THE UNIVERSITY OF VIRGINIA

Angular Dependence of Drell-Yan in the SeaQuest Experiment using Deep Neural Network-Based Reconstruction

by

Arthur Conover
Bachelor of Arts in Physics, Kenyon College, 2017
Master of Science in Physics, The University of Virginia, 2020

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

to the
Faculty of the
Department of Physics

July 2024

Declaration of Authorship

I, Arthur Conover, declare that this thesis titled, 'Angular Dependence of Drell-Yan in the SeaQuest Experiment using Deep Neural Network-Based Reconstruction' and the work presented in it are my own. I confirm that:

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

Signed: 

Date: 7/30/2024

Abstract

The SeaQuest and SpinQuest experiments at Fermilab were designed to probe the internal dynamics of protons and neutrons via the angular dependence of muons created by colliding 120 GeV protons into stationary targets. To do this, it is necessary to translate the detector data into coherent information about the particles detected in the experiment. This dissertation describes the development and implementation of QTracker, a neural network-based algorithm designed to reconstruct muons.

The application of QTracker to experimental data yields improved reconstruction of muon tracks, allowing for a more precise investigation of the transverse momentum distributions and angular modulations in the Drell-Yan process. This work highlights the potential of neural network-based methods to advance particle tracking and enhance our understanding of nucleon structure.

Acknowledgements

I would like to express my gratitude to my advisor, Dustin Keller, for his support, guidance, and encouragement throughout my research and dissertation process. Your expertise and feedback have been invaluable. I also want to thank Donal Day, who introduced me to the UVA Polarized Target group, for his guidance and conversations over the last six years.

I would also like to extend my thanks to my friends and colleagues who have been a constant source of support and motivation. Your companionship and intellectual discussions have enriched my academic journey and made this experience both enjoyable and rewarding.

To my parents, Sharon and Charles, your love, encouragement, and belief in me have been the foundation of my success. As long as I can remember, you have encouraged my curiosity and love of learning, even when I'm sure you would have preferred I stopped asking so many questions.

Finally, I would like to express my deepest appreciation to my wife, Nicole Story. Your love, support, and patience have been a life jacket keeping me afloat during this entire process. Thank you for being my partner and best friend.

This dissertation would not have been possible without the support and encouragement of all these wonderful people. Thank you all from the bottom of my heart.

Contents

List of Figures

Chapter 1

Introduction

The SeaQuest (Fermilab E906) and SpinQuest (Fermilab E1039) experiments at Fermilab were designed to probe the internal dynamics of protons and neutrons. These experiments focus on the Drell-Yan process, which is extremely useful for studying the partonic structure of nucleons. This dissertation explores the application of deep neural network-based techniques for reconstructing muon particle tracks, with a primary emphasis on the development and implementation of QTracker, a novel neural network-based reconstruction algorithm.

Probing the internal structure of protons and neutrons has been a goal of nuclear physics for decades. Understanding how quarks and gluons, the fundamental constituents of nucleons, are distributed and how they interact is essential for a comprehensive theory of strong interactions. The SeaQuest and SpinQuest experiments aim to provide insights into these questions by measuring the Drell-Yan process, where a quark from a beam nucleon and an antiquark from a target nucleon annihilate to produce a muon pair. The angular distributions and transverse momentum spectra of these muons are sensitive to the partonic structure and dynamics within the nucleons.

Traditional methods of tracking particles in high-energy physics experiments rely on geometrical and statistical techniques to reconstruct the paths of charged particles through detectors. However, these methods face limitations in terms of accuracy and computational efficiency, especially in environments with high noise. Machine learning, particularly deep learning, offers promising alternatives for particle track reconstruction. Neural networks' ability to model complex, non-linear

relationships can potentially outperform conventional algorithms in both precision and speed.

The SeaQuest experiment (E906) at Fermilab employed a 120 GeV proton beam from the Fermilab Main Injector to bombard various fixed targets, including liquid hydrogen and deuterium. The primary objective was to measure the Drell-Yan process and extract information about the antiquark content of the nucleon. Building on SeaQuest, the SpinQuest experiment (E1039) employs a polarized target to study spin-dependent effects in the same process. These experiments utilize a spectrometer equipped with multiple detectors, including drift chambers, hodoscopes, and proportional tubes, to measure the trajectories of produced particles with high precision.

The complexity and volume of the data in these experiments necessitates advanced algorithms for accurate track reconstruction. This is where QTracker, the focus of this dissertation, comes into play. QTracker uses the power of deep neural networks to speed up and improve the reconstruction of muon tracks, thereby enhancing the quality of the data analysis.

QTracker is a deep learning-based reconstruction algorithm specifically designed to process data from the SeaQuest and SpinQuest experiments. It comprises several neural network models, each tailored to different aspects of the reconstruction process. The key components of QTracker include:

- Event Filter: A neural network that filters out noise and irrelevant events from the raw data, ensuring that only meaningful events are processed further.
- Track Finder: This component identifies potential muon tracks within the filtered events. It uses a convolutional neural network (CNN) architecture to recognize patterns indicative of particle tracks.
- Momentum Reconstruction: Once potential tracks are identified, this model evaluates the three-momentum of the particles that generated that track.
- Vertex Finder: This component reconstructs the interaction vertices, providing crucial information about the points where particles originated.

The development of QTracker involved training and validation using both simulated and real experimental data. The training process used labeled datasets

generated by Monte Carlo simulations, which attempt to accurately represent the physical processes and detector responses. The performance of QTracker was benchmarked against traditional tracking algorithms, demonstrating significant improvements in both speed and precision.

By enhancing the accuracy of muon track reconstruction, QTracker enables more precise measurements of the Drell-Yan process. This, in turn, could lead to better constraints on parton distribution functions (PDFs) and a deeper understanding of nucleon structure. Moreover, the techniques developed for QTracker can be generalized and applied to other high-energy physics experiments, paving the way for broader adoption of deep learning in particle physics.

In addition, this dissertation illustrates the transformative potential of neural networks in experimental physics. It showcases how interdisciplinary approaches, combining physics with advanced computational techniques, can address long-standing challenges and open new avenues for research.

This dissertation is structured as follows:

- Chapter 2 provides a detailed review of the theoretical framework underpinning the SeaQuest and SpinQuest experiments, including the Standard Model of particle physics and the specifics of the Drell-Yan process.
- Chapter 3 describes the experimental setup, including the Fermilab Main Injector, the targets, and the detection systems. It also describes a novel method we have developed to irradiate large amounts of material for polarized target experiments in an efficient way.
- Chapter 4 details the traditional reconstruction methods and introduces QTracker, detailing its design, training, and validation.
- Chapter 5 presents the application of QTracker to experimental data, highlighting its performance and the insights gained from the improved track reconstruction.
- Chapter 6 discusses the broader implications of this work and potential future directions for research and technology development in particle tracking.

By integrating deep learning techniques into the analysis pipeline, this dissertation not only advances the specific goals of the SeaQuest and SpinQuest experiments

but also contributes to the broader field of particle physics by demonstrating the utility of machine learning in high-energy physics research.

Chapter 2

Physics Theory

In this chapter, we will discuss the physics theory behind the SeaQuest and SpinQuest experiments. First, we will explore on the standard model, its history, and the parts of it pertaining to the strong nuclear force. We will then examine the physics of protons and neutrons (collectively referred to as nucleons). Finally, different scattering processes, including the main ones present in SeaQuest and SpinQuest, will be covered.

2.1 The Standard Model

The Standard Model is one of the most successful theories in the history of science. It has been developed and explored over the past 50 years and has provided numerous predictions that experiments have later confirmed. It describes the interactions of the fundamental building blocks of our universe and the forces that drive those interactions.

2.1.1 History of the Standard Model

After the success of Quantum Electrodynamics (QED) in the late 1940s in explaining the electromagnetic force, theoretical physicists turned their attention to the weak and strong forces. Unfortunately, the methods that had proved so effective in QED, namely perturbation theory, proved ineffective in explaining the other forces.[19]

In the weak force, the then-prevalent four-fermion theory posited by Enrico Fermi in 1933[20] worked well at the lowest-order, but the next order introduced non-removable infinities. The strong force, meanwhile, faced another problem: Yukawa's scalar theory of elementary particle interaction[21] proved inhospitable to perturbation theory.

The deeper problem was that all of the leading theories had no cohesive links between them. They were all mainly phenomenological, and the strong force theories had no concrete evidence to support any one or another. These difficulties led many physicists to abandon quantum field theory (QFT) in favor of other theories to explain the strong and weak theories. None proved successful.[19]

Through the 1950s and 1960s, the individual parts of the Standard Model were developed piecemeal by many physicists. Yan Chen-Ning and Robert Mills developed a non-abelian gauge theory in 1954. Extending the abelian gauge theory used in QED to allow it to explain the strong force.[22] In 1956, Chien-Shiung Wu conducted a groundbreaking experiment that showed parity violation in the weak force.[23] Sheldon Glashow was able to combine the electromagnetic and weak forces in 1961, which was combined with the Higgs mechanism by both Steven Weinberg and Abdus Salam later in the decade.[24] [25] [26] The three later shared the Nobel Prize in 1979 for this theory.

Parallel to developing the theories for the weak force, work was being done to explain the strong force that holds protons, neutrons, and other hadrons together. In 1964, both Murray Gell-Mann and George Zweig formulated theories that included fundamental particles, quarks, that, when combined in different ways, form all the hadrons being discovered en-mass by new accelerator-based experiments.[27][28] At the end of the decade, Richard Feynman published a paper in which he presented a model to describe hadron collisions utilizing partons, which overlapped with Gell-Mann and Zweig's theories.[29] In 1973, David Politzer, David Gross, and Frank Wilczek discovered that interactions between particles reduced in strength at shorter and shorter distances (are asymptotically free) in Quantum Chromodynamics (QCD).[30][31]

This development was the final piece to the Standard Model of physics, which has remained essentially unchanged in the last 50 years. Equation 2.1 shows the simplest form of the Standard Model Lagrangian, which encodes the behavior of electromagnetism, the strong force, and the weak force. The theory includes 17

fundamental particles, which are shown in figure 2.1. Although we know that this model is not a complete theory (it does not include the gravitational force and cannot explain dark matter and dark energy), it represents one of the most successful theories scientists have ever formulated.

$$\mathcal{L} = -\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + i\bar{\psi}\not{D}\psi + h.c. + \bar{\psi}_i y_{ij}\psi_j\phi + h.c. + |D_\mu\phi|^2 - V(\phi) \tag{2.1}$$

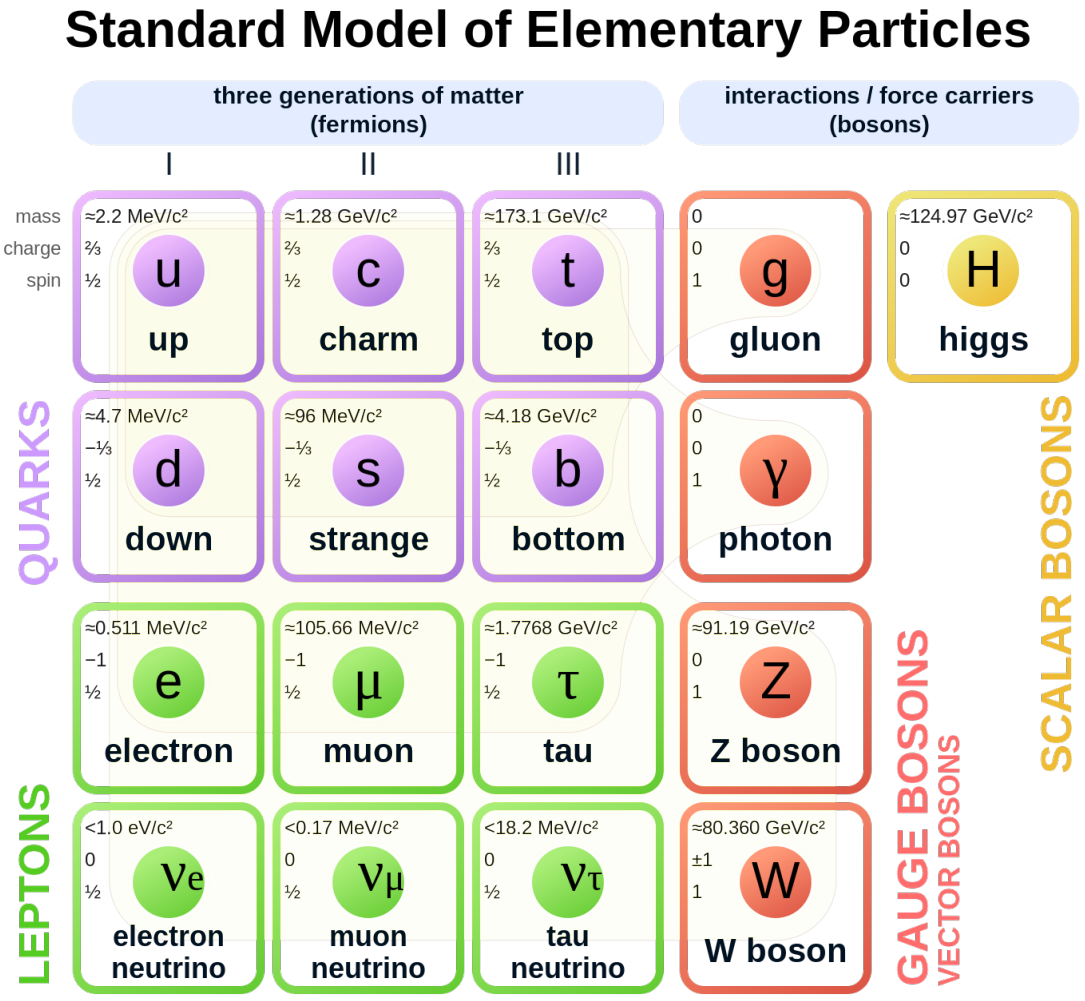


FIGURE 2.1: The 12 fermions and the 5 bosons of the Standard Model. [1]

2.1.2 Generalized Hadronic Physics

A hadron is a bound state of two or more quarks held together by strong-force mediating gluons. The most famous and common hadrons are protons and neutrons, which make up the vast majority of all matter. The nucleons are both baryons, a

term encompassing all hadrons with an odd number of valence quarks, as opposed to mesons, which contain an even number.

As shown in figure 2.1, there are six flavors of quark, split into three generations. They are up, down, charm, strange, top, and bottom quarks. They each have a charge, spin, and isospin, which together define the overall properties of the hadron they comprise. For instance, the neutron is a combination of an up quark and two down quarks, as given by the wave function:

$$\left|n^{\uparrow}\right\rangle = \frac{1}{6}(2\left|d^{\uparrow}d^{\downarrow}u^{\uparrow}\right\rangle - \left|d^{\uparrow}d^{\uparrow}u^{\downarrow}\right\rangle - \left|d^{\downarrow}d^{\uparrow}u^{\uparrow}\right\rangle) \tag{2.2}$$

Since the up quark has a charge of $+2/3$ and the down quark has a charge of $-1/3$, the neutron has a net charge of zero and a spin of $+1/2$.

In addition to the valence quarks, there are also a large number of virtual quark-antiquark pairs present inside a given hadron, known as the sea quarks. Although they are virtual and therefore only exist for a very short amount of time (as defined by the uncertainty principle), they do contribute to the overall properties of hadrons and can be observed in scattering experiments.

Quarks cannot exist in isolation, a property known as color confinement.[32] In addition to electric charge and spin, all quarks (antiquarks) carry one of three color (anti-color) charges , referred to as red, green, and blue (anti-red, anti-green, and anti-blue). The color charge is the charge for the strong force, analogous to the electrical charge.

The necessity of the color charge arises from the Pauli exclusion principle, which states that no two spin 1/2 particles in a system may occupy the same quantum state. If we consider the Δ^{++} baryon, which is a spin 3/2 particle comprised of three up quarks, it would naively have a wave function given by

$$\left|\Delta^{++}\right\rangle = \left|u^{\uparrow}u^{\uparrow}u^{\uparrow}\right\rangle \tag{2.3}$$

If this were the case, however, it would break the Pauli principal, as switching any of the three quarks would result in no change to the system. To fix this,

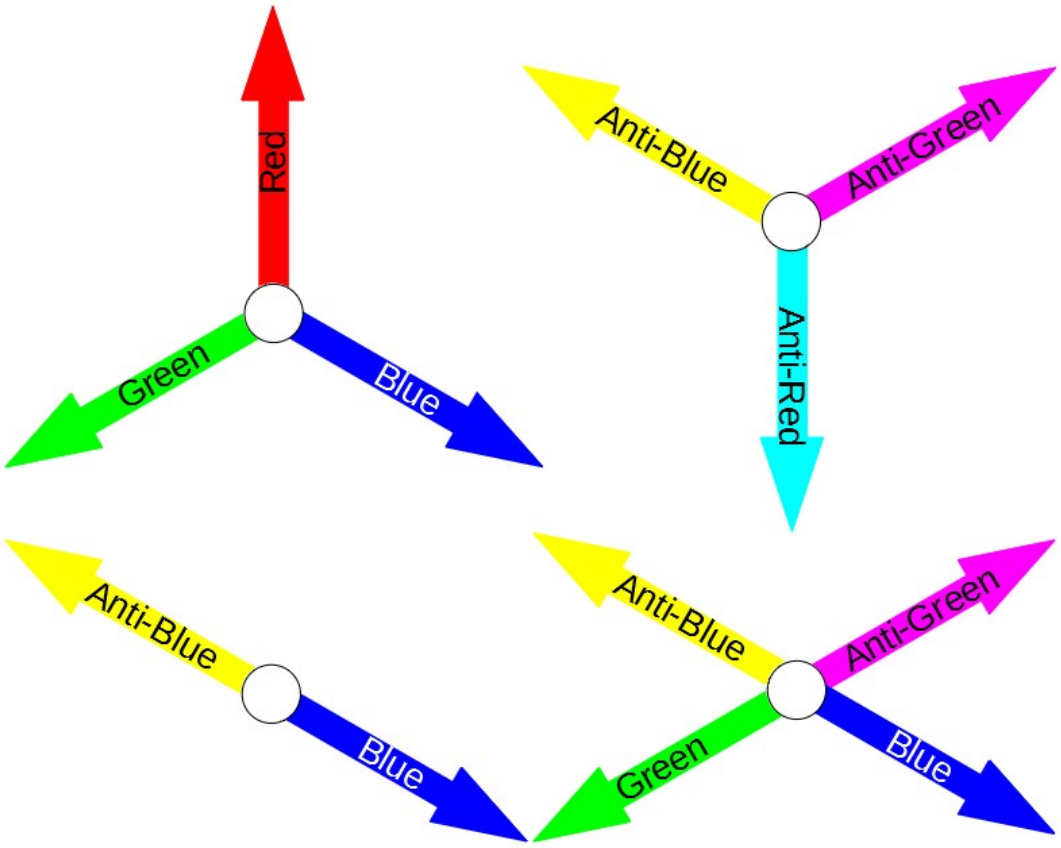


FIGURE 2.2: Examples of colorless hadrons: a triquark baryon with red, green, and blue quarks, a triquark baryon with anti-red, anti-green, and anti-blue quarks, a meson with blue and anti-blue quarks, and a tetraquark meson with blue, anti-blue, green, and anti-green quarks.

there must be some difference between the up quarks comprising the baryon. If we assign them each a color, red, blue, and green, we can redefine the Δ^{++} wave function as

$$\left|\Delta^{++}\right\rangle = \sqrt{\frac{1}{6}}\epsilon^{\alpha\beta\gamma}\left|u_\alpha^\uparrow u_\beta^\uparrow u_\gamma^\uparrow\right\rangle, \tag{2.4}$$

with α, β, γ as color quantas, which eliminates the issue.

All quarks are found within hadrons, which must be net-colorless. Colorlessness can be achieved in one of two ways: having an equal number of red (anti-red), blue (anti-blue), and green (anti-green) quarks, or having an equal number of color and anti-color quarks (red and anti-red, etc.). Examples of possible combinations are shown in figure 2.2.

As previously mentioned, the force-mediating boson for the strong force is the gluon. Unlike the photon, the force-mediating boson for the electromagnetic force, the gluon carries the charge for the force that it mediates. This drastically complicates the theory surrounding the strong force, as it is possible for gluons to interact with other gluons.

Gluons carry both color and anti-color charges in a quantum state. Because the possible color combinations are given by a 3×3 group, gluons can come in either a singlet state or an octet state. The octet states can be given by

$$r\bar{g}, r\bar{b}, g\bar{b}, g\bar{r}, b\bar{r}, b\bar{g}, \sqrt{\frac{1}{2}}(r\bar{r} - g\bar{g}), \sqrt{\frac{1}{6}}(r\bar{r} + g\bar{g} - 2b\bar{b}) \tag{2.5}$$

and the singlet state by

$$\sqrt{\frac{1}{3}}(r\bar{r} + g\bar{g} + b\bar{b}) \tag{2.6}$$

The singlet state, however, is symmetric in color space, meaning that it cannot be exchanged between particles with color charge. Therefore, the octet states define the eight types of gluons that mediate the strong force between particles.[33]

2.1.3 Quantum Chromodynamics

The interactions in the strong force are described by a theory known as Quantum Chromodynamics (QCD). It is an $SU(3)$ non-abelian gauge theory and has three properties that define most of its results. Those properties are color confinement, as defined in the previous section, asymptotic freedom, and chiral symmetry breaking.

Chiral symmetry refers to an approximate symmetry of the QCD Lagrangian under transformations between left-handed and right-handed quarks. In an idealized situation where quarks are massless, this symmetry would be exact. However, in the real world, quarks have small but nonzero masses, and chiral symmetry is spontaneously broken.

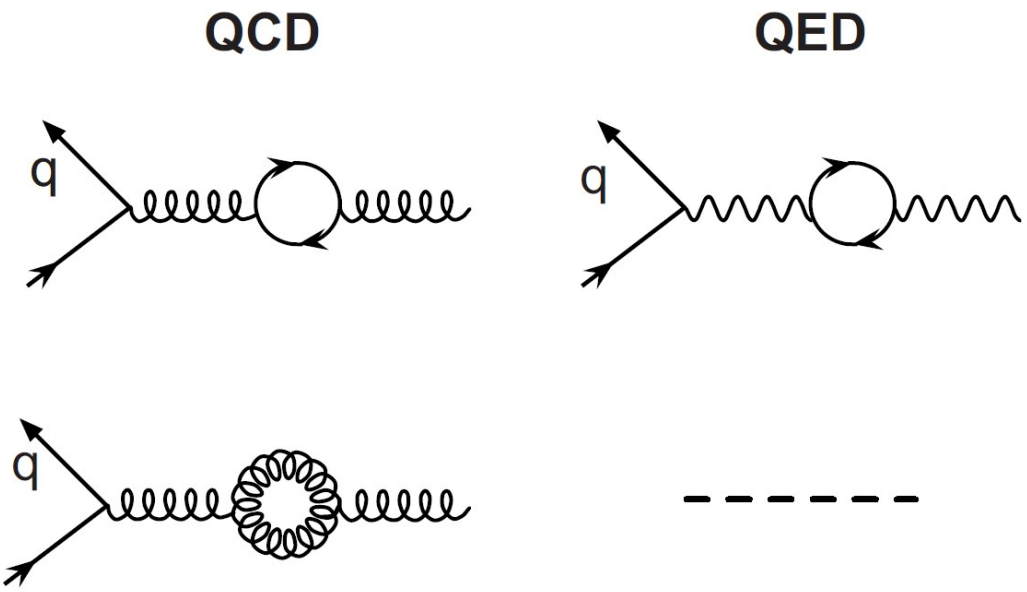


FIGURE 2.3: First-order Feynman loops in QCD and QED. In QCD, quarks may form loops of particles and anti-particles, as well as two gluons. In QED, the photon may only form particle-antiparticle loops.

This spontaneous breaking occurs because the vacuum state of QCD, the lowest energy state, does not respect chiral symmetry. In a chiral-symmetric vacuum, the quark condensate—an expectation value of the quark bilinear—would be zero. However, in reality, this condensate is nonzero, indicating that the chiral symmetry is not preserved. This nonzero quark condensate gives rise to the masses of hadrons, which are not solely due to the masses of the constituent quarks but rather arise from the dynamics of chiral symmetry breaking.

Asymptotic freedom, as a property of QCD, means that the strong force interaction becomes asymptotically weaker as the energy scale increases and the length scale decreases. Conversely, this means that at low energy scales and long distances, the coupling between quarks becomes infinitely strong, meaning that quarks cannot be found in isolation.

The asymptotic freedom of QCD is the direct result of the color charge of gluons. In quantum field theories, there exists a phenomenon known as screening, which results from the polarization of the vacuum around a charge. Examples of this in QED and QCD are shown in figure 2.3.

In the case of QED, a photon may form a virtual particle-antiparticle pair, with the opposite charge virtual particle attracted to the original particle, and the like

charge repelled. This has the effect that the larger the energy scale and the smaller the distance, the greater the coupling constant of the interaction.

