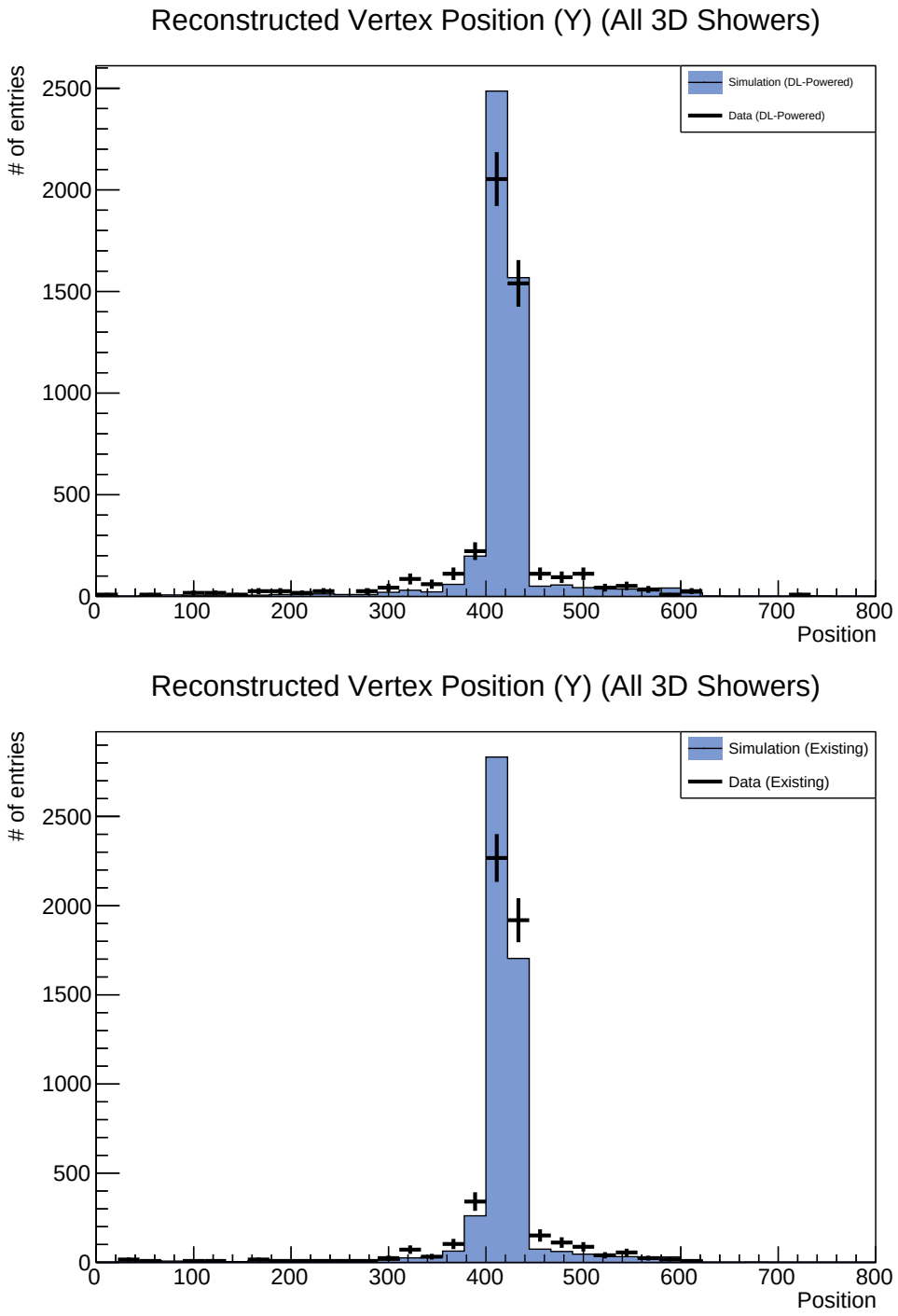


Figure C.12: Data and simulation comparison of the reconstructed Y 3D vertex position at ProtoDUNE-SP for both shower growing algorithms.