Gluons, however, throw a wrench into this picture, as they are able to couple with themselves. This leads to the production of virtual gluon-gluon pairs, which results in anti-screening. This works in opposition to screening and gives a first-order QCD coupling constant of

$$\alpha_s(Q^2) = \frac{12\pi}{(33 - 2n_f) * ln(Q^2/\Lambda^2_{QCD})} \tag{2.7}$$

where Q^2 is the energy scale, Λ_{QCD} is a free constant known as the QCD scale, and n_f is the number of quark flavors. From this, we can draw two conclusions:

First, the sign of this coupling constant is determined by the number of quark flavors present in the theory. For a theory with greater than 16 types of quarks, there is a sign change. In our universe, however, there appear to be six distinct types of quarks.

Second, this coupling constant has an asymptote at $Q = \Lambda_{QCD}$. This means that at energies approaching the QCD scale, perturbation theory becomes less and less useful in calculating predictions of QCD.

2.1.4 Cross-Sections

Cross-sections are perhaps the most important quantity in particle physics, as they are what link the theoretical concepts of particle physics with the measurable realities of experiments. A cross-section is the probability of a specific interaction occurring in a collision between two particles. It is typically measured as an area in units of barns (100 fm^2), which is roughly the cross-sectional area of a Uranium nucleus. Although it is measured as an area, it cannot be simply given as the area of a target particle, as the likelihood of a specific interaction also contributes to the cross-section. The name "barn" was chosen during the Manhattan Project to refer to a large cross-sectional area, partly in hopes of it obscuring its connection to nuclear structure.[34]

Cross-sections serve as a window into the structure of nucleons, shedding light on their inner composition and the interactions between their constituent quarks and gluons. When particles such as protons and neutrons are bombarded with other particles, the resulting cross-sections provide insights into the spatial and kinematic distributions of quarks within these nucleons.

Particles like nucleons can act either as unified entities or as a grouping of particles. This behavior depends on the specific interaction and context involved. For instance, when a proton is struck by a low-energy electron, the electron treats the proton as a single, compact particle. However, if the electron carries significant energy, it becomes capable of distinguishing the individual quarks within the proton and exploring how they are distributed.[35]

2.1.5 Differential Cross-Sections

Differential cross-sections provide more details of particle interactions by capturing how the cross-section changes with respect to specific kinematic variables. Unlike total cross-sections, which provide a global measure of interaction probability, differential cross-sections measure how the interaction probability changes as a function of particular observables such as scattering angle, energy transfer, or momentum transfer.

One class of differential cross-sections are expressed as functions of solid scattering angles. It is represented as a differential of the total cross-section over an infinitesimal angle: $d\sigma/d\Omega$, where $d\Omega = \sin\theta d\theta d\phi$.

The angular distribution of scattered particles can reveal information about the scattering process and the forces involved. For instance, in elastic scattering experiments, where incoming particles are deflected without any internal changes, the angular distribution of the scattered particles can provide insights into the spatial distributions of charges within the interacting particles.

Differential cross-sections for variables other than angles are also commonly used, as they can reveal other aspects of the interaction. For instance, the invariant mass-dependent cross-section or target and beam x-dependent cross-sections might be given by $d\sigma/dM$ or $d^2\sigma/dx_1dx_2$. More complex interactions, such as inelastic scattering, involve energy transfers between particles, and studying the differential

cross-sections as a function of energy transfer helps reveal the dynamics of these interactions.[36]

2.2 Nucleon Structure

In parton theory's simplest form, protons and neutrons are made of three quarks: two up quarks and a down quark for the proton; two down quarks and an up quark for the neutron. This, it turns out, is an oversimplification, but serves as a good starting point for discussion of nucleon structure. In QCD, the proton and neutron systems are complex, involving not only the three valence quarks but also the force mediating gluons and virtual quark-antiquark pairs, the sea quarks. For this discussion, we will focus on the quarks and antiquarks.

All of the possible information about quarks' position and momentum is encoded by the quark-quark correlator. From the correlation function, it is possible to derive the Wigner distribution of each of the partons, giving a six-dimensional quasi-distribution over the position 3-space and the momentum 3-space, $W(\vec{x},\vec{k})$. The Wigner distributions, however, are not particularly useful to experimentalists, as they are inaccessible via direct measurements. For that reason, it is useful to integrate or take the limit of the quark-quark correlator to find observable quantities.[37]

The quark-quark correlator can be written as follows in the symmetric infinite momentum frame, from Lorce et al [2]:

$$\tilde{W}^{[\Gamma]}_{\Lambda'\Lambda}(P,k,\Delta,N;\eta) = \int \frac{d^4z}{(2\pi)^4} e^{ik\cdot z} \left\langle p',\Lambda'\right| \bar{\psi}(-\frac{z}{2})\Gamma\mathcal{W}\psi(\frac{z}{2}) \left|p,\Lambda\right\rangle . \tag{2.8}$$

In this formalism, Λ and Λ' are the initial and final helicities of the hadron in the light-cone reference frame. $P = (p'+p)/2$ is the average hadron four-momentum, k is the quark four-momentum, $\Delta = p' - p$ is the four-momentum transfer, and Γ is a member of the basis $\{\mathbb{1}, \gamma_5, \gamma^\mu, \gamma^\mu\gamma_5, i\sigma^{\mu\nu}\gamma_5\}$ in Dirac space.

$\mathcal{W} \equiv \mathcal{W}(-\frac{z}{2},\frac{z}{2}|n)$ makes sure that the correlator is color gauge invariant, and connects

$$-\frac{z}{2} \to -\frac{z}{2} + \infty\cdot n \to \frac{z}{2} + \infty\cdot n \to \frac{z}{2}, \tag{2.9}$$

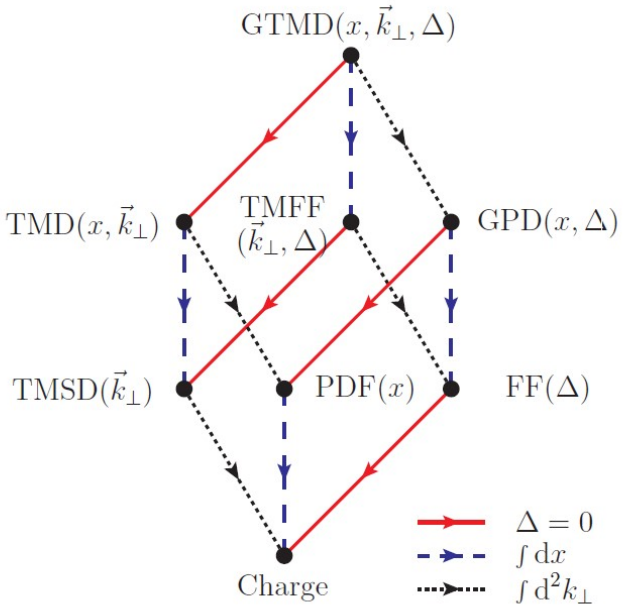


FIGURE 2.4: The projections of the Generalied Transverse Momentum Dependent distributions (GTMDs) onto other representations of parton distributions and form factors. Solid red arrows represent the forward momentum limit of the hadron, dotted black arrows correspond to integrating over the transverse momentum of the quark and dashed blue lines represent integrating over the quark's longitudinal momentum. Diagram taken from Lorce et al.[2]

which means that the Wilson line is dependent on the light cone direction, n, which is a member of the set $[0, \pm 1, \vec{0}_\perp]$. Because the magnitude of n is irrelevant (it is multiplied by infinity), the correlator instead depends on the value

$$N = \frac{M^2 n}{P \cdot n} \tag{2.10}$$

where M is the hadron mass. We can introduce η =sign(n^0), which can equal ± 1 and indicates whether the Wilson line is future- or past-pointing.

Transverse momentum-dependent parton distributions, parton distribution functions, and form factors all correspond to limits or projections of equation 2.8. They all have in common that they are taken at light-cone time of $z^+ = 0$, so we can integrate equation 2.8 over k^-.

$$W^{[\Gamma]}_{\Lambda'\Lambda}(P,x,\vec{k}_\perp,\Delta,N;\eta) = \int dk^- \tilde{W}^{[\Gamma]}_{\Lambda'\Lambda}(P,k,\Delta,N,\eta) \tag{2.11}$$

$$= \frac{1}{2}\int \frac{dz^- d^2 z_\perp}{(2\pi)^3} e^{ixP^+z^- - i\vec{k}_\perp\cdot\vec{z}_\perp} \langle p',\Lambda'|\,\bar{\psi}(-\frac{z}{2})\Gamma\mathcal{W}\psi(\frac{z}{2})\,|p,\Lambda\rangle \Big|_{z^+=0} \tag{2.12}$$

where we are using $a^\mu = [a^+, a^-, \vec{a}_\perp]$ where $a^\pm = (a^0 \pm a^3)/\sqrt{2}$ and $\vec{a}_\perp = (a^1, a^2)$. $x = k^+/P^+$, which is the longitudinal momentum fraction of the quark (also known as the Bjorken-x), and $\vec{k}_\perp$ is the transverse momentum of the quark.

There are three "directions" that one can go from the GTMDs to arrive at observables, as shown in figure 2.4. One can take the forward momentum limit of the hadron, or integrate over either the transverse or longitudinal momentum of the parton. In the sections below, we will follow each of those steps to show the derivations of three important observables: the transverse momentum-dependent distributions, the parton distribution functions, and charges.[2]

2.2.1 Transverse Momentum Distributions

Transverse Momentum Distributions (TMDs) are useful probes into the internal dynamics of nucleons. As their name would suggest, they measure the distribution of the parton momenta that are transverse to the momentum transfer in an interaction. The Quark TMDs can be defined via the quark-quark correlator, by taking the $\Delta = 0$ limit of equation 2.12.

$$\Phi^q(P,x,\vec{k_T},N;\eta) = W^{[\Gamma]}_{\Lambda'\Lambda}(P,x,\vec{k}_\perp,0,N,\eta) \tag{2.13}$$

$$= \frac{1}{2}\int \frac{dz^- d^2 z_\perp}{(2\pi)^3} e^{ixP^+z^- - i\vec{k}_\perp\cdot\vec{z}_\perp} \langle p',\Lambda'|\,\bar{\psi}(-\frac{z}{2})\Gamma\mathcal{W}\psi(\frac{z}{2})\,|p,\Lambda\rangle \Big|_{z^+=0} \tag{2.14}$$

$$= \int \frac{dz^- d^2 \vec{z}_\perp}{(2\pi)^2} e^{iP^+z^- - i\vec{k}_\perp\cdot\vec{z}_\perp} \langle P,S|\,\bar{\psi}^q(z)\,|P,S\rangle\,, \tag{2.15}$$

$$= \Phi^{q[\gamma^+]}\frac{\gamma^-}{2} + \Phi^{q[\gamma^+\gamma^5]}\frac{\gamma^-\gamma^5}{2} + \Phi^{q[i\sigma^{\alpha}\gamma^5]}\frac{i\sigma^{\alpha}\gamma^5}{2} \tag{2.16}$$

where α is a transverse index, and $\Phi^{q[\Gamma]} = \frac{1}{2}Tr[\Phi^q \Lambda]$. This allows us to write the eight TMDs as follows:

$$\Phi^{q[\gamma^+]} = f_1^q(x, \vec{k_T}^2) - \frac{\epsilon + T^{ij} k_T^i S_T^j}{M} f_{1T}^{\perp q}(x, \bar{k_T}^2), \tag{2.17}$$

$$\Phi^{q[\gamma^+\gamma^5]} = S_L g_{1L}^q(x\vec{k_T}^2) + \frac{\vec{k_T} \cdot \vec{S_T}}{M} g_{1T}^q(x, \vec{k_T}^2), \tag{2.18}$$

$$\Phi^{q[i\sigma^\alpha\gamma^5]} = S_T^\alpha h_1^q(x, \vec{k_T}^2) + \frac{k_T^\alpha(\vec{k_T} \cdot \vec{S_T}) - \frac{1}{2}\vec{k_T}^2 S_T^\alpha}{M^2} h_{1T}^{\perp q}(x, \vec{k_T}^2) \tag{2.19}$$

$$+ S_L \frac{k_T^\alpha}{M} h_{1L}^{\perp q}(x, \vec{k_T}^2) + \frac{\epsilon + T^{ij} k_T^i S_T^j}{M} h_1^{\perp q}(x, \vec{k_T}^2) \tag{2.20}$$

where $S_{L,T}$ are the nucleon longitudinal and transverse polarization vectors. These terms are shown in terms of their quark/nucleon polarization relation in figure 2.5. The simplest of these, $f_1(x, \vec{k_T}^2)$, describes the probability of finding a parton with a given longitudinal momentum fraction x and a transverse momentum of k_T. It is directly related to the more general Parton Distribution Function $f_1(x)$ by integrating over all possible transverse momenta, $f_1(x) = \int f_1(x, \vec{k_T}^2) dk_T^2$.[38]

Two other TMDs: the Boer-Mulders function, which describes "the net polarization of quarks inside an unpolarized proton"[4], and the Sivers function, which describes "the correlation between the spin of the proton and the orbital motion of its constituents"[39], merit more discussion for the purposes of this dissertation.

2.2.2 Boer Mulders Function

Denoted as $h_1^\perp$ in Figure 2.5, the Boer-Mulders function was formally defined in a 1997 paper by D. Boer and P.J. Mulders.[40] It quantifies a spin-orbit correlation, and a non-zero value indicates that the nucleon has a certain handedness, $(P \cdot (k_T \times S_T))$, and is time-reversal odd, meaning it changes sign when time is reversed. This means that the values found using different processes in experiment will have equal magnitude but no consistent sign. As Boer explains,

> It turns out that the quarks can be polarized on average even inside ... an unpolarized proton, as long as they are not moving exactly along the proton direction. If the proton moves along the z direction say,

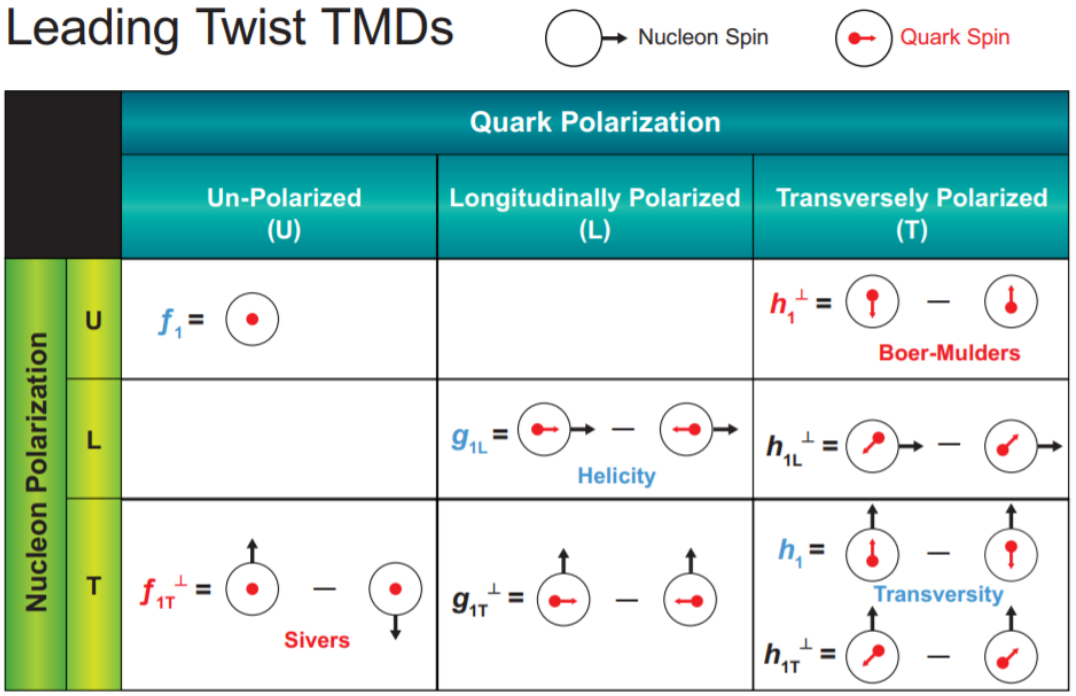


FIGURE 2.5: The leading Twist Transverse Momentum Dependent Parton Distribution Functions from Acardi et al.[3]

> then the quarks can have some transverse momentum k_T with respect to the proton momentum, which together define a plane. The quarks can then have a net polarization orthogonal (or transverse) to that plane. A nonzero Boer-Mulders function means that there is such a net quark polarization.[4]

Lattice QCD, a non-perturbative way to solve QCD, predicts a non-zero Boer-Mulders function.[41] It is also suggested by Boer to be an explanation for an asymmetry in unpolarized Drell-Yan scattering.[9] Since the SeaQuest experiment measured unpolarized Drell-Yan scattering, it is possible to use that data to measure the angular dependence of the process and thereby access the Boer-Mulders function.

2.2.3 Sivers Function

The Sivers Function, $f_{1T}^{\perp}$, "describes the distribution of unpolarized quarks inside a transversely polarized nucleon, through a correlation between the quark transverse momentum $\vec{p_T}$ and the nucleon transverse spin S_T."[38] Like the Boer-Mulders

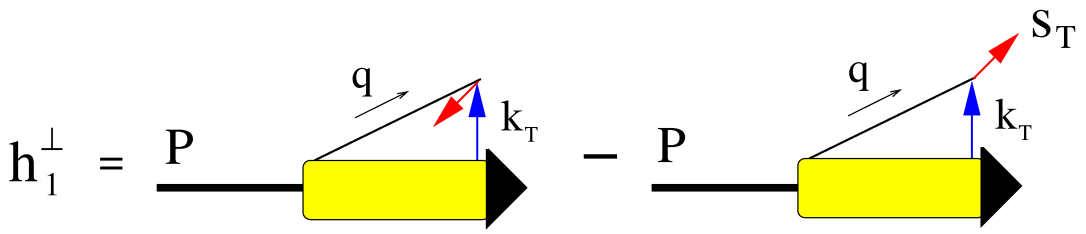


FIGURE 2.6: Illustration of quark polarization in an unpolarized proton: If the proton moves along the z direction, the quarks can possess transverse momentum k_T relative to the proton's momentum. This creates a plane, and the quarks can exhibit a net polarization S_T that is transverse to this plane, as described by the Boer-Mulders function.[4].

function, it is time-reversal odd. Probably the most studied TMD, it can give insight into the orbital angular momentum of partons within the nucleon, which is thought to be an important component of the overall spin of the nucleon, as shown in figure 2.8.

Because it is time-reversal odd and believed to be process-dependent, the Sivers functions measured via Semi-Inclusive Deep Elastic Scattering and the Drell-Yan process have an equal magnitude but opposite sign. It can be measured via the angular distribution of Drell-Yan scattering in a polarized process, and its measurement is the main goal of the SpinQuest experiment.

2.2.4 Parton Distribution Functions

Parton distribution functions (PDFs) measure the distribution of partons inside the nucleon as a function of the longitudinal momentum fraction x, and the Q^2 of the interaction, $f(x, Q^2)$. They can be obtained from the correlator Φ, as defined in equation 2.16 by integrating over the transverse momentum $\vec{k}_\perp$ as follows:

$$\mathcal{F}^q(P, x, N) = \int d^2\vec{k}_\perp \Phi^q(P, x, \vec{k_T}, N; \eta) \tag{2.21}$$

$$= \frac{1}{2}\int \frac{dz^-}{2\pi} e^{ixP^+z^-} \langle p', \Lambda' | \bar{\psi}(-\frac{z}{2})\Gamma\mathcal{W}\psi(\frac{z}{2}) | p, \Lambda\rangle \Big|_{z^+=z_\perp=0}. \tag{2.22}$$

This is parameterized as

$$\mathcal{F} = \begin{pmatrix} f_1 & 0 & 0 & 0 \\ 0 & h1 & 0 & 0 \\ 0 & 0 & h1 & 0 \\ 0 & 0 & 0 & g1 \end{pmatrix}, \tag{2.23}$$

where f_1, h_1, and g_1 are the unpolarized parton distribution function, the transversity, and the helicity, respectively. These have very simple relations to specific TMDs, as follows:[2]

$$f_1(x) = \int d^2k_\perp f_1(x, \vec{k}_\perp^2), \tag{2.24}$$

$$g_1(x) = \int d^2k_\perp g_{1L}(x, \vec{k}_\perp^2), \tag{2.25}$$

$$h_1(x) = \int d^2k_\perp h_1(x, \vec{k}_\perp^2). \tag{2.26}$$

Next-to-leading-order PDFs are especially useful for experimentalists in Monte Carlo simulations of physics processes.[42] Most cross-sections are dependent on the parton distribution at leading order, meaning that a good estimate of the PDFs can allow for good predictions of yields in scattering experiments. To that end, there are several collaborations that use the results of scattering experiments to calculate or predict the parton distributions for the different types of partons in the nucleon.[5] An example of one of these is shown in figure 2.7.

2.2.5 Charges

The third and final "direction" in figure 2.4 is to integrate over x. By doing that to the PDFs, we can get the charge:

$$\mathcal{Q}(P) = \int dx \mathcal{F}(p, x, N) \tag{2.27}$$

$$= \frac{1}{2P^+} \langle p', \Lambda' | \, \bar{\psi}(-\frac{z}{2}) \Gamma \mathcal{W} \psi(\frac{z}{2}) \, | p, \Lambda \rangle \,, \tag{2.28}$$

which can be parameterized as:

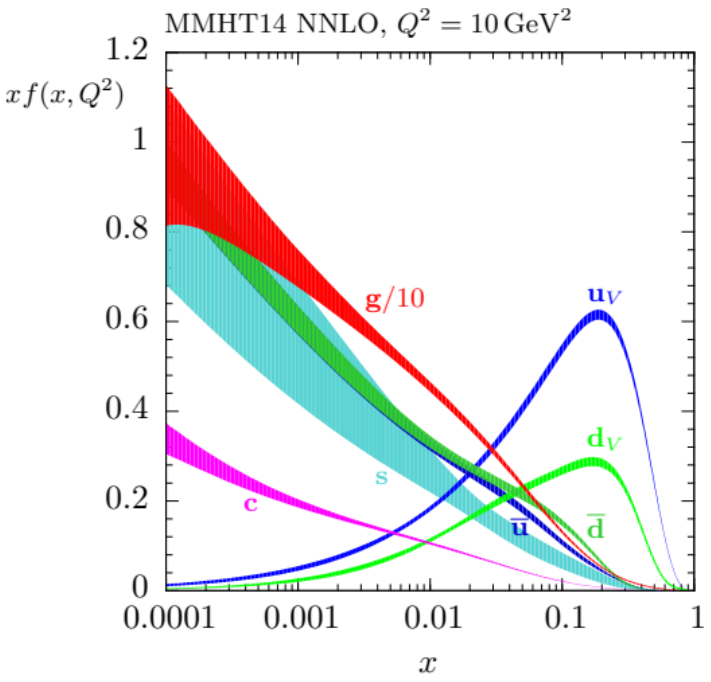


FIGURE 2.7: The Next-to-next-to-leading order MMHT 2014 PDFs for up, down, charm, and strange quarks, as well as gluons at $Q^2 = 10$GeV. Curves are derived by fitting QCD to global hard-scattering data. Uncertainties shown are one standard deviation[5].

$$\mathcal{Q} = \begin{pmatrix} q & 0 & 0 & 0 \\ 0 & \delta q & 0 & 0 \\ 0 & 0 & \delta q & 0 \\ 0 & 0 & 0 & \Delta q \end{pmatrix}. \tag{2.29}$$

In this, q is the vector charge, which is the number of quarks with flavor q. Δq and δq are the axial charge and tensor charges, which represent the fraction of the helicity and transversity carried by quarks of flavor q.[2]

2.2.6 The Spin Crisis

[This section was excerpted from the author's master's thesis [43]]

In the original parton model, the spin of the nucleon was carried entirely by the valence quarks. The stability of nucleons' measured spin seemed to confirm this, as chaotic internal dynamics would seem to add a degree of randomness in the observable. Problems began to arise in the late 1980s when the European

Muon Collaboration measured the valence quark contribution to the spin of the proton. They found that the intrinsic spin of the valence quarks contributed a very small amount to the entire spin. Their analysis found that the spin of the quarks contributed $(14 \pm 9 \pm 21)\%$ of the spin of the proton, which is consistent with zero.[44]

More recent studies have found that although the spin of the valence quarks does contribute to the overall spin of the nucleon, the intrinsic spin of the constituent quarks and gluons is not sufficient to account for the entire spin. For that reason, it is now believed that the orbital angular momentum of the partons also contributes to the spin of the nucleon. Recent studies using lattice quantum chromodynamics (LQCD), a non-perturbative method to solve QCD, estimate that 60% of the spin comes from the intrinsic spins of quarks and gluons, and the remaining 40% comes from the orbital angular momentum of the quarks, as shown in Figure 2.8.

LQCD evaluates QCD by dividing space-time into discrete points, with connecting links between neighboring points. The fermion fields are defined at each point of the lattice, and the gauge fields are defined on the links between the points. This allows QCD to be solved without any assumptions, letting theorists calculate observables from first principles. As the spacing between the points approaches zero, the limit of continuous QCD is recovered.

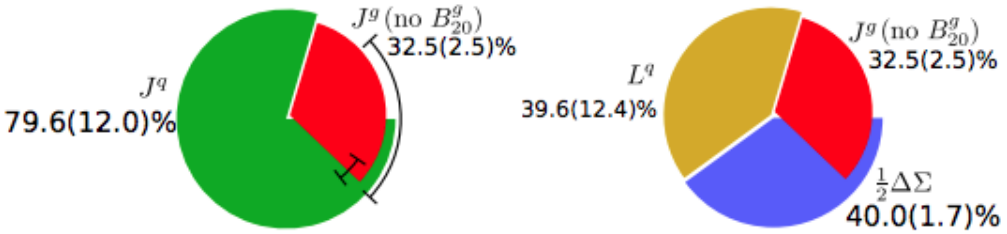


FIGURE 2.8: Spin breakdown of a proton (Wiese et al.)[6]. Approximately 80% of the spin is contributed by the quarks, of which half is from the orbital angular momentum of the quarks.

According to a calculation by Wieste et al.,[6] the majority of the contribution from orbital angular momentum comes from the antiquarks in the nucleon sea, with the valence quarks accounting for a fairly small portion of the spin. This has led a large focus of spin physics research to be the dynamics of the sea quarks.

2.3 Scattering Processes

Many processes can be used in nuclear physics experiments to probe the internal dynamics of nucleons. In this section, we will discuss three of them: Deep inelastic scattering, Drell-Yan scattering, and quarkonium production. Drell-Yan scattering and quarkonium production, specifically J/ψ production, play major roles in the SeaQuest and SpinQuest experiments, as they are important hadron-hadron processes. Deep inelastic scattering does not play a role in these experiments, but discussing it provides insight into other experiments that measure nucleon structure, as well as deepening understanding of Drell-Yan scattering, as they are closely related processes.

2.3.1 Deep Inelastic Scattering

When an electron and a proton interact, a virtual photon is exchanged between the two, which changes the trajectory of the electron and the proton. Electron-proton scattering can be categorized into four regimes based on the wavelength of the virtual photon, each of which provides a different probe of the proton.

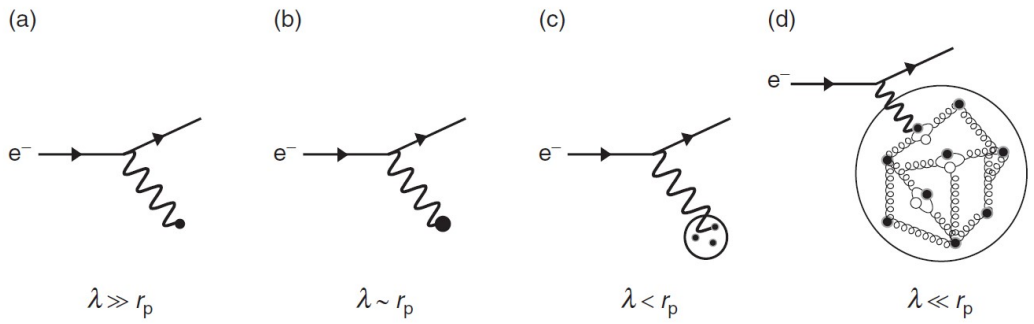


FIGURE 2.9: The four regimes of electron scattering from Thomson.[7]

Those regimes are shown in figure 2.9 and described below.

a. $\lambda \gg r_p$. In this regime, the proton is treated as a point-like charge and the scattering can be treated as electromagnetic scattering off of a static potential, known as elastic scattering.

b. $\lambda \approx r_p$. At this energy, the proton cannot be treated as point-like, and its charge and magnetic moment distributions need to be taken into account.

c. $\lambda < r_p$. When the virtual photon wavelength decreases, inelastic scattering begins to dominate over elastic scattering. In inelastic scattering, the virtual photon is exchanged with a specific quark, allowing for probing of the internal nucleon structure, mostly of the valence quarks.

d. $\lambda \ll r_p$. At high electron energies, the wavelength of the virtual photon starts to be small enough to see the complex interior nature of the proton, including the sea quarks and gluons. At this energy, the proton is able to break up, meaning that there are several hadrons in the final state.

Elastic scattering is well described by only the outgoing angle of the electron, as in Rutherford scattering:

$$\frac{d\sigma}{d\Omega} = \frac{\alpha^2}{16E_k^2 sin^4(\theta/2)}, \tag{2.30}$$

where $\alpha = e^2/4\pi$ and $E_k = p^2/2m_e$, e is the fundamental charge, p is the momentum of the incoming electron, and m_e is the mass of the electron.

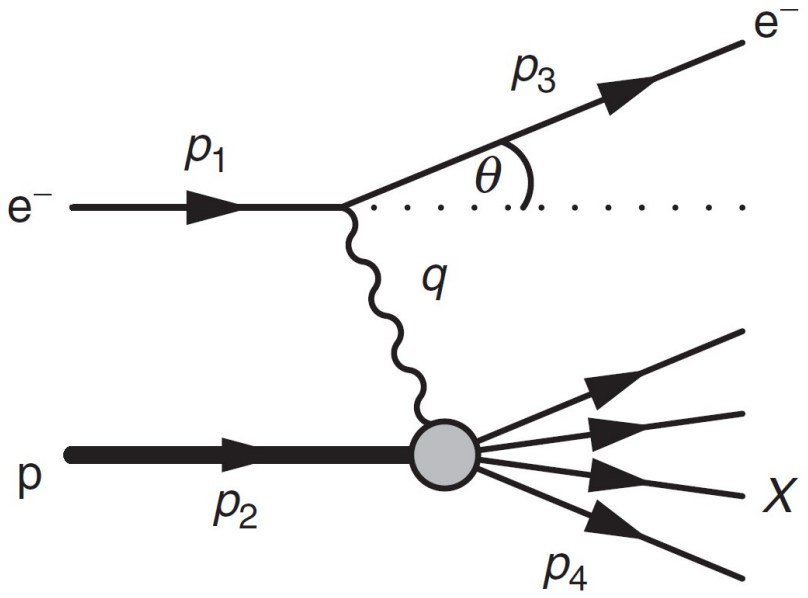


FIGURE 2.10: The Deep Inelastic Scattering process from Thompson. [7]

For inelastic scattering, however, it is necessary to use two kinematic variables. We have some options in choosing these variables, but it is useful to choose between W, x, y, v, and Q^2, which are, respectively, the invariant mass of the hadronic system, the Bjorken-x of the struck parton, the inelasticity, the energy loss, and the

negative square of the four-momentum of the virtual photon. For a given center-of-mass energy, you can describe the kinematics of the process fully by choosing two of the quantities, as they are related by the following:

$$Q^2 \equiv -q^2, \qquad x \equiv \frac{Q^2}{2p_2 \cdot q}, \qquad y \equiv \frac{p_2 \cdot q}{p_2 \cdot p_1}, \tag{2.31}$$

$$v \equiv \frac{p_2 \cdot q}{m_p}, \qquad W \equiv p_4 \tag{2.32}$$

Where $p_{1,2,3,4}$ are the four-momenta of the incoming and outgoing particles, and q is the four-momentum of the virtual photon, as shown in figure 2.10.[7]

Deep inelastic scattering has been used to probe the interior dynamics of nucleons for over 50 years. In 1968, it was the first direct confirmation of the quark model of protons, when DIS at the Stanford Linear Accelerator Center observed resonances at three low momentum transfer values.[45]

In modern experiments, it is more common for experiments to measure semi-inclusive deep inelastic scattering (SIDIS). SIDIS and DIS are the same process, but to capture more information about the interaction, SIDIS measures not just the outgoing electron, but also the highest momentum hadron that leaves the collision.

2.3.2 Drell-Yan Scattering

Drell-Yan Scattering is closely related to DIS by a rotation of the Feynman diagram. When two hadrons interact, it is possible for a quark from one and an antiquark from the other to annihilate, forming a virtual photon. This virtual photon produces a dilepton pair, as shown in the Feynman diagram in figure 2.11.[8][46]

Similarly to SIDIS, we can define the following quantities (as in [47]):

$$q = (q_0, \boldsymbol{q}_T, q_L) \qquad Q^2 = M^2 = (-q)^2 \qquad y = \frac{1}{2} ln \frac{q_0 + q_L}{q_0 - q_L} \tag{2.33}$$

$$x_F = \frac{2q_L}{\sqrt{s}} = x_1 - x_2 \qquad s = (E_1 + E_2)^2 \qquad s x_1 x_2 = M, \tag{2.34}$$

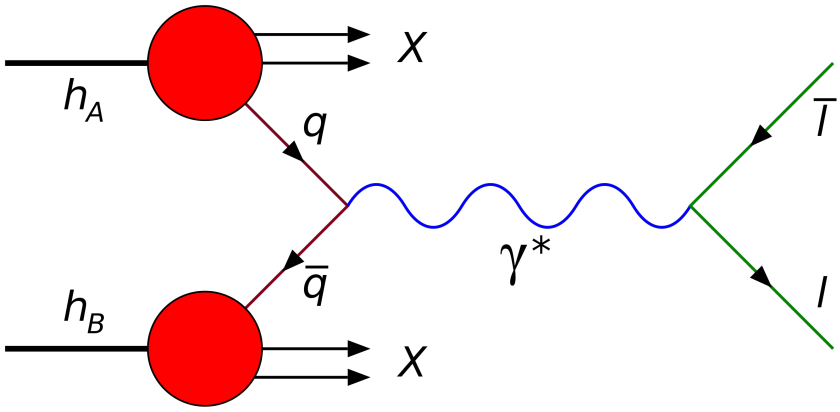


FIGURE 2.11: The Drell-Yan Scattering process.[8]

where q is the four-momentum, q_0 is the rest mass, and $\boldsymbol{q}_{T,L}$ are the transverse/-longitudinal momenta of the Drell-Yan virtual photon, respectively. Q^2 is the square of the sum of the invariant masses of the leptons, and s is the square of the center-of-mass energy. x_F is the Fermi-x, and $x_{1/2}$ are the Bjorken-x for the beam and target, respectively. These can be calculated from the x_F, s and M via

$$x_{1,2} = \frac{1}{2}(\sqrt{x_F^2 + \frac{4M^2}{s}} \mp x_F). \tag{2.35}$$

At leading order, the x-dependent Drell-Yan differential cross-section is only dependent on the PDFs of the quarks and anti-quarks, as follows:

$$\frac{d\sigma}{dx_1 dx_2} = \frac{4\pi\alpha^2}{9sx_1x_2}\sum_i e_i^2(q_i^1(x_1,Q^2)\bar{q}_i^2(x_2,Q^2) + \bar{q}_i^2(x_1,Q^2)q_i^1(x_2,Q^2)), \tag{2.36}$$

α_s is the fine structure constant, e is the charge of the quark, and the sum is over the different quark flavors.

Because of the x dependence of the PDFs for quarks and antiquarks, by selecting the x_1 and x_2 values that we are measuring, we can isolate the two terms. By measuring where $x_2 << x_1$, as is done in both SeaQuest and SpinQuest, the cross-section becomes approximately

$$\frac{d\sigma}{dx_1 dx_2} \approx \frac{4\pi\alpha^2}{9sx_1x_2}\sum_i e_i^2 q_i^B(x_1, Q^2)\bar{q}_i^T(x_2, Q^2). \tag{2.37}$$

Because a proton has two up quarks, and the up quark has an electric charge of $+2/3$ (as opposed to the $-1/3$ of down), proton-proton Drell-Yan scattering is dominated by up-antiup annihilation. Therefore, in order to access the down-antidown annihilation, one must first perform proton-proton scattering, and use that information to extract the information from proton-deuteron scattering.

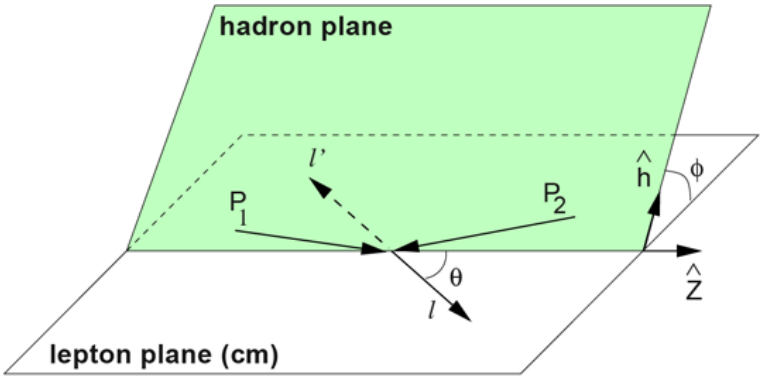


FIGURE 2.12: The Drell-Yan process in the Collins-Soper frame from [9].

In the Collins-Soper frame, as seen in figure 2.12, Drell-Yan scattering has an angular dependence given by:

$$\frac{1}{\sigma}\frac{d\sigma}{d\Omega} = \frac{3}{4\pi}\frac{1}{\lambda+3}(1 + \lambda cos^2(\theta) + \mu sin(2\theta)cos(\phi) + \frac{\nu}{2}sin^2(\theta)cos(2\phi)) \tag{2.38}$$

where θ is the polar angle and ϕ is the azimuthal angle of the virtual photon in the Collins-Soper frame.[48] λ, μ, and ν are not dependent on θ and ϕ, but are dependent on other kinematic variables.

At leading-order, $\lambda = 1$, while $\mu = \nu = 0$. At next-to-leading-order, however, we can have $\lambda \neq 1$ and $\mu \neq \nu \neq 0$. Another condition, however, is imposed, where $\lambda + 2\nu = 1$. This is known as the Lam-Tung Relation.[49][50] At higher orders this relation does not appear to hold, and the Lam-Tung breaking coefficient, $2\nu - (1 - \lambda)$, is a useful probe of parton transverse momentum.[51]

These angular dependencies, as well as the Lam-Tung Relation, have been measured in $p \to p$ and $p \to d$ before by E866 at Fermilab, albeit with a much higher

energy than SeaQuest (SeaQuest used a 120 GeV beam, E866 used an 800 GeV beam). Other experiments, such as CERN NA10 and Fermilab E-615, measured the angular dependence of Drell-Yan scattering with a pion beam. Because of the parton distribution functions of quarks and antiquarks in protons versus pions, proton beams are better at probing the sea quarks distributions, while pions are better at probing the valence quark distributions. A plot of the angular distributions for these three experiments are shown in Figure 2.13.

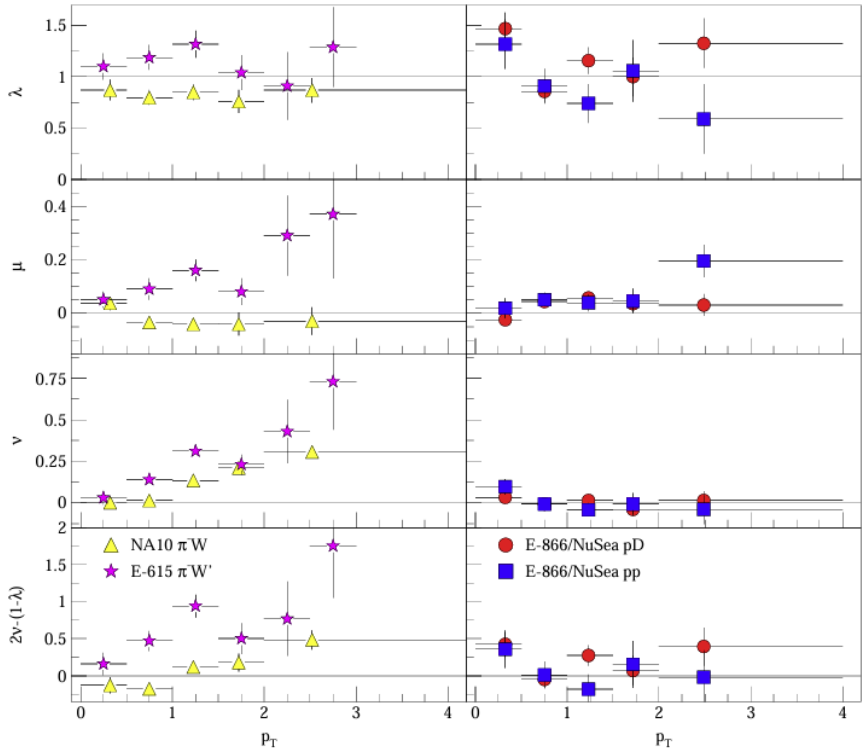


FIGURE 2.13: The p_T dependence of the Drell-Yan asymmetry variables from E866.[10][11]

As pointed out by Boer in [9], the presence of a non-zero ν can be explained by a non-zero Boer Mulders function. He calculated that the Boer-Mulders function, $h_1^\perp$ gives rise to a non-zero ν value:

$$\nu \propto \frac{h_1^\perp}{f_1}\frac{\bar{h}_1^\perp}{\bar{f}_1} \tag{2.39}$$

Because of this, by measuring the $cos(2\phi)$ dependence of Drell-Yan scattering, we are able to access the Boer-Mulders function. Both SeaQuest and SpinQuest were designed to measure the muons produced by Drell-Yan scattering, making it the most important process for the purpose of this dissertation.

As can be seen in Figure 2.13, there is a large discrepancy between the ν values in pion-induced Drell-Yan compared to proton-induced Drell-Yan. While the pion data shows a strong p_T dependence to ν, the proton data shows no such relation.

The absence of a p_T dependence in the proton-induced Drell-Yan data implies that the sea quark Boer-Mulders function is relatively small compared to the valence quark distributions. Additionally, the Lam-Tung Relation is clearly broken in pion-induced data, while it is consistent with zero in proton-induced data.

The discrepancies in the angular dependencies between pion-induced and proton-induced Drell-Yan scattering highlight differences in the Boer-Mulders functions for sea and valence quarks. These results highlight the importance of proton-induced experiments like SeaQuest and SpinQuest in providing additional data on sea quark distributions. By analyzing the angular dependencies, we gain deeper insights into parton dynamics and the internal structure of hadrons.

2.3.3 Quarkonium Production

Quarkonia are bound states of a heavy quark and its antiquark. They can be created in particle collisions via either quark-antiquark annihilation or gluon-gluon fusion. Because they consist only of a particle and its antiparticle, they are extremely short-lived, and their production is a large source of muon pairs in detector experiments.

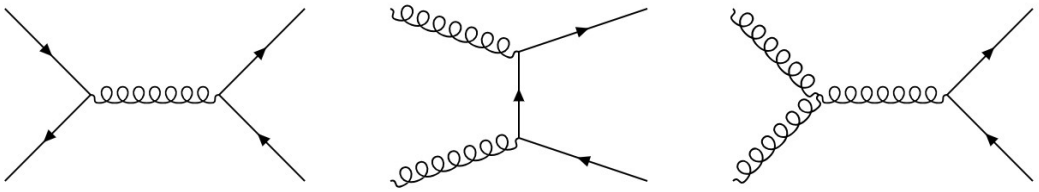


FIGURE 2.14: Feynman diagrams showing different first-order (no loop) pathways of quark-antiquark production in QCD. The left diagram shows quark-antiquark annihilation, resulting in a charmonium pair, and the middle and left diagrams show two versions of gluon-gluon fusion.

Because quarkonia have a well-defined mass, they exhibit spectroscopic behavior in the mass spectrum of an experiment's measured leptons. This can be both useful and a hindrance in analysis. The relatively higher differential cross-section of quarkonia production makes it difficult to measure Drell-Yan muons close to

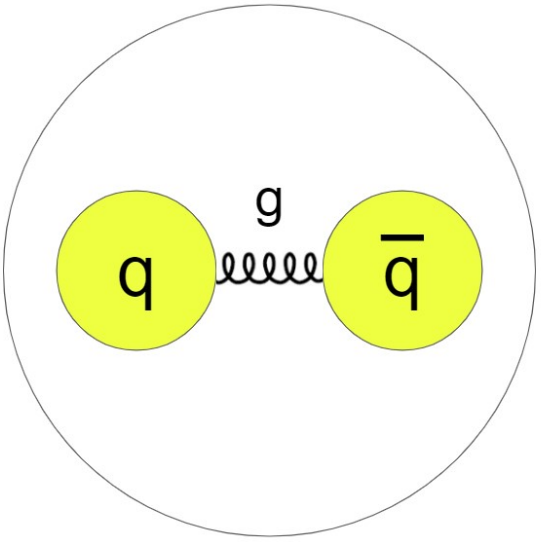


FIGURE 2.15: A quarkonium: a meson comprising of a heavy quark and its antiquark bonded by the strong force.

the masses of quarkonia. Because the muons produced by quarkonia decays have very specific masses, however, they do not appear in other mass ranges, allowing a simple cut of the data around the mass of quarkonia to isolate them from Drell-Yan data.

There are several quarkonia, most notably the J/ψ meson and the Υ meson, which are bound states of charm-anticharm and bottom-antibottom pairs, respectively. The SeaQuest/SpinQuest spectrometer is highly sensitive to the kinematic range at which the J/ψ is found, but not very sensitive to the Υ. As such, we will primarily discuss the J/ψ. Υ meson production is largely analogous to that of the J/ψ meson.

The J/ψ meson, a quarkonium comprised of a charm and anti-charm quark ($c\bar{c}$) was discovered in 1974 by researchers at both Brookhaven National Laboratory and the Stanford Linear Accelerator Center. Independent teams, led by Ting and Richter (at Brookhaven and SLAC, respectively), measured lepton production resonance at a mass of 3.1 GeV, with a narrow decay width (later shown to be only around 100 keV). The group at SLAC named the particle the ψ, while the Brookhaven named it the J particle, leading to its common name. It has a spin of 1 and a neutral electric charge. Since it is the lowest mass charmonium and spin 1, it is the most common charmonium. Excited states of the J/ψ have also been measured, most notably the ψ' meson, with a mass of around 3.7 GeV. [12][52][53]

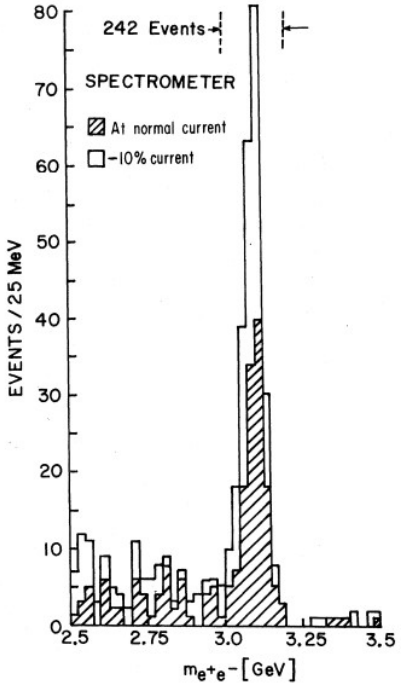


FIGURE 2.16: A plot showing the mass spectrum measured in the Brookhaven experiment showing the existence of a then-unknown particle, now known to be the J/ψ meson.[12]

J/ψ meson production, unlike the Drell-Yan process, is not only a quark-antiquark process in nucleons. Although it can be created by the annihilation of quarks and antiquarks, it is also able to be created via gluon-gluon fusion, as shown in Figure 2.14. The levels at which these different processes contribute to the overall cross-section of the process depend heavily on the parton distributions.

At higher energies, gluons carry more of the longitudinal momentum of nucleons, which means that at high energies the gluon-gluon fusion is expected to dominate J/ψ production, while at lower energies quark-antiquark annihilation is expected to be a large contribution to the overall process cross-section, dependent on x_F. At the SeaQuest/SpinQuest center-of-mass energy of $\sqrt{s} = 15.4$ GeV, it was calculated by Bhaduri et al. that at low x_F the gluon process dominates, while at high x_F, the quark annihilation dominates.[54]

Because of the kinematic range at which SeaQuest and SpinQuest operate, a large number of the dimuons detected originate from the decay of the J/ψ process. That means that although it is not the main focus of the experiment, understanding the process is very important for signal/background analysis. In addition, by reconstructing the J/ψ invariant mass peak, we are able to get a measure of the quality of our kinematic variable resolution.

Chapter 3

The SeaQuest and SpinQuest Experiments

The understanding of the proton's inner structure, its quark constituents, and their dynamics has been a central pursuit in the field of nuclear and particle physics. To this end, two experiments, SeaQuest and SpinQuest, were designed to explore the nucleons' compositions, shedding light on their quark distributions. Both of these experiments use the Fermilab main injector beam and employ the E906 spectrometer for data acquisition. However, they differ in their specific objectives, focusing on sea quark distribution and Sivers asymmetry in Drell-Yan reactions, respectively. This chapter will discuss the motivations behind each experiment, outline their distinctive characteristics, and provide details of the experimental setup.

3.1 Motivations

As discussed in Chapter 2, nucleons consists of a complex interplay of quarks, anti-quarks, and gluons, collectively shaping their properties. Among the constituents, the behavior of quarks, particularly the sea quarks, plays a pivotal role in defining their characteristics. Sea quarks are constantly emerging and annihilating in the proton, influencing its internal structure. The SeaQuest experiment was designed to explore the flavor dependence of sea quarks, particularly the ratio of anti-down to anti-up quarks in the nucleons, and to investigate any potential modifications of the sea quark structure within heavier nuclei.

The SeaQuest experiment (E906) sought to measure the distributions of sea quarks in the nucleon and explore the anti-quark EMC effect. It also provided data with which to study Boer-Mulders function by analyzing the azimuthal distributions of muons resulting from Drell-Yan scattering.

The SIDIS experiments at various facilities have provided insights into the Sivers function. However, Drell-Yan scattering can help to provide more insight, as it is a cleaner probe of the Sivers function than SIDIS.

The SpinQuest experiment (E1039) seeks to address questions about the Sivers function by using the E906 spectrometer and a newly developed transversely polarized proton target with high polarization. This experiment aims to provide more precise measurements of the Sivers asymmetry of sea quarks.

3.2 Fermilab

The Fermi National Accelerator Laboratory, or Fermilab, is a particle physics research facility run by the US Department of Energy. It was founded in 1967 and has been a leader in particle physics research over the past half-century. It is primarily focused on probing the nature of energy and matter through the study of high-energy particles.[55]

Over its history, Fermilab has hosted numerous experiments that use particle accelerators to study high-energy interactions. Groups working at Fermilab have discovered several important discoveries, such as discovering the Bottom Quark (1977) the Top Quark (1995), and the Tau Neutrino (2000). These three particles were the last fermions to be discovered and completed all three generations of leptons and quarks. In addition, Fermilab has conducted experiments searching for particles that could explain dark matter and the matter/antimatter asymmetry observed in the universe.

3.3 The Main Injector Beam

The Main Injector Beam (MIB) at Fermilab is a proton accelerator with a circumference of approximately 3.9 kilometers. The MIB can accelerate protons to energies up to 120 GeV, making it an important tool for various physics experiments.

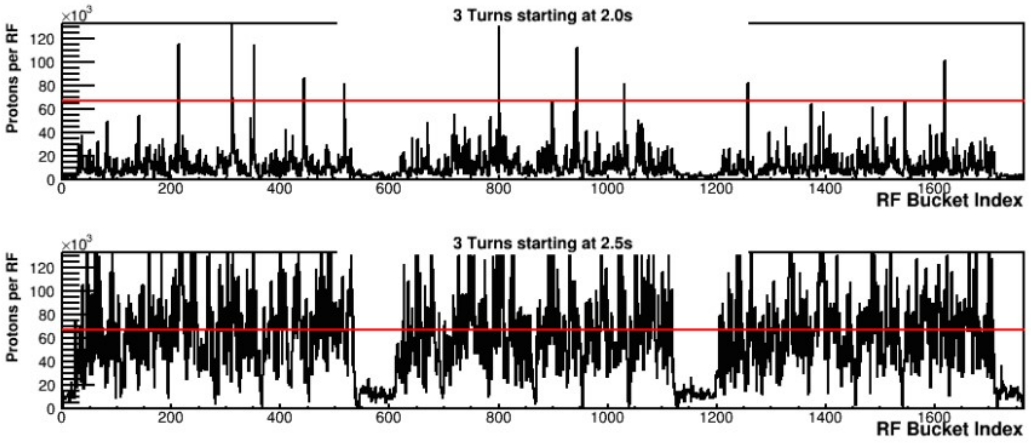


FIGURE 3.1: The intensity of the beam delivered to the SeaQuest experiment, as measured by the Beam Intensity Monitor. The red line represents the trigger cutoff threshold. If the threshold was reached, the trigger was inhibited for the previous 9 and next 9 RF buckets. These two strips are approximately 33 μs plots taken from the same spill within half a second of each other, showing that the intensity varied dramatically, even within the same spill.[13]

This is accomplished by subjecting the particles to a series of radiofrequency cavities, which provide the necessary electromagnetic fields to increase the particles' energy.

The MIB is used in a wide array of experiments conducted at Fermilab. It can provide beams for experiments in diverse fields of high-energy physics, including neutrino physics, dark matter research, and precision measurements of particle properties, such as SeaQuest and SpinQuest.

For SeaQuest and SpinQuest, the MIB delivers 120 GeV protons to the NM3 hall. The protons are delivered in approximately 4-second spills. The MIB has a frequency of 54.1 MHz, which divides the beam into "RF Buckets" that occur every 18.8 ns and are approximately 2 ns long. Ideally, each of these buckets has an equal, predictable number of protons. During the SeaQuest experiment, however, the number of protons in each bucket varied significantly, as shown in figure 3.1. Each spill therefore has approximately 200 million buckets in a pulse structure.

The SeaQuest and SpinQuest experiments use a trigger that is designed to detect when two oppositely charged high-mass muons pass through the detector array. Although the trigger is able to differentiate between muons from interactions from different RF buckets, it is not able to differentiate between interactions in the

same bucket. Therefore, when the intensity of a single bucket has too many protons, the trigger system can be saturated with false-positive dimuon pairs, which leads to missing true dimuon pairs. To combat this, the SeaQuest experiment designed a Beam Intensity Monitor that inhibited the event trigger when the beam intensity exceeded a certain threshold per bin (around 65,000-90,000 protons per bucket).[13]

3.4 The SeaQuest/SpinQuest Detector

The SeaQuest/SpinQuest spectrometer at Fermilab was constructed with the purpose of detecting oppositely charged pairs of muons, or dimuons. These dimuons are generated through the interactions between the MIB and the target (discussed in sections 3.5 and 3.6). It was specifically designed to explore the antiquark distributions within the nucleons of the target via Drell-Yan scattering.

The spectrometer's configuration comprises several key components, including two dipole magnets and four detector stations. The initial magnet positioned upstream is a closed-aperture solid iron magnet, which serves a dual role as both a magnet and a beam dump. In contrast, the second magnet is designed as an open-aperture magnet.

Each of the four detector stations within the spectrometer is equipped with scintillator hodoscopes and either proportional tubes or drift chambers. To efficiently trigger the data acquisition process, the spectrometer employs a Field Programmable Gate Array (FPGA) based trigger system. This trigger system analyzes the signals from the hodoscopes and compares them to a predetermined set of roadmaps, effectively determining whether the event under scrutiny contains oppositely-signed, high-mass muon pairs. The trigger decision is returned in 770 ns. A diagram of the spectrometer is shown in figure 3.2.[13]

3.4.1 Magnets

The upstream magnet, known as FMag, is a solid iron magnet with dimensions of 43.2 cm in width, 160 cm in height, and 503 cm in length. This magnet was assembled using iron slabs reclaimed from the dismantled Columbia University Nevis

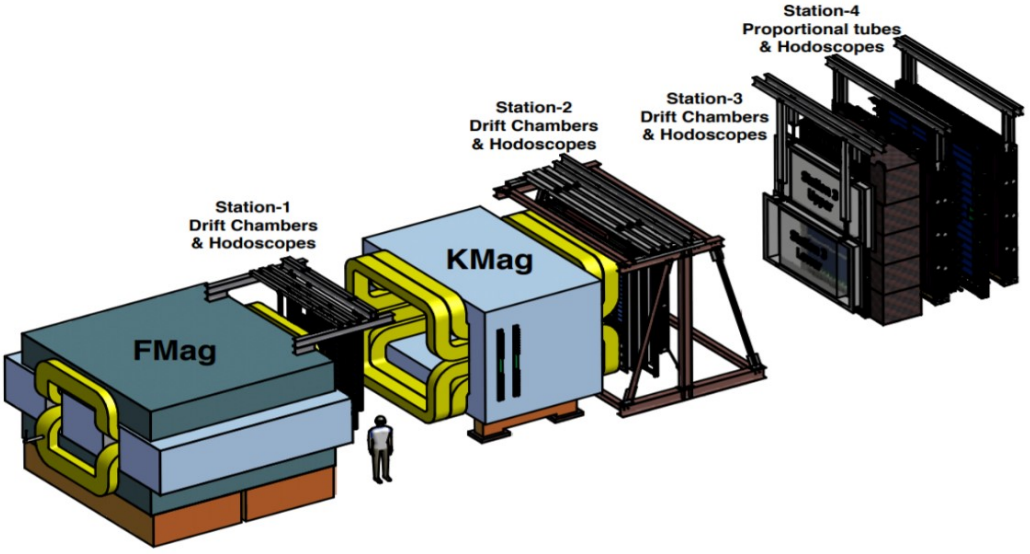


FIGURE 3.2: A schematic of the SeaQuest/SpinQuest spectrometer. It is comprised of two magnets and four tracking stations. The two magnets, FMag and KMag, are both dipole magnets. The first three stations are made of groups of drift chambers and hodoscopes, and the fourth station is comprised of proportional tubes and hodoscopes. The beam enters from the far left, and charged particles pass through the stations.[13]

Laboratory target cell insulation vacuum in 1980. FMAG is able to accommodate a current of 2000 amperes at 25 volts, consuming 50 kW of power. This configuration yields a magnetic field strength of 1.8 Tesla. This deflects the particles passing through it by angle $\theta = \frac{0.3\int Bdl}{p} = \frac{3.3GeV}{p}$ FMag utilizes one of the three sets of "bedstead" coils that were repurposed from the decommissioned E866 SM3 magnet. These coils are constructed from 5 cm square extruded aluminum.

Monitoring of the current in FMag is integrated into the Fermilab accelerator control system, and this data is relayed to the slow data acquisition system during each acceleration cycle. To ensure safety, the magnet current is incorporated into the safety system, preventing beam particles from colliding with the E906 spectrometer unless FMag is energized to a minimum level. The final calibration of the magnetic field was achieved through the examination of the reconstructed mass of the J/Ψ resonance.

FMag serves a dual purpose, acting both as a spectrometer magnet and as the beam dump for the 120 GeV beam directed toward the SeaQuest spectrometer. To increase the target-dump seperation, there is a 5 cm diameter, 25 cm deep hole drilled into the upstream end of FMag along the beam axis.

The downstream magnet, referred to as KMag, is an iron rectangular magnet, measuring 300 cm in length, with a central air gap of 289 cm in width and 203 cm in height. KMag is excited to a central magnetic field strength of 0.4 Tesla, equivalent to a 0.39 GeV/c magnetic deflection, achieved by applying 1600 amperes at 270 volts, consuming 430 kW of power. This deflects the particles passing through it by angle $\theta = \frac{\int Bdl}{p} = \frac{0.36GeV}{p}$.

Like FMag, the final calibration of the magnetic field is anchored in the precise measurement of the mass of the J/Ψ resonance. Both FMag and KMag are configured with their magnetic fields oriented vertically, ensuring that the bend plane is horizontal. In the normal operational state, both magnets collectively bend muons in the same direction, allowing for easier particle identification.

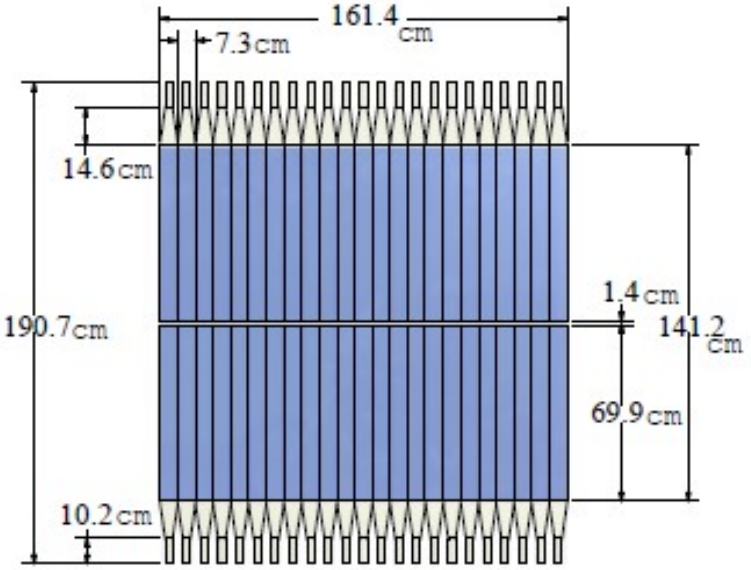
3.4.2 Hodoscopes

The detector hodoscopes work by using a combination of scintillating materials and photodetectors. When charged particles traverse a scintillating material, they interact with the atoms, inducing the excitation of electrons to higher energy levels. Subsequently, as these excited electrons return to their ground state, they emit photons in the form of visible or ultraviolet light.

Surrounding the scintillating material within hodoscopes are arrays of photodetectors, which detect the scintillation light and convert it into electrical signals. The spectrometer uses photomultiplier tubes (PMTs) on each end of the scintillator bars.

The spectrometer's primary trigger relies on four hodoscope stations constructed with plastic scintillators. These stations are configured with both x-planes (measuring the x-position) and y-planes (measuring the y-position). Station 1 and 2 each consist of a single x-y plane with 1-inch PMTs, Station 3 has a single x-plane, and Station 4 comprises two y-planes and one x-plane, all equipped with 2-inch PMTs. To enhance efficiency, the scintillator bars in each plane are slightly overlapped.

An example of one of these stations is shown in figure 3.3.



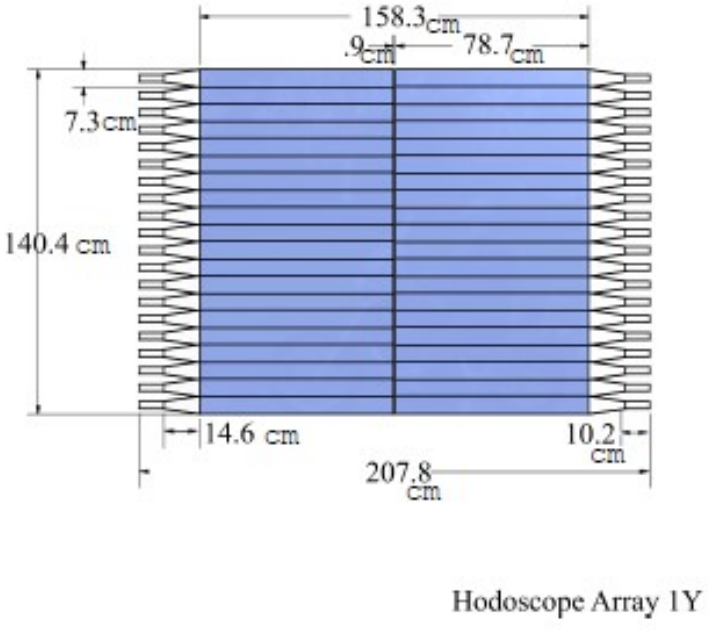


FIGURE 3.3: A drawing of the hodoscopes located in station 1, named H1X and H1Y. The X array has 46 scintillating tubes and the Y array has 40.[14]

Detector	Width (cm)	Overlap (cm)	# of paddles	x-position (cm)	y-position (cm)	z-position (cm)
H1T	7.32	0.32	23	162.00	69.85	667.12
H1B	7.32	0.32	23	162.00	69.85	667.12
H1L	7.32	0.32	20	78.74	140.12	654.03
H1R	7.32	0.32	20	78.74	140.12	654.03
H2T	13.04	0.32	16	203.24	152.00	1421.06
H2B	13.04	0.32	16	203.24	152.00	1421.06
H2L	13.07	0.32	19	132.00	241.29	1402.86
H2R	13.07	0.32	19	132.00	241.29	1402.86
H3T	14.59	0.32	16	227.52	167.64	1958.51
H3B	14.59	0.32	16	227.52	167.64	1958.51
H4T	19.65	0.32	16	304.52	182.88	2234.50
H4B	19.65	0.32	16	304.52	182.88	2250.68
H4Y1L	23.48	0.32	16	152.40	365.80	2130.27
H4Y1R	23.48	0.32	16	152.40	365.80	2146.45
H4Y2L	23.48	0.32	16	152.40	365.80	2200.44
H4Y2R	23.48	0.32	16	152.40	365.80	2216.62

TABLE 3.1: The hodoscopes present in the spectrometer, with the dimensions, positions, and number of scintillating paddles in each detector.

3.4.3 Drift Chambers

Drift chambers are designed to detect charged particles as they traverse through a gas-filled chamber. These chambers are equipped with wire planes, arranged as vertical x-position planes and left-right stereo-angle planes (u and v). When a charged particle passes through the gas, it ionizes the atoms along its path, creating free electrons. An electric field is maintained within the chamber, causing these free electrons to drift towards the wire planes. The time it takes for these electrons to reach the wires is recorded, allowing for the calculation of the particle's position. By measuring the drift times and positions in multiple wire planes, the trajectory of the particle can be reconstructed with high precision.

For the SeaQuest and SpinQuest experiments, the drift chambers each have a total of six wire planes. These chambers were the main source of tracking information for the experiment. To achieve high-precision mass measurements, it was important to ensure that the position resolution of each individual plane was very high. Specifically, the spatial resolution of an individual plane needed to be within the range of 400 μm. The resolution are defined as the average RMS value for each chamber, calculated by taking the difference between the measured position in a plane and the position calculated at the z-coordinate of that plane, using a fit where

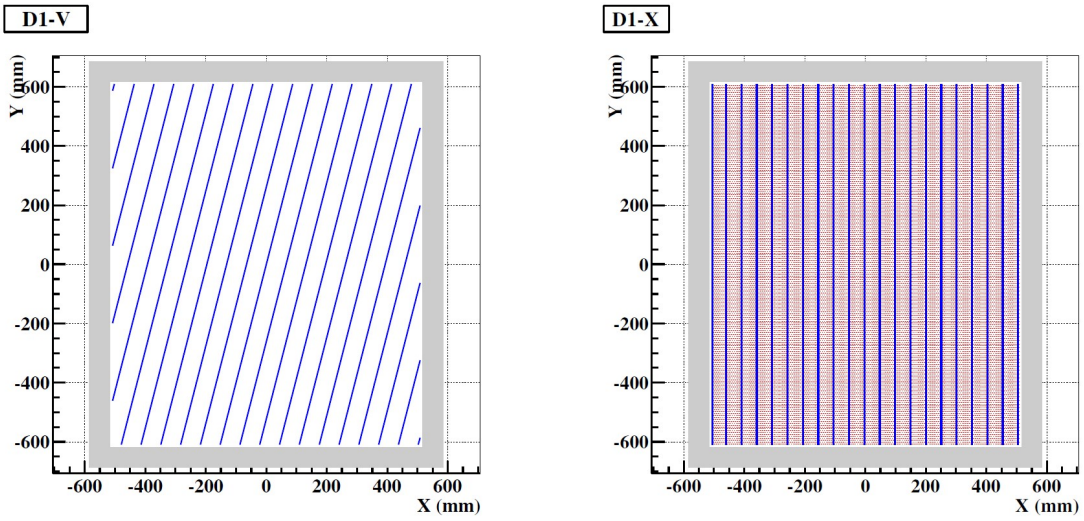


FIGURE 3.4: A drawing of the drift chambers located in station 1. The X-plane wires are oriented vertically, while the V and U (not shown) are angled 0.245 radians (14 degrees) from vertical. This allows for high resolution in the x-direction, which corresponds to the direction of bending from the spectrometer magnets.[14]

the plane is excluded from the calculation. This level of precision corresponds to a momentum resolution of 0.03%. This requirement ensures that the contribution of the position resolution to the overall mass resolution remains under 10%.

The tracking stations were not only designed for precision but also for efficiency. All the planes of all the detectors have over a 95% efficiency, based on analysis of drift chamber and hodoscope hits.

3.4.4 Proportional Tubes

Proportional tubes operate based on the principle of proportional gas amplification. When a high-energy charged particle travels through these tubes, it ionizes the gas within the chamber, creating free electrons, similarly to in a drift chamber. The applied electric field accelerated these electrons toward the central anode wire, where the wire acts as the primary collection electrode. As these electrons drift toward the anode wire, they create a detectable electrical signal, the magnitude of which is proportional to the number of ionization events initiated by the passing muon.

The proportional tubes are all located at station 4, downstream of a 1-meter-thick iron wall. Because of the previous beam dump, magnetic fields, and the

Detector	Cell Size (cm)	# of Cells	Tilt Angle (rad)	Width (cm)	Height (cm)	z-position (cm)
D1U	0.635	201	0.245	101.60	121.92	594.49
D1Up	0.635	201	0.245	101.60	121.92	595.13
D1X	0.635	160	0	101.60	121.92	617.09
D1Xp	0.635	160	0	101.60	121.92	617.72
D1V	0.635	201	-0.245	101.60	121.92	637.17
D1Vp	0.635	201	-0.245	101.60	121.92	637.81
D2V	2.021	128	-0.245	233.27	264.16	1314.98
D2Vp	2.021	128	-0.245	233.27	264.16	1321.96
D2Xp	2.083	112	0	233.27	264.16	1340.36
D2X	2.083	112	0	233.27	264.16	1347.34
D2U	2.021	128	0.245	233.27	264.16	1365.99
D2Up	2.021	128	0.245	233.27	264.16	1372.98
D3pVp	2.000	134	0.245	320.00	166.00	1923.33
D3pV	2.000	134	0.245	320.00	166.00	1925.33
D3pXp	2.000	116	0	320.00	166.00	1929.33
D3pX	2.000	116	0	320.00	166.00	1931.33
D3pUp	2.000	134	-0.245	320.00	166.00	1935.33
D3pU	2.000	134	-0.245	320.00	166.00	1937.33
D3mVp	2.000	134	0.245	320.00	166.00	1886.77
D3mV	2.000	134	0.245	320.00	166.00	1888.77
D3mXp	2.000	116	0	320.00	166.00	1892.77
D3mX	2.000	116	0	320.00	166.00	1894.77
D3mUp	2.000	134	-0.245	320.00	166.00	1898.77
D3mU	2.000	134	-0.245	320.00	166.00	1900.77

TABLE 3.2: Parameters of all the drift chambers. Stations 1 and 2 have one array of six drift chambers each, while station 3 has two arrays, which are in the same z-plane but offset in the y-plane.

final iron wall, the vast majority of particles passing through station 4 are muons. Much like the other stations, station 4 encompasses both triggering hodoscopes, as previously described, and tracking detectors. The tracking detectors in station 4 comprise four layers of proportional tube planes. Each of these planes is comprised of nine proportional tube modules, with every module constructed from 16 proportional tubes. These proportional tubes are 12 feet (3.66 meters) in length, with a diameter of 2 inches (5.08 centimeters), staggered to form two sub-layers.

The proportional tubes are oriented along either the horizontal or vertical direction, providing precise tracking in either the y or x-coordinate for the first and fourth planes and the second and third planes, respectively. The wall thickness of each tube is 1/16 inch, with the central anode wire having a diameter of 20

Detector	Radius (cm)	Layers	Modules in Layer	Tubes in Module	Tilt (rad)	x (cm)	y (cm)	z (cm)
P1H	5.08	2	9	8	0	368.3	388.6	2102.1
P1V	5.08	2	9	8	π	388.6	368.3	2178.8
P2H	5.08	2	9	8	0	368.3	388.6	2394.4
P2V	5.08	2	9	8	π	388.6	368.3	2371.3

TABLE 3.3: The properties of the four detectors comprised of proportional tubes. There are four detectors, each with 2 layers, 9 modules per layer and 8 tubes per module (for a total of 576 tubes). The detectors are split into two sub-stations, each with one detector each in the x- and y-orientations.

micrometers. The same gas mixture used in the drift chambers is employed within the proportional tubes.

Typically, a high-energy muon passing through the proportional tubes traverses two of these tubes in each plane, inducing hit signals on two anode wires. In the process of muon identification, a total of 8 hits from the four planes of the proportional tubes are used to construct a track that points back to the target.

3.5 The SeaQuest Target

The SeaQuest target was located 130 cm upstream of the first detector station. It had five target materials: liquid hydrogen, liquid deuterium, iron, carbon, and tungsten. There were also two target positions that were used to measure the background: an empty liquid flask and an empty solid-target holder. These targets were all placed on a movable table that moved the different materials in and out of the beamline between spills, as shown in figure 3.5.

The number of interactions expected from a target is dependent on both the density and length of the target material. Because of this, it is useful to consider the number of interaction lengths of a target. An interaction length is similar to a mean free path and is a measure of the average distance that a particle can travel through a material before it undergoes an inelastic interaction.

Because of their low density, the H_2 and D_2 targets were made significantly longer than the solid targets. With this design, the active targets all have interaction lengths on the same order.

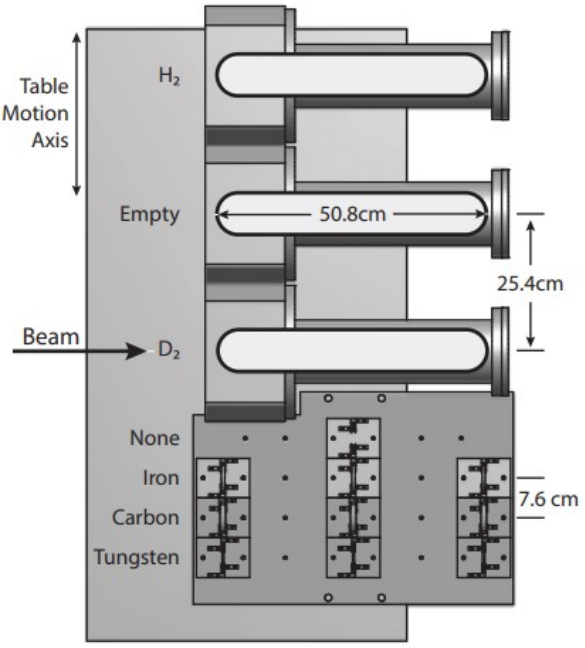


FIGURE 3.5: A top-view of the movable SeaQuest target. The target had seven possible positions, depending on the desired target.[13]

Position	Material	Density (g/cm^2)	Thickness (cm)	Interaction Lengths	Spills/ Cycle
1	H_2	0.071	50.8	0.069	10
2	Empty Flask	-	-	0.0016	2
3	D_2	0.163	50.8	0.120	5
4	No Target	-	-	0	2
5	Iron	7.87	1.905	0.114	1
6	Carbon	1.80	3.322	0.209	2
7	Tungsten	19.3	0.953	0.096	1

TABLE 3.4: A summary of the seven target positions in the SeaQuest experiment. The number of spills per cycle is approximate and was decided on to balance the expected number of interactions (from interaction lengths) with importance to physics analysis.

Because of the importance of the H_2 and D_2 measurements, the decision was made to collect more data from the cryogenic liquid targets. A cycle, comprised of 23 spills, was comprised of ten spills to the H_2 target, five spills to the D_2 target, two each to the empty flask, no target, and carbon target, and one each to the iron and tungsten targets. A summary of the target characteristics is shown in table 3.4.

3.6 The SpinQuest Target

For SpinQuest, the desired measurement of polarized Drell-Yan scattering requires a target capable of high levels of polarization that can withstand the high intensity and energy of the Main Injector Beam. The total beam energy is 22 kW, delivered over a 4.4-second spill, which means the beam delivers approximately 100 kJ every minute. This presents several challenges:

- Preventing magnet quenches
- Electronics cannot survive in target cave due to radiation exposure
- Radiation damage to the target
- Keeping the target polarized
- High helium expenditure

These challenges all influenced the design and operation of the target. Below, we discuss the different subsystems of the SpinQuest target, built by the University of Virginia and Los Alamos National Laboratory.

3.6.1 Target Material

There are many solid polarizable target materials that have been used in experiments. There are several factors that need to be considered when selecting a target material.

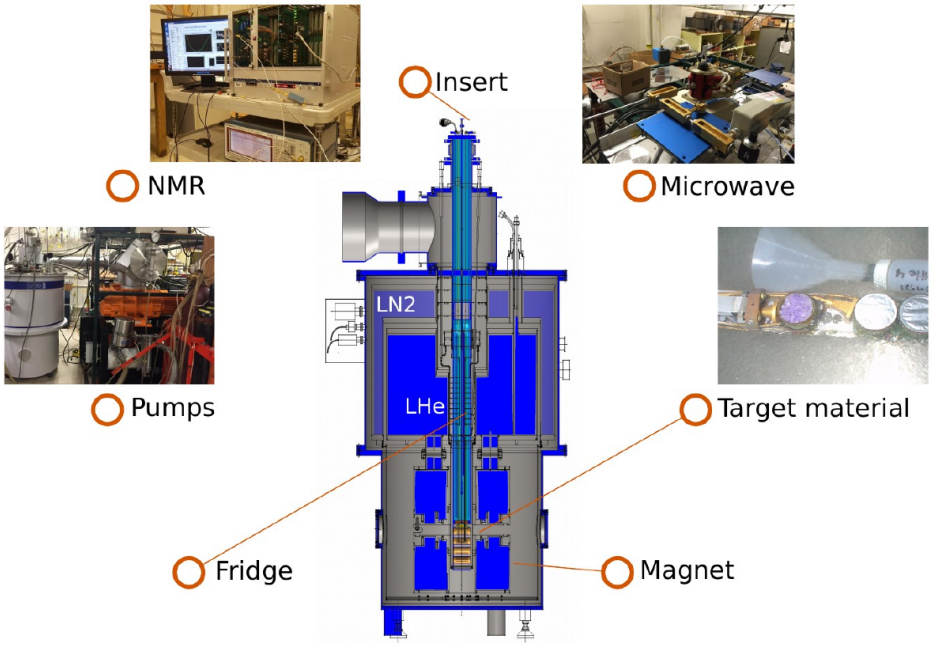


FIGURE 3.6: The SpinQuest targets and its subsystems. The target was designed and built by Los Alamos National Laboratory and the University of Virginia Target Group.[15]

The first two of these considerations are the degree of polarization of the material, P, and the ratio of the polarizable cross-section to the total cross-section of the target, known as the dilution factor f.

In a polarized scattering experiment, an asymmetry A is measured by comparing the number of events N in one spin configuration to the opposite spin configuration. If the experiment was able to achieve a polarization of 100%, this corresponds exactly to the cross-sections of the processes in the up and down spin configurations:

$$A = \frac{\sigma\uparrow - \sigma\downarrow}{\sigma\uparrow + \sigma\downarrow} \tag{3.1}$$

$$= \frac{N\uparrow - N\downarrow}{N\uparrow + N\downarrow}. \tag{3.2}$$

Unfortunately, due to a design flaw in the universe, this is not possible in any experiment. We can instead measure a counting asymmetry ϵ, given by

$$\epsilon = \frac{N\uparrow - N\downarrow}{N\uparrow + N\downarrow}, \tag{3.3}$$

This can be related to the desired asymmetry A by the relationship

$$A = \frac{\epsilon}{Pf}. \tag{3.4}$$

This correction accounts for the target having neither perfect polarization nor purity. Since we know the total number of events is proportional to the total measurement time, we can give the following relationship for the time required for the experiment to reach a given level of statistical error:

$$t \propto \frac{1}{\rho(Pf)^2} \tag{3.5}$$

where ρ is the density of the material. It is therefore important to select a material with a combination of high dilution factor and polarization level to reduce the required run time of the experiment.

Other important considerations for selecting a material include the difficulty of preparation and handling, the temperature at which it is polarized, the speed at which it is polarized (polarization build-up time), the presence of other polarizable nuclei, and resistance to radiation damage. Table 3.5 shows a summary of some of the more common materials used in polarized target experiments.[56]

Ammonia (NH_3) and deuterated ammonia (d-ammonia or ND_3) were selected as the best options for the target material for SpinQuest. This is due to their high dilution factors, polarizability, and radiation characteristic flux. Additionally, they polarize very quickly, which is helpful, since the targets will need to be replaced regularly in the experiment.

Since each spill delivers on the order of 10^{12} protons, and the radiation characteristic flux for ammonia and d-ammonia are on the order of 10^{16}, we will have to replace the target material on the order of once every 10^4 spills. Since a spill happens each minute, that is approximately one week[1]. Ammonia and d-Ammonia have a polarization build-up time on the scale of hours, as opposed to a scale

[1]This can be easily calculated using the *Rent* method of time calculation. Since there are 525,600 minutes in a year, 10,000 minutes is approximately one fifty-second of a year, or a week.

Material	**Temp. Required (K)**	**Dilution Factor**	**Maximum Polarization %**	**Radiation characteristic flux particles/cm^2**
Butanol $C_2H_4(OH)_2$	0.3	0.135	93	3×10^{14}
d-Butanol C_4D_9OD	0.3	0.238	50	3×10^{14}
Ammonia NH_3	1	0.175	97	7×10^{15}
d-Ammonia ND_3	1	0.30	50	1.3×10^{16}
Lithium deuteride LiH	0.2	0.50	70	4×10^{16}

TABLE 3.5: A summary of some common polarizable target materials used. The radiation characteristic flux is a measure of the radiation dose that reduces the maximum polarization of the material by a factor of $1/e$.

of days for Lithium deuteride, which saves a considerable amount of time in the experiment.

3.6.2 Target Material Production

Ammonia (NH_3) is a highly caustic and hazardous material, which means that it must be produced, handled, and stored carefully. It is dangerous to the eyes and can cause suffocation by destruction of the airways in high concentration. It is intolerable at densities larger than 35 mg/m^3. It freezes at 195.5 K and boils at 239.8 K. As a solid at 194 K, it has a density of 0.817 g/cm^3.

Because it freezes at nearly 200 K, it is easy to freeze ammonia in liquid nitrogen, which boils at 77 K. This can be done by percolating gaseous ammonia through liquid nitrogen. This tends to create a slug of solid ammonia, which must be crushed and sifted through screens to turn it into 1 mm beads.

To enable dynamic nuclear polarization, it is necessary to deposit paramagnetic radicals into the ammonia. This is done in two stages: first at "high" temperatures (80-90 K) and then at a low temperature (1 K). The high-temperature irradiation

has historically been performed using high-current, low-energy beams, on the order of 10^{14} 10 MeV electrons per second. The ammonia is held under cryogenic conditions in the beamline and accumulates dose. This was initially done with liquid nitrogen, but that caused unexpected explosions, which led to the use of liquid argon.[57]

This process requires the use of high-intensity, relatively low-energy electron beams. The UVA target group has irradiated materials at the Bates Linear Accelerator Laboratory, the SUNSHINE facility at SLAC, the Monterey Naval Research Facility, the Thomas Jefferson National Accelerator Facility, and the Saskatchewan Accelerator Laboratory.

Unfortunately, the availability of these beams for use to irradiate target material has become limited, with the vast majority of high-temperature irradiation now being done at one facility – the Medical-Industrial Radiation Facility (MIRF) at the National Institute of Standards and Technology (NIST).

Because beam time is limited, it is important to optimize the time that we have available at this facility. While effective at creating highly polarizable material, the existing method for irradiation is time-consuming and requires frequent manipulation of the ammonia, which requires beam downtime. For details on the existing method, see [58].

To help address this issue, we designed a new method for the irradiation of ammonia for use in polarized target experiments. It employs a larger, rotating basket to hold the ammonia to be polarized, which eliminates the need for the ammonia to be turned halfway through the irradiation, and presents a larger target for the beam, allowing a greater portion of the energy to be deposited in the ammonia.

A larger, rotating basket has two main benefits for the irradiation of target material. First, because a greater portion of the beam energy is deposited into the material, it increases the yield of irradiated ammonia for a set amount of beamtime. Second, it increases the homogeneity of the dose deposited to the ammonia, as the rotation eliminates the inhomogeneity created by the Gaussian beam profile.

The MIRF houses a traveling-wave linear accelerator (linac) with an energy range of 7 to 32 MeV. It can generate a beam current of up to 20 μA (the maximum current has decreased as the linac has aged).

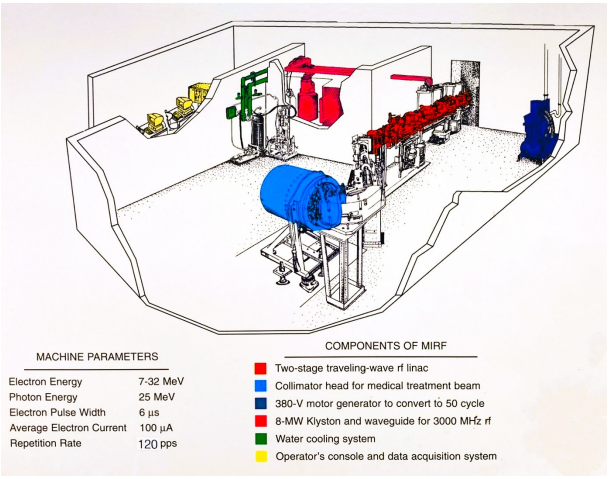


FIGURE 3.7: The Medical-Industrial Radiation Facility (MIRF) at the National Institute of Standards and Technology (NIST). Image of a poster created by Fred Bateman of NIST.

The linac was originally a radiotherapy machine used for fifteen years on an estimated 5 to 10 thousand patients before was donated to NIST by the Yale New Haven Radiation Oncology Center in 1992. The linac has been modified since its medical use, including converting it to 60 cycle, which allows for 120 beam pulses per second.

At NIST, the MIRF is used in a wide variety of applications. Its original primary application was to calibrate standards for radiation treatment for cancer patients. It has also been used to test the radiation hardness of materials, electron-beam curing, municipal waste sterilization, mail sanitation and to study material modification, which is the purpose for which we use it.[59]

When used for irradiating ammonia, the beam is set to an energy of 12.5 GeV, and a beam current of 10 μA, or approximately 6×10^{13} electrons per second delivered to the ammonia. The dose required to achieve good polarization is typically measured in electrons per square centimeter. For NH_3, the optimal dose is approximately 10^{16} to 10^{17} electrons per square centimeter.[56] The cup holding the target material has a cross-section of 8 square centimeters, which means that to achieve the desired dose, we irradiate the material for approximately 40 minutes. Because the cup is approximately 6 cm long, we irradiate on one side for 40 minutes, then turn the cup 180 degrees, and irradiate for another 40 minutes.

Figure 3.9 shows a simulation of the dose delivered per electron to the target cup along its length, in the existing geometry. This geometry assumes a beam widener 400 mm downstream of the beam exit, with the target material separated from

FIGURE 3.8: A photograph of the MIRF Linac.

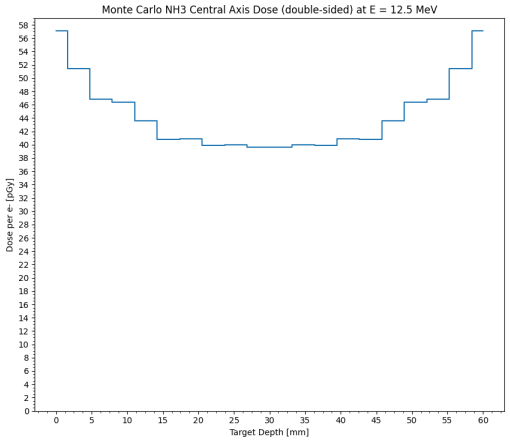


FIGURE 3.9: A simulation, performed by Fred Bateman of NIST, showing the dose delivered to the central axis of the target cup for the existing irradiation method.

the widener with 15 mm of liquid Argon. The total radiation delivered to the ammonia during the irradiation process is approximately 500 kGy/cc.

To increase the yield, as well as increase the homogeneity of the delivered dose, we have designed a rotary basket (as shown in figure 3.10). It is 8 cm in diameter and 3 cm in height. The basket sits in the beam path and in rotated at 1 rpm to

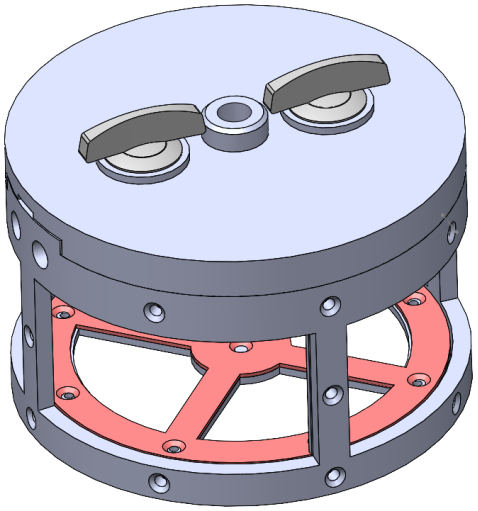


FIGURE 3.10: A design for a rotary target basket to increase yields of irradiated ammonia.

evenly distribute the dose angularly. The basket is closed with two quarter-turn cam latches on the lid. The frame is made of solid aluminum to reduce activation, and the bottom and sides of the basket have an aluminum mesh outer wall to hold the target material. It has a total interior volume of 151 cc, allowing us to irradiate a large amount of ammonia in a single batch.

The purpose of the new design is twofold: increasing the amount of material held in the basket and increasing the homogeneity of the dose received. In order to determine the optimal size of the basket, we need to determine two things: how homogeneous the dose received is, and how much more (or less) efficiently the material is produced. Other design considerations included the ease of access to the ammonia inside the basket and limiting the amount of material directly in the beam path that could be activated.

To calculate the dose to the target, we can model the electron beam as a Gaussian beam. A Gaussian beam has an intensity given by

$$I(r,z) = I_0 \left(\frac{w_0}{w(z)}\right)^2 exp\left(\frac{-2r^2}{w(z)^2}\right), \tag{3.6}$$

where r is the radial distance from the central beam axis, z is the axial distance from the beam waist, w_0 is the waist radius, and $w(z)$ is the radius of the beam at a given z. Because we are only interested in modeling a small portion of the z

axis, we can assume that $w(z)$ is constant over the area of interest, giving a beam intensity of

$$I(r) = I_0 exp\left(\frac{-2r^2}{\sigma^2}\right), \tag{3.7}$$

The dose deposited per electron is attenuated exponentially as the beam travels through the argon and ammonia. As such, we can model the dose received as a combination of an exponential decay in the x-direction and a Gaussian in the y-direction, where x is the direction of the electron beam. Thus,

$$\textbf{Flux} \propto e^{-\lambda z} e^{-\frac{r^2}{\sigma^2}} \tag{3.8}$$

From the simulation of our setup, we get values of $\lambda = 0.432$ and $\sigma = 1.776$. To determine the ideal diameter of the basket, we calculated the dose delivered to the ammonia as a function of the radius at which it sits. This is shown in figure 3.11.

To compare the dose distribution, we calculated the standard deviation of the normalized radial dose (non-uniformity). In addition, we calculated the length of time needed to irradiate the basket to an average dose of 500 kGy/cc, the total volume of the basket, and the improvement over the older basket design in terms of cubic centimeters irradiated per hour. The results for baskets of diameter 1 cm to 16 cm are shown in table 3.6.

The old version of the target cup has a non-uniformity value of 0.21, and takes 80 minutes to irradiate 29 cc of ammonia. Based on these calculations, baskets with a diameter of between 6 cm and 14 cm have an improved yield per hour (not counting the reduction in time needed to manipulate and change ammonia target cups). All sizes under 14 cm also improve the homogeneity of the dose delivered to the ammonia.

Based only on yield and homogeneity, a basket diameter of 9 or 10 cm would be ideal. However, there is one additional consideration that we need to take into account: the length of a workday. The MIRF is only available for use for 8 hours a day, which limits beamtime to approximately 6 hours per day with setup and cleanup accounted for. For that reason, an 8 cm diameter for our basket was determined to be the correct choice. At another facility with more flexible hours, a 9 or 10 cm diameter basket would probably be ideal.

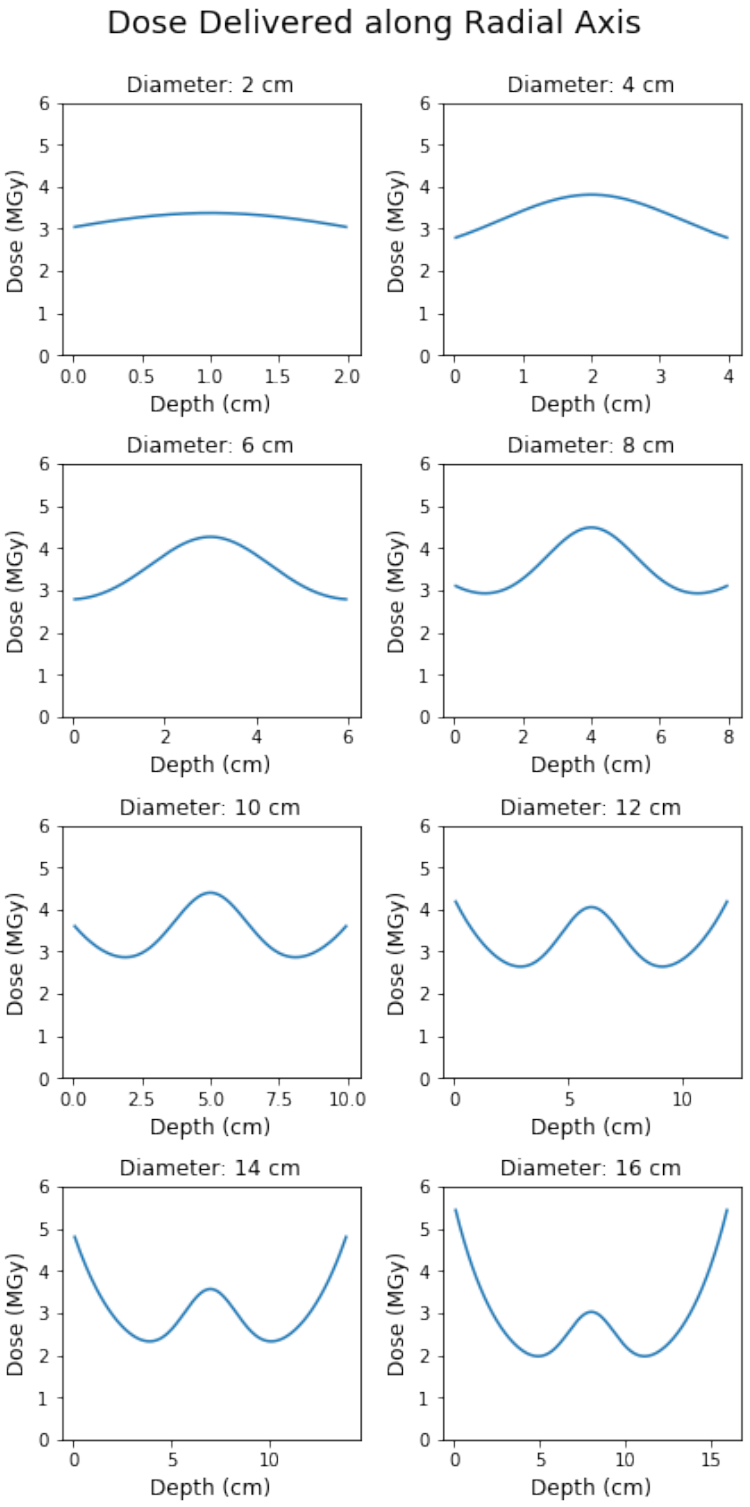


FIGURE 3.11: The radial dependence of the received dose for different dimensions of rotary baskets. Doses are normalized so that the average for each plot is 1.

Diameter (cm)	**Non-Unifor-mity**	**Irradia-tion time (hours)**	**Volume (cc)**	**Improvement**
2	0.03	1.1	9.42	-60%
4	0.10	2.0	37.7	-16%
6	0.15	3.5	84.8	10%
8	0.16	5.4	151	26%
10	0.15	8.3	236	28%
12	0.15	13	339	20%
14	0.21	20	462	6%
16	0.31	30	603	-10%

TABLE 3.6: Non-uniformity, irradiation time, volume, and improvement in production over traditional irradiation baskets for differently sized rotary baskets. The old style of basket has a non-uniformity of 0.21, a volume of approximately 29 cc, and takes 80 minutes to irradiate. Therefore, baskets smaller than 14 cm in diameter have an improved uniformity, and baskets between 6 cm and 14 cm in diameter have an improved yield compared to the older style.

FIGURE 3.12: Ammonia irradiated using the new basket design. The dark purple color indicates that the desired dose for high polarization has been achieved.

The novel method has effectively and efficiently irradiated NH_3. Following irradiation, the entire 125 g batch of ammonia exhibited a uniformly deep purple color, indicating that the polarizable electrons were evenly distributed throughout the sample. A photograph of the material after being placed in a storage bottle demonstrates these promising results, as shown in Fig. 3.12.

In the initial run, approximately 125 g of optimized target material was produced

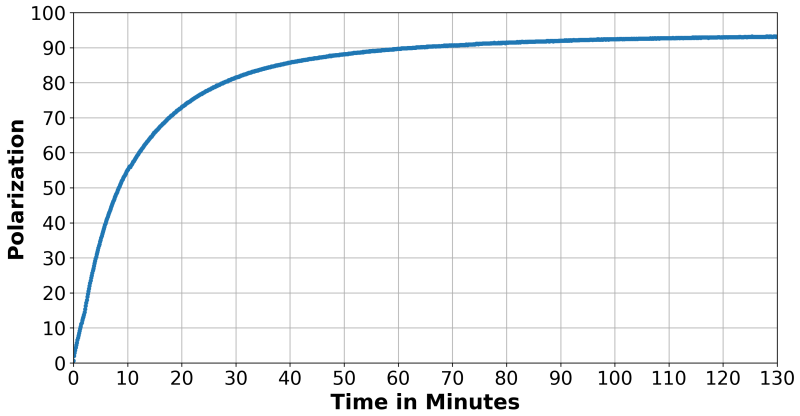


FIGURE 3.13: Polarization measurement of ammonia at the SpinQuest experiment irradiated using the novel basket design. Ammonia reached 95% polarization while recieving a dose of 10^{11} protons per second.

per day, involving 2.5 hours for setup and cleanup, and 5.5 hours for irradiation at NIST. In comparison, the previous method produced around 75 g of target material per day, requiring 4 hours for setup and cleanup, and 4 hours for irradiation. This represents an approximate 67% increase in daily material production.

Early results from the SpinQuest experiment indicate that the method does indeed produce highly polarizable ammonia. During beam and target commissioning using NH_3 produced using this method, polarization levels over 95% were observed in material recieving a dose of 10^{11} protons per second.

3.6.3 Polarization Measurement

In a typical polarized target experiment, there is a single nuclear magnetic resonance (NMR) coil in each cup. For SpinQuest, since the cup is unusually long, it has been equipped with three NMR coils that are evenly distributed along its length. This arrangement of NMR coils allows for a more comprehensive assessment of the polarization of the entire target. By capturing data from multiple points within the extended cup, researchers can obtain a more detailed and accurate depiction of the polarization, enhancing the effectiveness of the NMR system in this specialized environment.

Another challenge encountered in the SpinQuest NMR system arises from the placement of the NMR Q-Meter in relation to the coils. Due to the high levels of radiation present in the target cave, the NMR Q-Meter must be positioned

at a greater distance from the target than would be ideal. Consequently, the NMR cable extends over a distance of $14\lambda/2$, in contrast to the typical maximum length of $7\lambda/2$. This extended cable length introduces additional noise into the system, necessitating a more meticulous and inventive analysis approach to extract meaningful signal data amid the heightened interference.

Because of these challenges, the target is required to use a unique design to ensure the NMR system's functionality and reliability. With the three NMR coils, we can have a more comprehensive view of the target's polarization. Furthermore, to mitigate the impact of the extended NMR cable on signal quality, advanced signal processing techniques and noise reduction methods are applied.

Nuclear Magnetic Resonance (NMR) is a technique used for a wide range of applications, including the measurement of polarization in polarized targets. NMR relies on the magnetic moments of atomic nuclei to provide insight into their behavior. The principle of NMR revolves around the interaction of nuclear spins with an external magnetic field.

Polarized nuclear spins precess or wobble at a characteristic frequency known as the Larmor frequency, which depends on the strength of the external magnetic field and the gyromagnetic ratio of the specific nucleus under investigation. This precession forms the basis for measuring polarization.

To measure the polarization of polarized targets, we can use a variation known as Continuous Wave NMR (CW NMR). CW NMR allows for the measurement of the spin polarization of various nuclear species. Importantly, the NMR measurement can achieve an accurate reading even with impurities and contaminants in the target space. The method's accuracy stems from its reliance on the relationship between the integral of the NMR absorption signal and the static magnetization generated by the target spin species of interest.

The accuracy of CW NMR is not significantly compromised by the presence of higher-spin nuclei, even when quadrupole interactions are in play, provided that the magnetic dipole interaction with the static field is dominant. This method defines the vector polarization (P(I)) of a specific spin species I as the expectation value of the spin component aligned along the static magnetic field, relative to its maximum achievable value. These methods are selective in measuring magnetization and polarization based on the well-resolved Larmor precession frequencies of various nuclear spin species, ensuring precision in polarization measurements.[60]

3.6.4 Microwave Source and Dynamic Nuclear Polarization

To enable a high polarization of the target NH_3 and ND_3, the SpinQuest experiment will use a 140 GHz microwave generator to induce and maintain the polarization of the protons or deuterons in the ammonia and d-ammonia, respectively. This is done via a process known as Dynamic Nuclear Polarization.

As has been discussed, to achieve a good measurement of the Sivers function, a high level of polarization of the target must be achieved. Naively, we can polarize the target by applying a strong magnetic field and holding it at a low temperature. For a spin-1/2 particle, the level of polarization is given by

$$P_{TE}^{\frac{1}{2}} = \tanh \frac{\Delta E}{2kT} \tag{3.9}$$

and for a spin-1 particle by

$$P_{TE}^{\frac{1}{2}} = \frac{4 \tanh \frac{\Delta E}{2kT}}{3 + \tanh^2 \frac{\Delta E}{2kT}} \tag{3.10}$$

Where k is the Boltzmann constant, T is the Temperature, and ΔE is the gap between the energies of the magnetic levels from the Zeeman Hamiltonian

$$\mathcal{H}_z = -\hbar\gamma B_0 I_z \tag{3.11}$$

Which gives

$$E = -m\hbar\gamma B_0 \tag{3.12}$$

The SpinQuest target employs a 5 Tesla magnet and is capable of reaching temperatures as low as 1 K. If we plug this into the polarization level equation for an electron, we get an electron polarization of 99.56%. Unfortunately, the larger energy gaps for protons mean that we can only achieve a 0.51% polarization. For the deuteron, we get an even worse 0.10%, which corresponds to a -0.35% polarization of the neutron.

Thankfully, there is another polarization method available to us: dynamic nuclear polarization (DNP). DNP is a technique that leverages the high polarizability of electrons to our advantage. By introducing unpaired electrons, either by chemical doping or irradiation (as discussed in Section 3.6.2), we are able to leverage the high gyromagnetic ratio of the electron compared to that of the proton and deuterium nuclei and transfer the polarization via the use of a strong microwave field. The general setup for this process is shown in Figure 3.14.

The DNP process can be explained by using the *solid state effect* [61]. For simplicity, this section will only discuss the spin-1/2 nuclear spin case. The spin-1 case is mostly analagous, but includes more possible energy levels. This is because the magnetic field splits a system into $2J + 1$ possible sublevels.

Let us consider a system at temperature T and magnetic field B of N_I nuclear spins and N_S electron spins, with Larmor frequencies ω_I and ω_S (defined as γB, where γ is the gyromagnetic ratio of the nucleus or electron).

There are four possible pure states for this system. In increasing energy, they are $|\Uparrow\downarrow\rangle$, $|\Downarrow\downarrow\rangle$, $|\Uparrow\uparrow\rangle$, and $|\Downarrow\uparrow\rangle$, where the double arrow corresponds to the nuclear spin and the single arrow corresponds to the electron spin. There are four allowed single-spin transitions between these states, categorized into two "NMR transitions" and two "EPR transitions". NMR transitions are the transitions where nuclear spin flips, while the EPR transitions flip the electron spin. Flipping both spins is classically prohibited by dipole selection rules. These levels are shown in Figure 3.15.

$$H = \vec{\mu_e}\vec{B} + m\vec{u}_n\vec{B} + H_{SS}. \tag{3.13}$$

The mixing term allows for there to be four mixed states replacing the four pure states, defined as

$$|A\rangle = \alpha_1 |\Uparrow\downarrow\rangle + \beta_1 |\Downarrow\uparrow\rangle \tag{3.14}$$

$$|B\rangle = \alpha_2 |\Downarrow\downarrow\rangle + \beta_2 |\Uparrow\uparrow\rangle \tag{3.15}$$

$$|C\rangle = \alpha_3 |\Uparrow\downarrow\rangle + \beta_3 |\Downarrow\uparrow\rangle \tag{3.16}$$

$$|D\rangle = \alpha_4 |\Downarrow\downarrow\rangle + \beta_4 |\Uparrow\uparrow\rangle \tag{3.17}$$

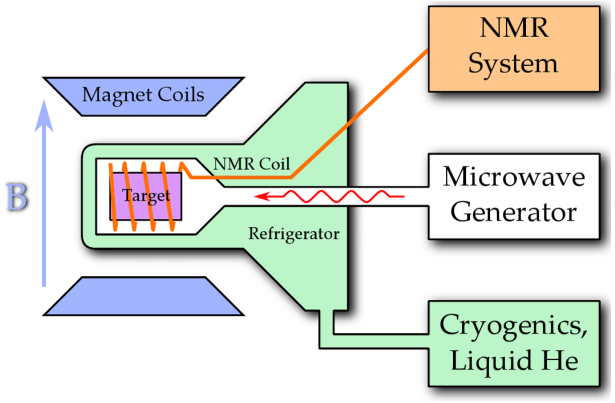


FIGURE 3.14: The basic setup for DNP polarization. A target is held in cyrofridge, where it is subjected to a high magnetic field. A microwave generator produces microwaves tuned around the Larmor fequency, which, when incident on the target material, allows for the transfer of electron polarization to the nucleons. The polarization of the material is measured using the NMR system.[16]

These mixed states allow for the forbidden double-spin flip transitions to occur.

When placed in a magnetic field, the most common state will be $|A\rangle$. Applying a microwave field with frequency $\omega_S - \omega_I$, we can force transitions of the $|B\rangle$ states to $|C\rangle$. Because the relaxion rate of electrons is very fast, these states will quickly transition to $|A\rangle$. Similarly, $|D\rangle$ states will transition into $|B\rangle$ states via electron relaxation, which allows them to undergo the same path to $|A\rangle$.

This means that we reach a state where the vast majority of electron-nuclear pairs are in the $|A\rangle$ state, allowing us to achieve high net polarization of the nucleon. [60]

The system for a spin-1 case involves similar principles but with a more complex energy level structure due to the additional sublevels. This increases the complexity of achieving polarization and lowers the maximum achievable polarization, but follows the same fundamental process of transferring electron polarization to nuclear spins via microwave-induced transitions.

3.6.5 Target Magnet

To achieve high polarization using DNP, it is necessary to have a high-strength, highly homogeneous magnetic field over the entire area of the target. To achieve this, SpinQuest will utilize a 5T magnet made by Oxford Instruments. It is homogenous to one part in 10^4 over the target area.

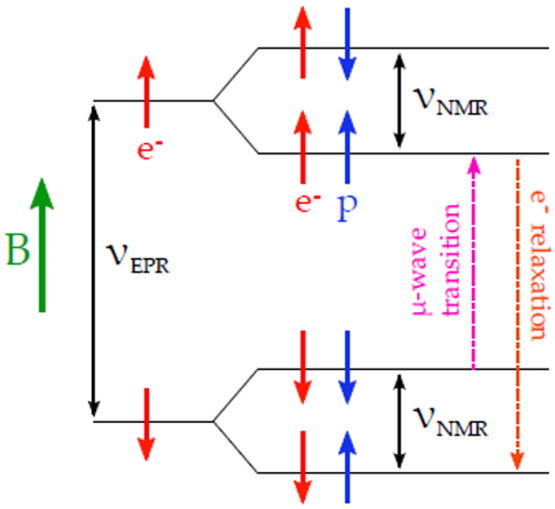


FIGURE 3.15: The energy levels of an electron-nucleon pair in a magnetic field. Solid arrows represent allowed transitions, while dashed arrows represent forbidden transitions.[16]

Due to the significant heat load on the target, simulations were necessary to determine if the intensity would cause a magnet quench. Zulkaida Akbar, a post-doctoral researcher formerly at the University of Virginia, used GEANT and COMSOL for these simulations. The results indicated that the system can withstand approximately 1×10^{13} protons per spill, which is twice the planned intensity.[62]

3.6.6 Evaporation Refrigerator

To maintain conditions for the magnet and target material, the experiment will utilize an evaporation refrigerator. This refrigerator has a power output of 1.4 W at 1 K and 3 W at 1.1 K, ensuring the temperature remains at a stable temperature during data taking.

An evaporation refrigerator uses liquid helium to maintain low temperatures. The process begins with the introduction of liquid helium into a low-pressure chamber. The liquid helium then undergoes controlled evaporation, transitioning into helium gas. This absorbs energy from the surroundings, resulting in a decrease in temperature within the chamber. The evaporated helium is pumped out of the chamber and replaced with more liquid helium, allowing for static temperatures.

The SpinQuest evaporation refrigerator uses a system of pumps, tasked with removing the evaporated helium. With an evacuation capacity of 17,000 m^3/hr, these high-powered pumps allow for a large heat load to be absorbed. Monte

Carlo modeling, performed by Zulkaida Akbar, shows that the target is able to withstand approximately 1×10^{13} protons per spill, which is double the planned intensity of protons[62].

3.6.7 Helium Liquifier

Given the quantities of liquid helium demanded by the experiment, it is vital to manage costs. Calculations revealed that the most economical strategy involved the in-house production of liquid helium. This is achieved by capturing the evaporated helium and subjecting it to a re-liquefaction process. The liquefier system is capable of producing approximately 200 liquid liters per day of liquid helium. However, owing to transfer efficiency considerations, only around 70% of this quantity is delivered to the target magnet.

Anticipating a daily consumption of approximately 110 liters of liquid helium, the production capacity is within the requirements of the experiment. This approach ensures that the experiment can sustain itself without relying on external helium sources. In the event of an unexpected quench, a contingency plan is in place, with 500 liters of helium stored, ready to replenish the magnet and minimize downtime.

3.7 Trigger System

Both SpinQuest and SeaQuest use various trigger setups for the spectrometer, optimized for different goals. Because there are many potenential sources of muons other than Drell-Yan from the material of interest, the main trigger for both is optimized to select dimuon pairs of invariant mass 4-10 originating in the region of the target. This does not completely exclude other sources of dimuons, such as charmonium-decay-induced dimuons and Drell-Yan from the beam dump.

SeaQuest and SpinQuest use two different types of triggers: Field-Programmable Gate Array (FPGA) triggers, and Nuclear Instrumentation Module (NIM) triggers. There are ten triggers, of which seven are regularly used in data taking, those being FGPA-1 through FPGA-5, NIM-1, and NIM-3. They are each optimized with different target particles in mind, as summarized in Table 3.7.

Each trigger also has a prescaling factor, which is the number of times the trigger condition must be met in order for an event to be recorded. For the first four types of triggers (FPGA-1 through FPGA-4) can be prescaled by a maximum of 24 bits, while the next four (FPGA-5, and NIM-1 through NIM-3) can be prescaled by up to 16 bits. The final two triggers, NIM-4 and NIM-5, cannot be prescaled because of equipment limitations.

3.7.1 FPGA-Based Trigger

The Field-Programmable Gate Array (FPGA) trigger system for SeaQuest and SpinQuest are comprised of nine CAEN V1495 VME modules. They are arranged into three levels, comprised of four, four, and one module, respectively.

The first level contains four modules, each linked to one of the hodoscope stations. It has two modes of operation, a "production" mode and a "pulser" mode. The production mode is a simple pass-through, and passes the hodoscope hits to the next level of the trigger. In pulser mode, however, it randomly generates hits from the hodoscopes, allowing the behavior of the subsequent layers to be verified.

The second level also contains one module per station, and acts as a rudimentary track finder, combining the output signals from the previous level into four-hit track candidates. The possible candidates are all pre-defined in a look-up table, based on Monte Carlo simulations of possible combinations from Drell-Yan events from the target. These combinations are referred to a "Trigger Roads", and allow us to select for Drell-Yan dimuon pairs, which would normally be overwhelmed by the much higher cross-section J/ψ-produced dimuon pairs. The trigger roads are grouped into "Road Sets", and have been iteratively improved throughout the use of the spectrometer and trigger system.

The third level is comprised of a single module, and is the "track correlator". It takes the track candidates from the previous level and determines if they satisfy requirements for each of the five trigger outputs, FPGA-1 through FPGA-5.

FPGA-1 and FPGA-2 are optimized to detect occurences of concurrent positive and negative muons. Their trigger conditions require there is at least one positive trigger road and at least one negative trigger road present within the event time window. For FPGA-1, there must be opposite-sign trigger roads present in opposite-position hodoscopes (positive in Top and Negative in bottom or vice

versa). FPGA-2 requires opposite-sign trigger roads to be in the same-position hodoscopes. The FPGA-1 trigger is the trigger used to collect Drell-Yan dimuons, and therefore has a prescaling factor or 1. FPGA-2, meanwhile, is unlikely to include usable Drell-Yan data, and thereforehasa prescaling factor of 10,000.

FPGA-3 is optimized to detect same-sign muons passing through opposite sides of the hodoscope array. Its conditions are met if two or more like-sign trigger roads are met in opposite position hodoscopes. It was used for background estimation in prior analyses. It has a prescaling factor of 123.

FPGA-4 and FPGA-5 are single-muon triggers. FPGA-4 has the most lax trigger condition of the FPGA triggers, requiring one or more trigger road of either sign. It is useful for investigating the behavior of the spectrometer as well as certain schemes of background estimation, and has a prescaling factor of 25,461. FPGA-5 adds the requirement of a high transverse momentum (over 3 GeV/c in the x-direction). These single muons are very similar to the muons that make up the Drell-Yan pairs that we are interested in, which makes FPGA-5 useful for investigating the behavior of those muons. It has a prescaling factor of 2,427.

3.7.2 NIM-Based Trigger

NIM-1 and NIM-2 are vestigal triggers from the beginning of the SeaQuest experiment, when FPGA-based triggers were not complete. They do not use trigger-road information, and only require concurrent hodoscope hits in certain combinations. The NIM-1 trigger requires coincident hits in all four hodoscope stations, all in either the top half or bottom half of the arrays. NIM-2 requires this condition to be met for both top and bottom hodoscopes. Becuase they are no longer required, their prescaling factors are set to 31,991.

The NIM-3 trigger is unlike all of the other triggers, in that it does not have any detector-based requirements. It is based on an RF-signal provided by the accelerator and a 7.5 kHz pulse generator, and requires them to be concurrent. It has a prescaling factor of 125.

While useless for detecting dimuon pairs, the NIM-3 is extremely useful for study, especially for the purposes of training our neural networks, as discussed in Chapter 5. This is because NIM-3 events are an unbiased, random snapshot of the spectrometer output, allowing us to obtain real background data to combine with

Monte Carlo events. When combined in the right way, this allows us to create truly realistic Monte Carlo training data for our networks

As previously mentioned, NIM-4 and NIM-5 are not able to be prescaled. They are not used for any physics purposes, but are used to detect the "beginning of spill" signal and "end of spill" signals sent by the main injector beam.

Trigger Name	Particle Target	Required Hodoscopes	P_T cut	Prescaling Factor	Notes
FPGA-1	$\mu^+\mu^-$	TB or BT	-	1	Main Physics Trigger
FPGA-2	$\mu^+\mu^-$	TT or BB	-	10,000	Same-Side Trigger
FPGA-3	$\mu^+\mu^+$ or $\mu^-\mu^-$	TB or BT	-	123	Like-Charge Trigger
FPGA-4	μ^+or μ^-	T or B	-	25,461	All-singles Trigger
FPGA-5	μ^+ or μ^-	T or B	$P_x > 3$ GeV/c	2,427	High-P_T Trigger
NIM-1	μ^+ or μ^-	T or B	-	31,991	Hodoscope-Coincidence Trigger
NIM-2	Multiple μ	T and B	-	31,991	Hodoscope-Coincidence Trigger
NIM-3	-	-	-	125	Random Trigger
NIM-4	-	-	-	-	Beginning of Spill
NIM-5	-	-	-	-	End of Spill

TABLE 3.7: The SeaQuest and SpinQuest triggers and their characteristics.

Chapter 4

Reconstruction and Tracking

Reconstruction and tracking algorithms are necessary to make sense of the detector information from the SeaQuest and SpinQuest experiments. The key challenge lies in accurately tracking the paths of these particles through the detector layers.

In order to access the physics information, we need to be able to accurately determine the path taken by particles as they pass through the detector array, as well as calculate the particle qualities that those paths are associated with. To do this, we can employ specialized tracking algorithms.

The SeaQuest and SpinQuest experiments have used a tracking algorithm known as KTracker, a C++ program adapted from a Fortran code that was first used in the 1980s. Although KTracker works relatively well, it has several drawbacks that can be addressed by using new technologies, specifically neural networks and parallelization.

In this chapter, we will first provide a brief overview of the existing KTracker algorithm. We will then discuss the development, training, and performance of our new reconstruction system, QTracker.

4.1 KTracker

Although it is not the focus of this dissertation, a brief overview of KTracker is warranted to provide context and highlight the need for advancements.

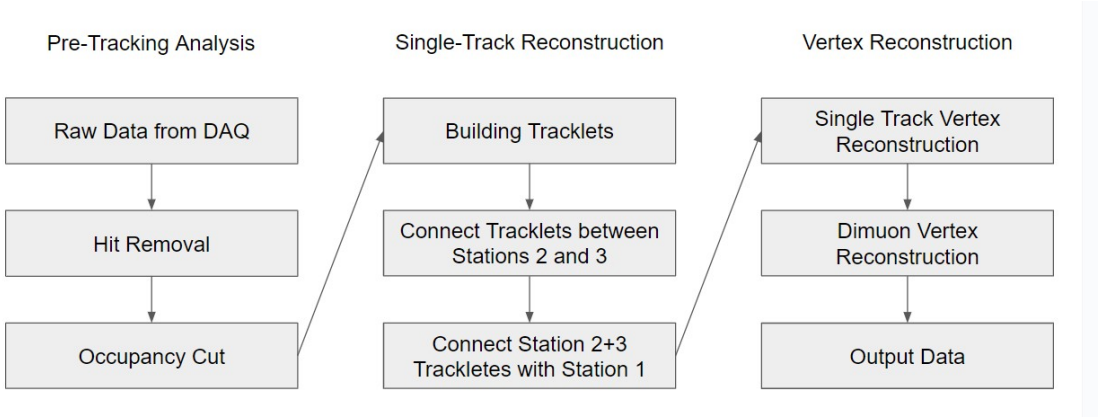


FIGURE 4.1: A flowchart representation of the KTracker algorithm.

KTracker is a C++ program adapted from a Fortran code originally employed in the 1980s. Its primary function is to trace the paths of particles as they traverse the layers of the detector array. This entails determining the trajectory of muons, calculating their momenta, and pinpointing the locations of particle interactions or decays.

The KTracker algorithm can be broken up into three sections: pre-tracking analysis, single-track reconstruction, and vertex reconstruction. Because we will be utilizing certain aspects of the pre-tracking analysis, we will go into more detail on those aspects, and provide more of an overview for reconstruction.

One of KTracker's drawbacks lies in its computational efficiency. KTracker's performance is significantly slowed when faced with high particle or hit multiplicities. This is because of its structure, where hits are compared in a combinatorical way, leading to a reconstruction time proportional to the factorial of the number of hits checked. These challenges necessitate an exploration of novel approaches to tracking and reconstruction.

4.1.1 Pre-Tracking Analysis

Pre-tracking analysis has two parts: hit removal and occupancy cuts. Hits to be removed are grouped into four categories: out-of-time hits, after-pulses, cluster hits, and random noise.

4.1.1.1 Hit Removal

Non-correlated hits, termed "extra hits," can significantly slow down the processing time of track reconstruction. It is important to eliminate these extra hits to enhance the accuracy and speed of track reconstruction. These extra hits can be categorized into four distinct types: out-of-time hits, after-pulses, cluster hits, and random noises.

Out-of-time hits refer to hits that fall outside the specified TDC time window and are discarded, as they do not represent proper hits. Random hits can be effectively filtered out by utilizing hodoscope hit positions. Since true hits, which are correlated with the muon track, consistently fall within the range of hodoscope paddles, the removal of random hits is achieved through chamber-hodoscope hit matching.

After-pulses are hits that emerge on signal wires subsequent to the occurrence of a genuine signal. In the analysis process, only the first hit on a wire is considered to eliminate after-pulses effectively. Extra hits that occur in close proximity to hit wires form what is referred to as a "hit cluster." Various types of hit clusters are categorized based on their size. For instance, "Edge hit" clusters arise when a muon traverses the junction of two drift chamber cells, producing hits in both cells. In this scenario, one of the hits within the "Edge hit" cluster is removed, as both are genuine hits, and only one is utilized for track reconstruction. The criterion for removal is typically based on the larger drift distance hit.

When charged particles traverse the drift chamber, the electronic circuitry may become unstable, increasing the likelihood of noise signals. "Electronic noise" clusters are thus formed, consisting solely of hits that lack any correlation with a muon track. All hits within such clusters are regarded as extra hits and are consequently removed.

Furthermore, "Delta ray" clusters manifest around genuine hits as a result of emitted delta rays from charged particles. Within a "Delta ray" cluster, one hit must correspond to a true hit, while the others are deemed extra hits. Consequently, all hits within the cluster, except those flanking the true hit, are removed as extra hits.

4.1.1.2 Occupancy Cuts

In some instances, events still have an excessive number of hits even after the initial hit removal process. To address this issue and filter out such events, events above a certain chamber occupancy are removed. In this context, "occupancy" refers to the count of hits remaining in each drift chamber following the initial hit removal. Specific occupancy thresholds are detailed in Table 4.1. Events that meet the criteria specified by the occupancy cut proceed to the reconstruction phase.

Detector	Occupancy	Number of total sense wires
St. 1	320	1124
St. 2	160	736
St. 3+	150	768
St. 3-	150	768

TABLE 4.1: The upper-limit occupancy cuts imposed by KTracker.

The occupancy cuts were chosen to correspond to the maximum occupancies for which KTracker is able to find dimuons and minimize the number of true dimuons thrown away. It is used in order to reduce total computational time by eliminating high multiplicity events that are unlikely to have any valuable data in them.

4.1.2 Single Track Reconstruction

The single-track reconstruction phase involves several steps aimed at reconstructing single muon tracks:

The first step is building "tracklets" within each drift chamber. Tracklets are small local tracks consisting of hit pairs in the drift chamber planes (X, X', V, V', U, U'). The process begins with identifying hit pairs in the XX' planes and then extends to the UU' planes based on the window determined by the hit pairs in the XX' planes. If that matching succeeds, the VV' planes are searched in a window corresponding to the geometry of the chambers. The layout of the different planes in a single drift chamber station is shown in figure 4.2.

All possible tracklets in St. 2 and St. 3 are reconstructed. These tracklets are then combined to form back partial tracks downstream of the second spectrometer magnet (KMag). The use of hits on X planes in both St. 2 and St. 3 allows for

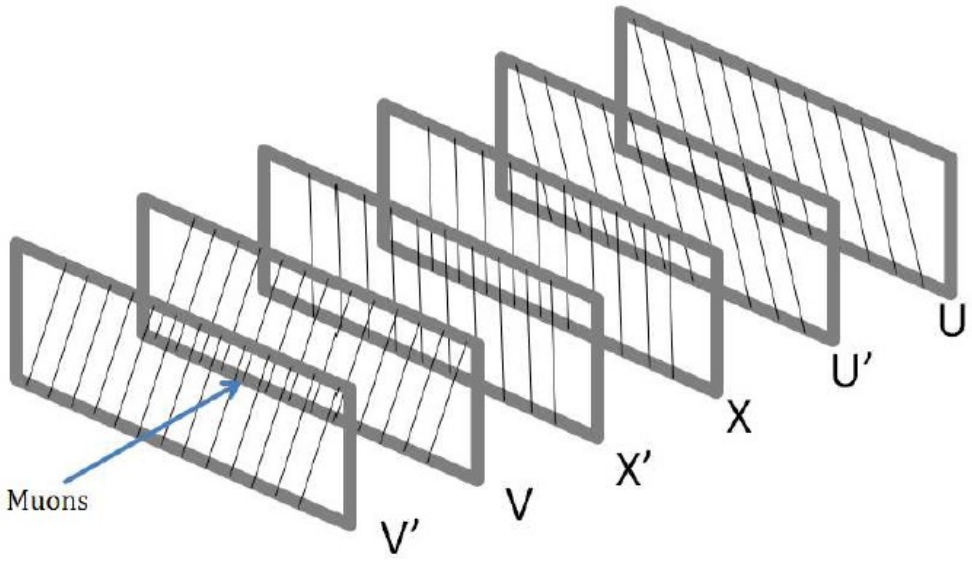


FIGURE 4.2: A cartoon representation of the drift chamber structure in the spectrometer.[14]

a quick calculation of the X-Z slope and intersection. Quality criteria, including checks on slopes, intersections, and proportional tube hits for muon identification, are applied. Tracks that meet these criteria are labeled as "back partial tracks."

The next step is to connect tracks from downstream to upstream of KMag. Back partial tracks are projected backward to the St. 1 drift chamber through the KMag magnetic field. A search window for St. 1 is determined based on the sagitta ratio between St. 1 and St. 2. Back partial tracks combined with St. 1 partial tracks are known as "global tracks." Each global track candidate undergoes a chi-squared fit to assess its quality.

These global tracks are described with several parameters: charge (s), $p_{x,y,z}$, t_x, t_y, x_0, and y_0, representing charge, momentum, slopes, and intersections in the X-Z and Y-Z planes. Iterative steps are employed to remove bad hits and enhance track quality. Selection criteria, such as hit count and momentum range are used to retain the final high-quality global tracks.

The ultimate goal of this single-track reconstruction process is to accurately identify and reconstruct single muon tracks while ensuring they meet specific quality standards and parameters. This improves the precision and efficiency of the analysis.

4.1.3 Vertex Reconstruction

The reconstruction of the reaction vertex involves two distinct steps: single-track vertex reconstruction and dimuon vertex reconstruction. The dimuon vertices are reconstructed based on the vertex information of the single tracks (corresponding to their closest approach to the beamline).

In the single-track reconstruction step, tracks downstream of the St. 1 drift chamber are reconstructed, focusing on the upstream part from FMag to the vertex position. The FMag, composed of iron slabs, introduces energy loss and bending due to the magnetic field. These two effects are calculated when projecting the path of the muon through the magnet.

To address the energy loss accurately, FMag is divided into 100 slices for analysis. Each FMag slice is subject to a p_T kick performed at its center, which involves calculating the track slopes at the downstream and upstream surfaces of the slice. KTracker implements a single value for energy loss through FMag, which is one of the main sources of uncertainty in this reconstruction, as real particles are scattered randomly, creating a non-uniform energy loss.

Upon completing this process for the entire FMag, the track is projeted to the beamline. Even in regions without a magnetic field or solid material, the extrapolation occurs in 2.5 cm slices to assess the vertex position. The distance between the beamline and the track at each slice is calculated to derive the z-position of the slice that corresponds to the vertex position.

The vertex determination is performed by evaluating the "distance of closest approach" (DOCA), which represents the smallest distance between the beamline and the track among all the slices. This process assumes that any detected muon originated along the beamline, which may not be accurate, as secondary scattering effects are also known to produce muons.

The dimuon reconstruction step is, unsurprisingly, the stage where dimuons are reconstructed. Initially, all the single tracks reconstructed within an event are classified as positive muons (μ^+) or negative muons (μ^-). All possible combinations of μ^+ andμ^- are tested to determine whether they construct dimuons.

The dimuon vertex position is determined based on the positions of the two tracks at each slice. The slice position, which represents the vertex position, is identified

as the point where the distance between the beamline and the difference of the track positions is minimized. This vertex position serves as the initial parameter for the iterative analysis.

Using a Kalman-Filter method[63], the vertex position is progressively updated until the chi-squared value reaches its minimum, indicating the optimal vertex position. The momenta from the single-muon reconstructions are then combined with this vertex information to describe the dimuon pair.

4.2 QTracker

The decision to develop QTracker as a complementary system to KTracker arose from a recognition of the potential for improvement in certain aspects of the tracking process. KTracker offers accurate reconstructions of particle momenta and vertex locations. However, KTracker has certain limitations, which we seek to remedy with QTracker.

One of the key considerations was the computational speed of KTracker. While KTracker's precision and accuracy are good, its speed can be improved, especially in scenarios with complex events involving a higher number of detector hits. This has been attempted using both multithreaded CPU and GPU programming. Using a completely new approach, however, QTracker can speed reconstruction even more than parallelization.

For QTracker, we have a number of goals:

- Increase the reconstruction speed.
- Enhance the precision and accuracy of selecting dimuon events.
- Increase signal DY statistics by improved recognition of dimuon events and reducing required cuts.
- Boost the precision and accuracy of kinematic reconstruction.
- Enhance the resolution for vertex location.

To achieve these goals, we will use a combination of Convolutional Neural Networks (CNNs) Feed-Forward Neural Networks, and non-AI Python scripting. We will use

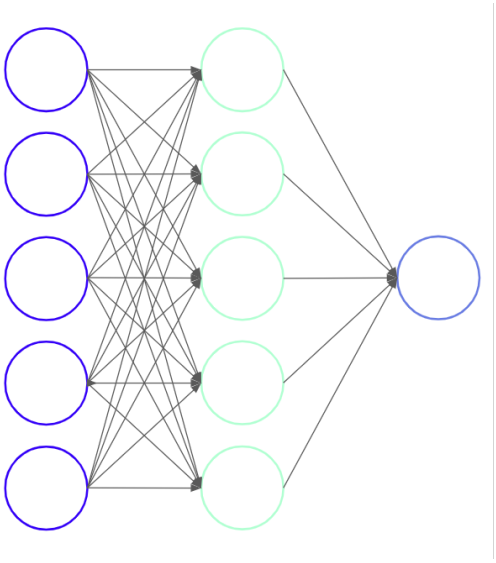


FIGURE 4.3: A visualization of a simple Neural Network. There are five input variables, and one output layer, with a single hidden layer containing five neurons.

the existing Monte Carlo framework to generate data to train on, using E906 data as a guide to generate realistic training data.

All of the necessary code for training data generation, network definitions, training, and execution are collected in a GitHub repository.

4.2.1 Neural Networks

Neural networks, inspired by the brain's neural structure, were first conceptualized in the 1940s by Warren McCulloch and Walter Pitts [64]. These early artificial neurons, or perceptrons, aimed to simulate information processing and decision-making like the human brain but in a simplified form. A diagram of a simple neural network is shown in Figure 4.3.

Enthusiasm for neural networks waned during the late 1950s and 1960s when researchers, led by Marvin Minsky and Seymour Papert [65], demonstrated the limitations of single-layer perceptrons in solving non-linear problems. The period following this is often referred to as the "AI winter."

Nevertheless, neural networks experienced a revival in the 1980s and 1990s. The development of multi-layer perceptrons and the introduction of backpropagation as a training algorithm [66] rekindled interest. Deep learning, characterized by

deep neural networks, emerged as a pivotal development in the hands of Geoffrey Hinton, Yoshua Bengio, and Yann LeCun [67], leading to breakthroughs in various domains.

Convolutional neural networks (CNNs) were a transformative milestone in artificial intelligence, particularly in computer vision. Originating in the late 1990s [68], CNNs draw inspiration from the human visual system, excelling in tasks like image classification and object detection. Their success extends beyond vision into areas like natural language processing and reinforcement learning, making them a cornerstone of modern deep learning, contributing to the resurgence of neural networks across industries.

Today, neural networks find applications across fields such as healthcare, finance, and autonomous vehicles. Specialized hardware, such as GPUs, has accelerated deep neural network training. We are now in the midst of an "AI Boom" driven by these improvements, with companies eager to jam AI into every imaginable product.

Once the structure of a neural network has been defined, it must be trained. This process is necessary, as it allows the network to learn from data and make predictions or classifications. At the heart of neural network training lies backpropagation, an algorithm that adjusts the network's parameters based on the disparity between its predictions and the actual outcomes. By iteratively propagating these errors backward through the network and updating the weights and biases accordingly, backpropagation enables the network to improve its performance over time. Various optimizers are used in the training of Neural Networks, including stochastic gradient descent (SGD), Adam, and RMSprop. These optimizers differ in their approaches to adjusting learning rates and momentum, thereby impacting the speed and effectiveness of the training. Choosing the right optimizer is crucial for achieving the best performance, as it can significantly affect the convergence rate and the network's ability to generalize from training data to unseen data.

To allow the computer to evaluate the performance, we use "loss functions". These functions quantify the disparity between the network's predictions and the actual outcomes, providing a measure of how well the network is performing on a given task. The two loss functions that we use are categorical cross-entropy and mean square error. Categorical cross-entropy is employed in classification tasks where the output is a probability distribution over multiple classes. It measures the

dissimilarity between the predicted probabilities and the true class labels, encouraging the network to assign higher probabilities to the correct classes.

Mean square error is used for regression tasks (such as the kinematic and vertex reconstructions) to measure the average squared difference between the predicted values and the actual targets. By minimizing this discrepancy, the network learns to accurately predict continuous values.

Learning rates are a critical parameter in the training of neural networks, influencing the speed and stability of the optimization process. The learning rate determines the magnitude of adjustments made to the network's parameters during each iteration of training. Too high a learning rate can lead to overshooting the optimal solution, causing instability and divergence in the optimization process. On the other hand, a learning rate that is too low can result in slow convergence, or find a local minimum in the loss function.

Epochs represent the number of times the entire training dataset is presented to the network for learning. During each epoch, the network adjusts its parameters based on the training data to minimize the loss function. Determining the number of epochs involves striking a balance between underfitting and overfitting. Too few epochs may lead to underfitting, where the network fails to capture complex patterns in the data, akin to ending a race prematurely before reaching the finish line. Conversely, too many epochs may result in overfitting, where the network learns to memorize the training data rather than generalize to unseen examples.

In our training, we utilize callbacks, which are mechanisms to monitor and intervene based on specific conditions or events. During training, after each epoch, the model's performance on the validation dataset is evaluated. If the validation loss starts to increase over a certain number of epochs, it indicates that the model may be overfitting. Once the network fails to improve for a certain number of epochs, the training is terminated and reverts to the configuration it had when the validation data was most accurately predicted by the network.

4.2.2 Monte Carlo Generation

In order to train the neural networks, we need a large amount of realistic, accurately categorized data. This presents a problem: although there is copious experimental data that we can draw from, the only tool we have to categorize and

reconstruct the muons is KTracker. Since KTracker is relatively slow (less than an event per second is analyzed) and inefficient (only approximately 10% of events containing dimuons are reconstructable), we must create our own training data from scratch.

To do this, we can utilize the pre-existing Monte Carlo-simulated physics processes used by the SeaQuest and SpinQuest collaborations. The system uses two programs, Pythia, a general-purpose Monte Carlo Event Generator, and Geant4, a toolkit for simulating particles passing through matter. This Monte Carlo gives us the clean particle tracks of muons through the detector array. We are then able to combine these tracks with a combination of partial muon tracks and random hits to mimic the distribution of detector information from the real E906 data.

4.2.2.1 Pythia Monte Carlo

The simulation framework for Drell-Yan and charmonium decay dimuon events combines the event generation capabilities of Pythia with the accurate tracking and interaction modeling provided by GEANT4. The system, in the context of the SeaQuest and SpinQuest collaborations, is known as Fun4Sim.

Pythia is employed to generate Drell-Yan events, simulating the annihilation of a quark-antiquark pair that produces a virtual photon or Z boson. This virtual particle then decays into a muon-antimuon pair. Pythia uses perturbative QCD calculations for the hard scattering process, along with parton showering, hadronization, and decay processes. For charmonium decay events, Pythia is used to simulate the production of charmonium states such as J//Psi. The subsequent decay of these states into dimuons is modeled, accounting for both direct decays and radiative decays.

The generated events from Pythia are then processed by the GEANT4 simulation, which emulates the passage of particles through a detailed detector array. This simulation includes the definition of the detector geometry, materials, and components. The detector geometry is defined within the GEANT4 framework, incorporating details of the experimental setup such as the tracking detectors, hadron dumps, and magnetic field configurations. The materials and dimensions of each detector component are specified to replicate actual experimental conditions.

GEANT4 tracks the generated particles through the defined geometry, simulating their interactions with detector materials. Electromagnetic and hadronic processes are modeled, accounting for ionization, multiple scattering, energy deposition, and decay processes. The toolkit ensures a realistic representation of particle trajectories and their interactions within the detector.

The simulation output, which includes information on particle tracks, and other relevant parameters including the results of a trigger emulation. The output of the Monte Carlo can then be resampled to better represent the full range of possible muon momenta, allowing for as unbiased training data as possible.

4.2.2.2 Event Monte Carlo

The inputs for QTracker are 54×201 matrices representing the detector ID and element ID of each detector and element of said detectors, which are referred to as "hit matrices". For QTracker, three hit matrices are made for each event, one for binary hits (hit represented by 1, non-hit represented by 0), one for drift (value ranging from 0 to 1 depending on drift value for said element) and TDC time (value equal to the TDC time). An example of a binary hit matrix for a real E906 event is shown in Figure 4.4. Because the TDC time matrix will only be used for timing cuts and hit reductions, we do not train the neural networks on those.

The first step for generating training data is to generate large amounts of Drell-Yan and J/ψ Monte Carlo data using Fun4All with appropriate modifications to replicate E906 data (change of target position, shielding). To achieve maximal generality, the data was generated without a trigger emulator and re-sampled to have a uniform invariant mass distribution.

The challenge for generating training data for QTracker is reproducing realistic hit matrices whose contents we can control. Most, if not all, dimuon events in the E906 and E1039 experiments also contain background hits. These come from a variety of sources, which makes them difficult to model.

Some background hits come from muons that would be of interest to us, but do not fully pass through the detector array. These are referred to as "partial tracks". They can be created by charmonium decay or the Drell-Yan process, but can also be products of other particle decays that occur in the high-radiation environment of the target and dump regions.

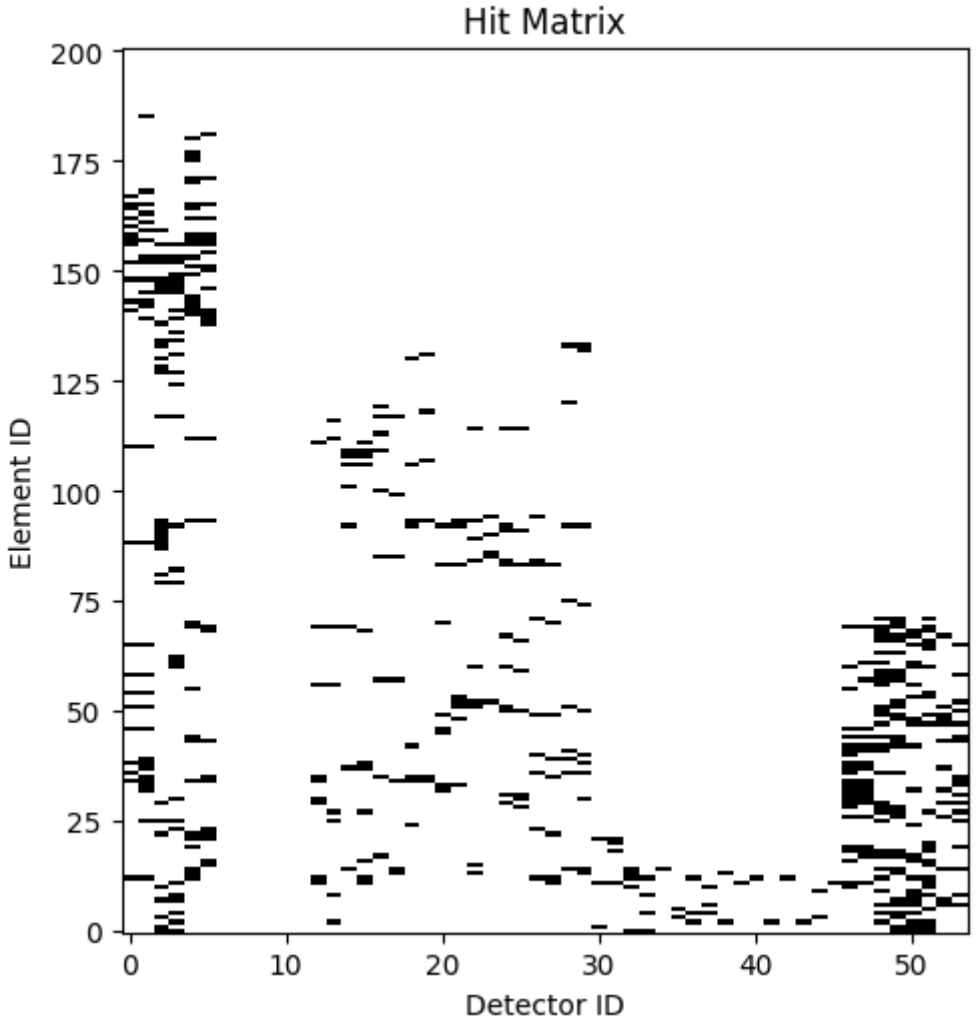


FIGURE 4.4: A hit matrix representing a real E906 event from Run 6.

Other background hits are simply random noise, such as those described in 4.1.1.1. Having accurate appoximations of these types of background hits are critical to creating representative training data.

Through the random trigger, NIM-3, we have a large selection (over 10 million) of examples of background hits and partial tracks that did not trigger the FPGA-type triggers. Although they are real background hits, the events may contain dimuons that are reconstructable by QTracker, which would result in incorrectly labeled training data.

To remedy this, we can use data from only one station per NIM3 event, which eliminates the possibility that a full muon track could be contained in the embedded data while keeping hit patterns and correlation between different hits. In

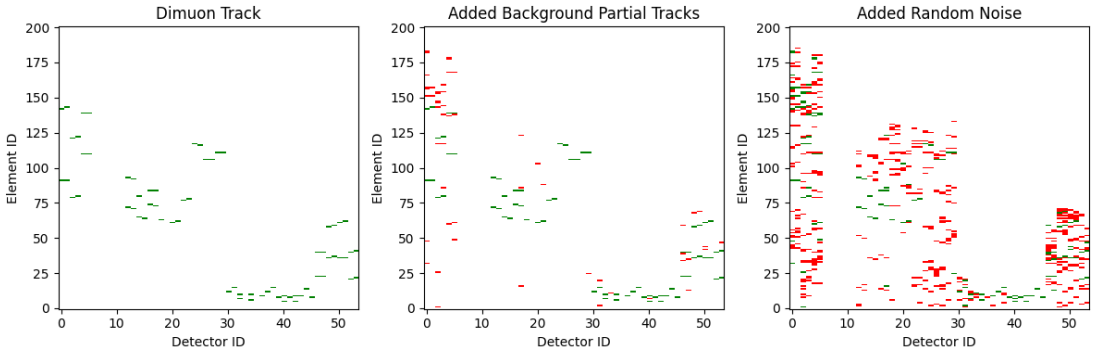


FIGURE 4.5: The process of generating training data for a single event.

order to mimic the occupancies of triggered events, we can include hits from multiple events in each station with probabilities tuned to mimic the occupancies of E906 data.

Another consideration for generating the training data was in accounting for detector efficiencies. When generating the MC, we turned off all hit realization, because for track finding we needed the detector that the particle would have hit, if the detector was 100% efficient. When inserting the MC data into the hit matrices, then, we needed to account for the real-world efficiencies of the detectors. These efficiencies have been calculated by the collaboration, and an estimate of 94% was used for training generation.

The steps we took to generate training data were as follows:

1. Create blank hit matrices for hits and drift time.
2. Place the full MC tracks on the hit matrices with probabilities based on calculated detector efficiencies.
3. Embed random noise hits.
4. Embed partial particle tracks from NIM3 events to match experimental detector occupancies.

This method is shown in Figure 4.5.

This process was slightly modified for different training configurations, which will be discussed in the training sections for each individual network.

4.2.3 QTracker Overview

The QTracker package is designed to process raw data files from the SeaQuest experiment. The reconstruction process begins by loading relevant event data from the raw files, including hit information from the spectrometer. This hit data is then organized into hit matrices, which serve as input for the deep learning models.

The core of the QTracker package consists of a series of neural networks trained on simulated SeaQuest or SpinQuest data. These networks are specialized for different tasks in the reconstruction process:

1. Event Filtering: The initial stage involves an event filter network that classifies events based on their likelihood of containing a dimuon pair. This filtering step significantly reduces the computational load by discarding events that are unlikely to be of interest.

2. Track Finding: For events that pass the filter, track finding networks are employed to identify potential muon tracks within the detector. Separate networks are used for positive and negative tracks, as well as for tracks originating from different interaction vertices (e.g., the target, the beam dump).

3. Track Reconstruction: Once tracks are identified, their kinematic properties (e.g., momentum, direction) and vertex information are reconstructed using additional neural networks. These networks are trained to predict the most likely parameters of the tracks based on the hit patterns in the detector.

4. Target/Dump Filtering: In the final stage, another filtering step is applied to distinguish between dimuon pairs originating from the target (signal events) and those from the beam dump (background events). This is achieved using a dedicated neural network that analyzes the reconstructed track and vertex information.

The QTracker package produces reconstructed event data in the form of NumPy files. These files contain a variety of information, including:

1. Reconstructed kinematic and vertex parameters for the identified muon tracks.

2. Probabilities associated with the event filter, target/dump filter, and other classification tasks.

3. Metadata such as run and event IDs, trigger information, and detector occupancies.

The reconstructed data can be further analyzed and processed to extract physics observables relevant to the SeaQuest experiment.

The QTracker package is implemented in Python and utilizes the TensorFlow deep learning framework. The neural networks are trained on simulated data, which allows for optimization of their performance and accuracy. The package is designed to be modular and extensible, making it easy to incorporate new models or modify existing ones as needed. In neural network training, it's often the case that the number of training events should be greater than the number of trainable parameters to avoid the neural network memorizing the training data, rather than learning the patterns. We employ training strategies to avoid such an issue, but future work could include training on larger amounts of data.

Testing and validation are steps in the development of any machine learning model, and QTracker is no exception. These processes ensure that the neural networks within QTracker can accurately and reliably reconstruct events from real experimental data.

4.2.4 Event Filter

The event filter is the first step of the QTracker method and aims to identify the detector trigger events that contain data that is worth analyzing. The FPGA-1 trigger requires a hit pattern in the hodoscope arrays that appear to be a dimuon pair. Because the rate of random hits in the detector array is much higher than that of true dimuon hits, it is very likely that the trigger condition being met is not, in fact, from a pair of muons, but from other processes.

It is therefore necessary to train a network that identifies which triggered events do indeed contain a dimuon pair. This is done by the use of a modified version of the AlexNet, a convolutional neural network (CNN) that operates on the principles of deep learning to interpret and classify images[69]. Its architecture comprises

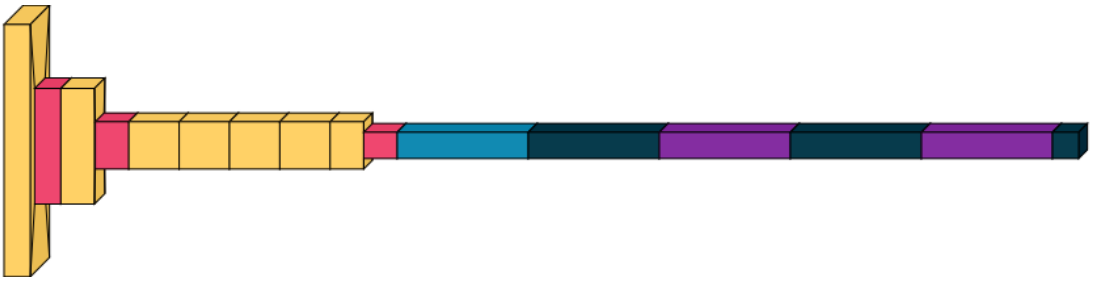


FIGURE 4.6: The infrastructure of the Event Filter. It is comprised of a series of convolutional layers (yellow) pooling layers (red), a flattening layer (blue), dense layers (black), and dropout layers (purple). Created using VisualKeras[17].

several layers designed to progressively extract and process features from input images.

The event filter receives the hit matrices as inputs. The hit matrices are then subjected to a series of convolutional layers which systematically analyze localized patterns within the hit matrices. Through the convolutional process, it identifies fundamental features, gradually discerning more complex structures and patterns in subsequent layers.

Following each convolutional layer, an activation function is applied to introduce non-linearity into the network. This activation function, a ReLU (Rectified Linear Unit), determines whether a neuron should be activated based on the weighted sum of its inputs, enhancing the network's capacity to model intricate relationships within the data.

Pooling layers are used throughout the network to downsample the feature maps generated by the convolutional layers. These layers summarize the most salient features while reducing spatial dimensions, allowing for superior computational efficiency and avoiding overfitting.

After passing through these layers, the processed features are fed into fully connected layers. These layers function as the network's decision-making core, aggregating and synthesizing the extracted features to classify the input hit matrix. Neurons in these fully connected layers are interconnected, enabling comprehensive analysis and interpretation of the feature representations.

Intermixed into the dense layers are dropout layers, which randomly set inputs to zero to pass to the following layer during training. The use of dropout layers is a commonly used strategy to prevent over-fitting a neural network.

The event filter has 27,383,238 trainable parameters. A visual representation of the event filter infrastructure is shown in figure 4.6.

The performance of the event filter is important for the overall effectiveness of QTracker. Efficient event filtering ensures that only relevant events containing valuable data are retained for further analysis. The significance of the event filter's performance lies in its ability to streamline data analysis processes by automatically categorizing events. This automation ensures that resources are focused on analyzing pertinent data, thereby improving the efficiency and efficacy of the QTracker method.

4.2.4.1 Training

Training the event filter involved creating MC events that met the trigger condition of having positive and negative muon tracks through the hodoscopes. For positively categorized events, the rest of the particle track through the drift chambers and proportional tubes were added. For negatively categorized events, only the random hits and partial tracks from NIM-3 events were added to the drift chambers and proportional tubes.

The muons and dimuons used in the training were generated using Pythia and GEANT4. They were generated as Drell-Yan dimuons with vertices within the target region with mass greater than or equal to 2 GeV, and resampled to have a flat mass spectrum from 2 GeV to 9 GeV.

Due to the size of the training data, it was not possible to perform the training in one session. As such, the training used the following strategy:

1. Generate validation data (100,000 events).
2. Generate training data (1,000,000 events).
3. Train the network on the training data, using the validation data loss as a callback.
4. Repeat steps 1, 2 and 3 until a total of 10,000,000 events have been used for training.

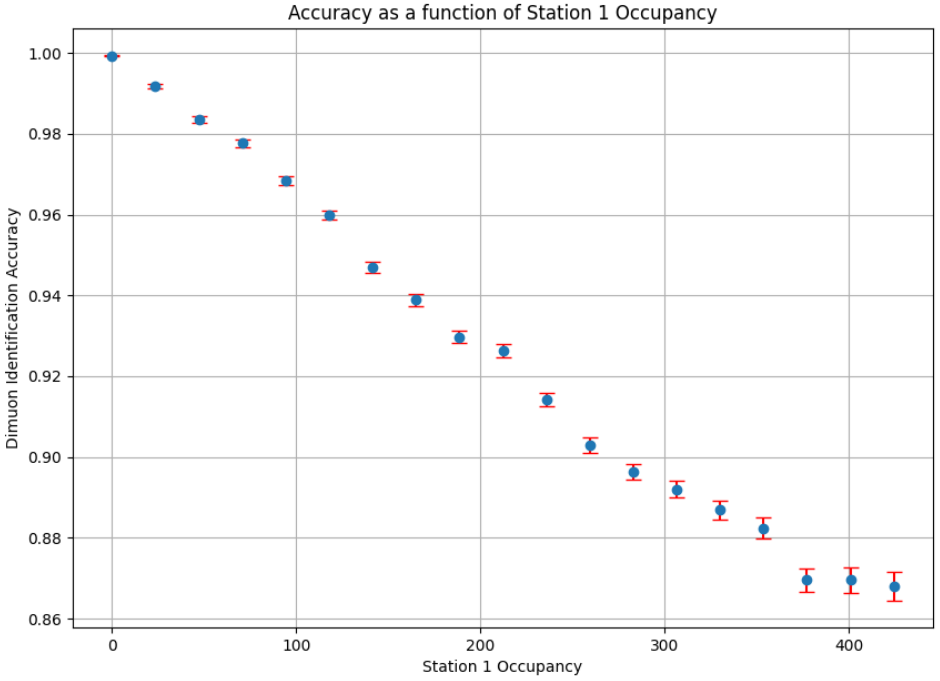


FIGURE 4.7: The accuracy of dimuon classification as a function of station 1 occupancy. Error bars are purely statistical.

This allowed us to train using a larger number of events with our limited computer resources.

The categorical cross-entropy loss function was employed, using the Adam optimizer with an initial learning rate of 1e-4. Each batch of training data had a validation loss patience of 10 epochs, with a callback mechanism to track the best epoch.

4.2.4.2 Testing

The simplest way to measure the quality of a classification is by examining the accuracy. By plotting the accuracy of the dimuon identification as a function of station 1 occupancy, we can get an idea of how well the classifier is doing. This plot is shown in figure 4.7.

For more in-depth evaluation of the performance of the event filter, we need to introduce new concepts common in classification neural networks: Precision, recall, and receiver operating characteristic (ROC) curves.

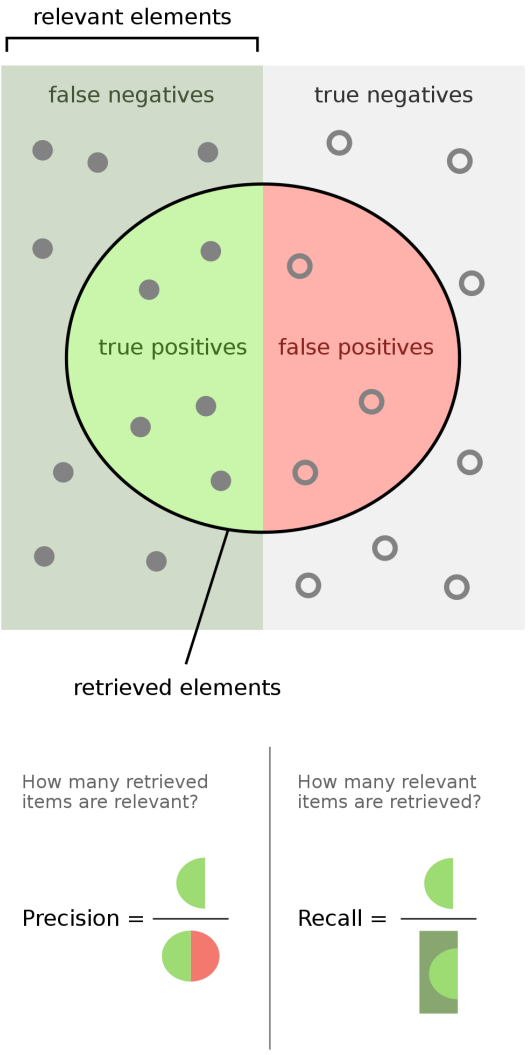


FIGURE 4.8: A visual representation of precision and recall.[18]

Precision measures the accuracy of positive predictions made by a classification model. It quantifies the proportion of positive predictions that were actually correct. Mathematically, precision is defined as the ratio of true positive predictions (correctly predicted positive instances) to the total number of positive predictions (including true positives and false positives). A visual representation of these values is shown in figure 4.8.

In words, precision can be expressed as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.1}$$

High precision indicates that when the model predicts a positive outcome, it is more likely to be correct. High precision is important for our goals because it prevents background from being included in the measured values.

Recall, also known as sensitivity or true positive rate, assesses a model's ability to correctly identify all positive instances in the dataset. It quantifies the proportion of actual positive instances that were correctly predicted as positive by the model. Mathematically, recall is defined as the ratio of true positive predictions to the total number of actual positive instances, and can be represented as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.2}$$

High recall indicates that the model effectively captures most of the positive instances present in the data. A high recall is important for our goals because it allows for higher statistics in our measured values.

Precision and recall are both influenced by the probability thresholds used to select positive events. As their values are dependent on the ratio of the positive and negative values in the data set, it is necessary to select the threshold value based on results from analyzing real data. If there is a high signal-to-noise ratio, then the value can be set relatively low to allow more true positives through, while if there is a low signal-to-noise ratio, that value must be set higher to avoid false positives overwhelming the data.

Another useful tool for classifiers is the ROC curve, a graphical tool for assessing a classifier's ability to distinguish between two classes by varying the classification threshold. In the context of multi-class classification, it can be adapted by treating one class as positive and the others as negative. The curve tracks the true positive rate as a function of the false positive rate.

The area under the curve is a good measure of the quality of the classifier. A classifier with an area of 0.5 has no predictive power (it is no better than guessing), while a classifier with an area of 1 is a perfect classifier. As shown in figure 4.10, our classifier has a ROC curve area of 0.99, which is considered excellent.

The ROC curve provides insights into the trade-offs between true positive and false positive rates, helping us select the optimal operating point or threshold for the

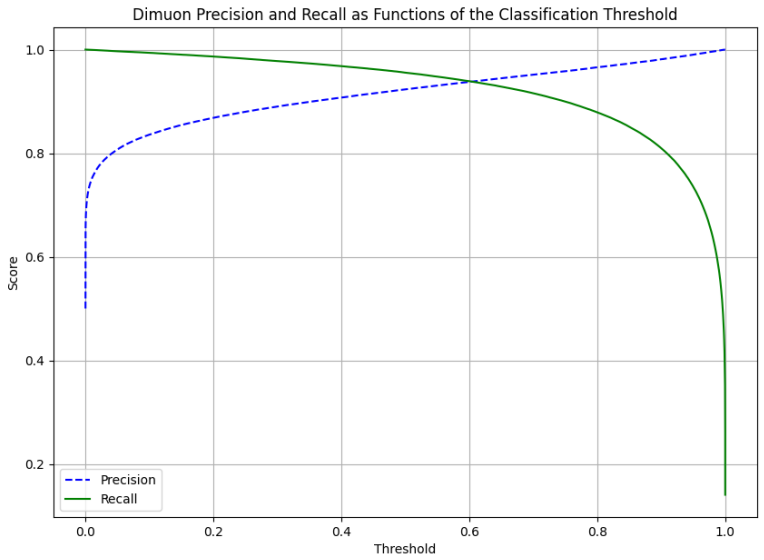


FIGURE 4.9: The precision and recall values for dimuons as a function of classification probability threshold. As threshold increases, the precision increases while the recall decreases.

classifier. This will be useful when analyzing real experimental data to determine the ideal threshold.

4.2.5 Single-Muon Track Finder

Similarly to KTracker, because it is highly likely for two randomly coincident muons to be the source of the trigger, we first want to find the tracks for the positive and negative muons passing through the detector arrays, without assuming that they were produced in the same place. This step is also done with a CNN.

Because of the geometry of the detector, each muon can pass through a maximum of 34 detectors. There are a total of 48 detectors, of which 28 lie in the same z-plane as another detector, meaning that they are mutally exclusive with one other detector. This means that not only does the track finder need to determine which hits correspond to a real track but also identify which detectors a specific muon passed through. This problem was solved by combining detectors in the output array. For hodoscopes, proportional tubes, and drift chambers in station 3, each detector has a paired detector in the same z plane. For these detectors, the track finder only outputs a single value, with a positive value corresponding to one detector, and a negative value corresponding to the other.

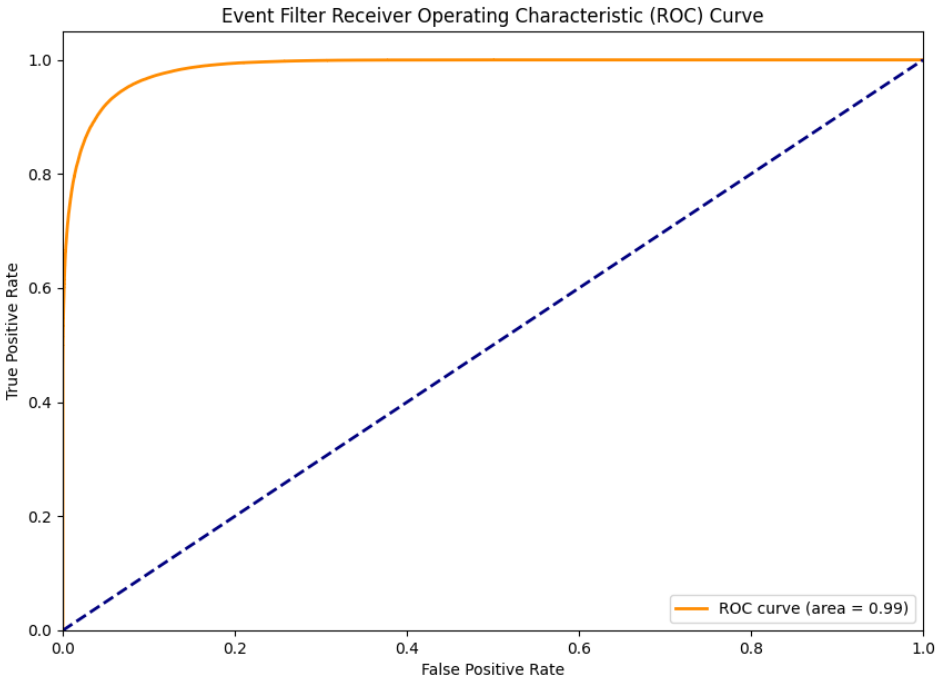


FIGURE 4.10: The ROC curve for dimuon classification on MC data mimicking FPGA-trigger events. The area under the curve is 0.99, which is considered an excellent classification score.

For instance, station three drift chambers were divided into P and M sub-stations. If a particle passed through 3P, elements 12-17 of the track were marked as positive, while if it passed through 3M, the elements were marked as negative.

This means that our muon track finding CNN must have an input shape of 54×201, and an output size of 34. To accomplish this, we use an infrastructure that is a repeating sequence of convolutions, max pooling, and relu activations. In total, there are five convoluational layers and seven dense layers. The network has over 60 million trainable parameters. The network takes in the hit matrix image and outputs two lists of the elements activated by each muon in a dimuon pair as they pass through the detector array. Figure 4.11 shows a visualization of the Track Finder network.

This neural network is designed to effectively identify and distinguish the tracks of muons passing through the detector arrays, ensuring high accuracy in tracking individual muon paths without the assumption of a common origin.

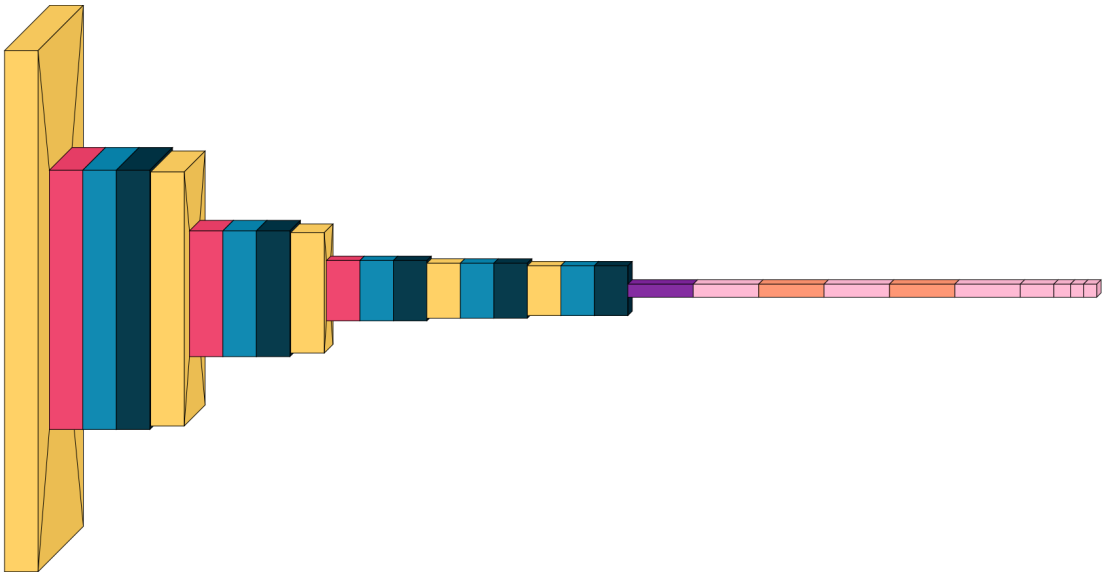


FIGURE 4.11: The infrastructure of the Track Finder networks. It is comprised of a series of convolutional layers (yellow), pooling layers (red), normalization layers (blue), activation layers (black), a flattening layer (purple), dense layers (pink), and dropout layers (orange). Created using VisualKeras[17].

4.2.5.1 Training

To train the single-muon track finders, events were generated with uncorrelated positive and negative muons injected in each event, meaning that they did not have to originate from the same vertex location. The injected muons were generated along the beamline as part of a Drell-Yan dimuon pair, then each muon was randomly selected independently.

For each event, a 2×34 array was saved, corresponding to the tracks of the positive and negative muons as they passed through the detector array. Two networks were then trained using this generated data, one of them attempting to find the positive muon track, the other attempting to find the negative muon track.

While a classification loss could have been used for the track finder, the number of possible categories (201 for station 1 U and V plane drift chambers) made it impractical. Additionally, the goal was to reward the optimizer for approaching the correct element ID, leading to the selection of a mean squared error (MSE) loss. As the most complex network, the track finder's training process took the longest time. It followed the same general process as the training of the event filter:

1. Generate validation data (100,000 events).

2. Generate training data (1,000,000 events).
3. Evaluate the validation and training data using the event filter, allowing the passing events to proceed to training of the track finder.
4. Train the network on the training data, using the validation data loss as a callback.
5. Repeat steps 1, 2 and 3 until a total of 10,000,000 events have been used for training.

10,000,000 events were used for training, using the Adam optimizer with an initial learning rate of 1e-5. Each batch of training data had a validation loss patience of 10 epochs, with a callback mechanism to select the best epoch.

4.2.5.2 Track Hit Matching

Up to this point in QTracker, the neural networks know nothing about the drift times of the drift chamber and proportional tube hits. This information will be important in momentum reconstruction and vertex finding, so it is necessary to match those hits to their timing information. This presents a small problem: although the track finder is able to identify the element ID over 95% of the time correctly, it occasionally misses the real hit. Additionally, sometimes there is no real hit, as the detector efficiency is not equal to unity.

These two truths about the system unfortunately work in opposition to each other. We can set up a piece of code that looks for the closest hit to the predicted element of the detector, but if there was not a real hit in that detector, it would identify an incorrect hit. From the true hits in other nearby detectors, the track finder is often able to determine where a hit should be in a detector, even if it did not register. This presents a balancing act: we want to search for the correct hit while knowing that it may not exist.

To balance these two desires, we can implement an algorithm that creates a search window around the predicted track, based on how precise the track finder is for that portion of the detector. This search window size was chosen based on the angular resolution of the detector, as viewed from the target position. For instance, the first x hodoscope in station zero has detector spacing of 7 cm and is 800 cm downstream of the target. This gives an angular resolution of $sin^{-1}(7\text{ cm}/800\text{ cm}) = 13$

	Drift Chamber	Proportional Tube	Hodoscope
Elements	112-201	72	16-23
Spacing (cm)	0.6-2.0	7-23	5
Angular resolution (milliradian)	1.1-1.7	2.2-2.6	7.8-13
Search Window	5	3	1

TABLE 4.2: The track finder hit-matching algorithm uses different size search windows, based on the angular resolution (in lab frame) of the detector. Drift chambers have the smallest angular resolution of the three detector types, so they have the largest search window, while horoscopes have the largest angular resolution, so the search window is only the central predicted value.

milliradians. A summary of the spacing, angular resolution, and search window size is shown in table 4.2

The hit-matching algorithm performs this search window for each detector to match the predicted track with real hits and drift information. If there is, however, no hit within the search window, the predicted element is passed on to the next step of the reconstruction, without any drift information.

In addition to the detector and drift information, the detector outputs a third value for each detector, which indicates whether the track position was a physical hit or an interpolated hit. This allows us to use that information for quality cuts, making sure that muon tracks are not hallucinated by the track finder.

With this matching, the output of the hit-matching is of shape $2 \times 34 \times 3$.

4.2.5.3 Testing

Although we trained the track finders using a root-mean-square loss, accuracy is also important for evaluating its performance. We will consider two performance issues: how accurate the track finder is on a binary scale (correctly identified element IDs) as a function of occupancy, and the average miss when the track finder is incorrect. For this testing, we have assumed a detector efficiency of 100%.

To perform the tests, we generated events with occupancies that mimic FPGA-1 occupancies with two injected muons from along the beamline. The events were generated with two independent muons of opposite signs. The reconstructed tracks were then checked for their quality (number of interpolated hits), and evaluated for accuracy and precision.

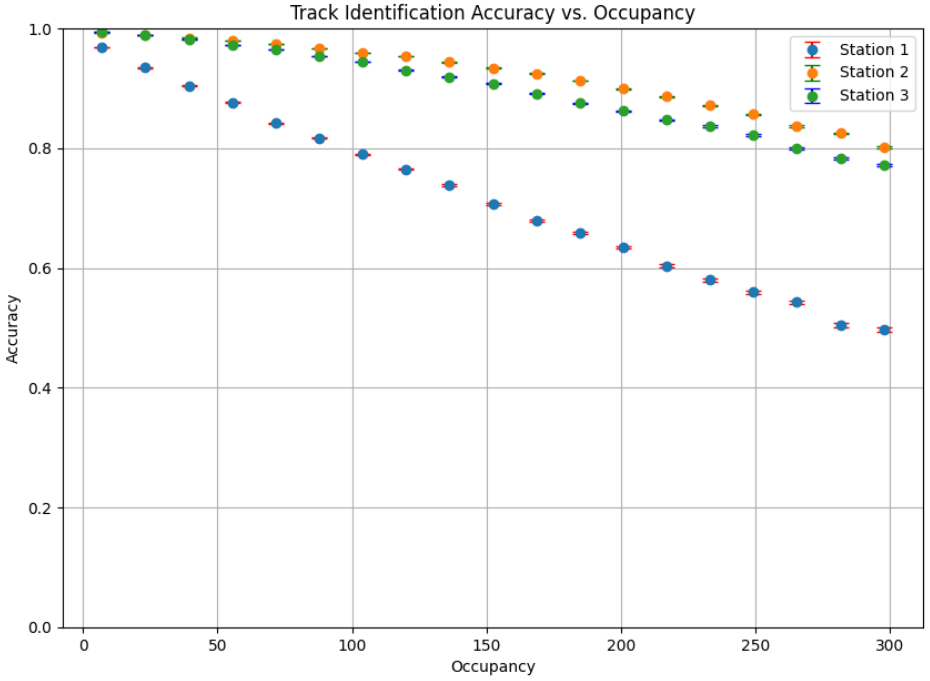


FIGURE 4.12: Track finding accuracy for the single-muon track finder as a function of occupancy for each of the three drift chamber stations. Finding is classified as accurate if the hit is within 1 element of the true track. Errors shown are statistical.

Although they are seperate networks, the positive and negative muon track finders behave extremely similarly. Therefore, for simplicity, we will combine their results for this testing.

First, we will look at the accuracy of the track finding as a binary classifier. Figure 4.12 shows the track finding accuracy for each of the three drift chamber tracking stations as a function of that station's occupancy.

Station 2 has a better accuracy than stations 1 or 3, with station 1 having the lowest accuracy of the three. This may be partially because it is placed before KMag, which affects positive and negative muons differently.

The other measure of track finding that we want to look at is the average miss of the track finding. Those plots are shown in Figure 4.13. All three types of detectors have misses of less than 1 element, on average.

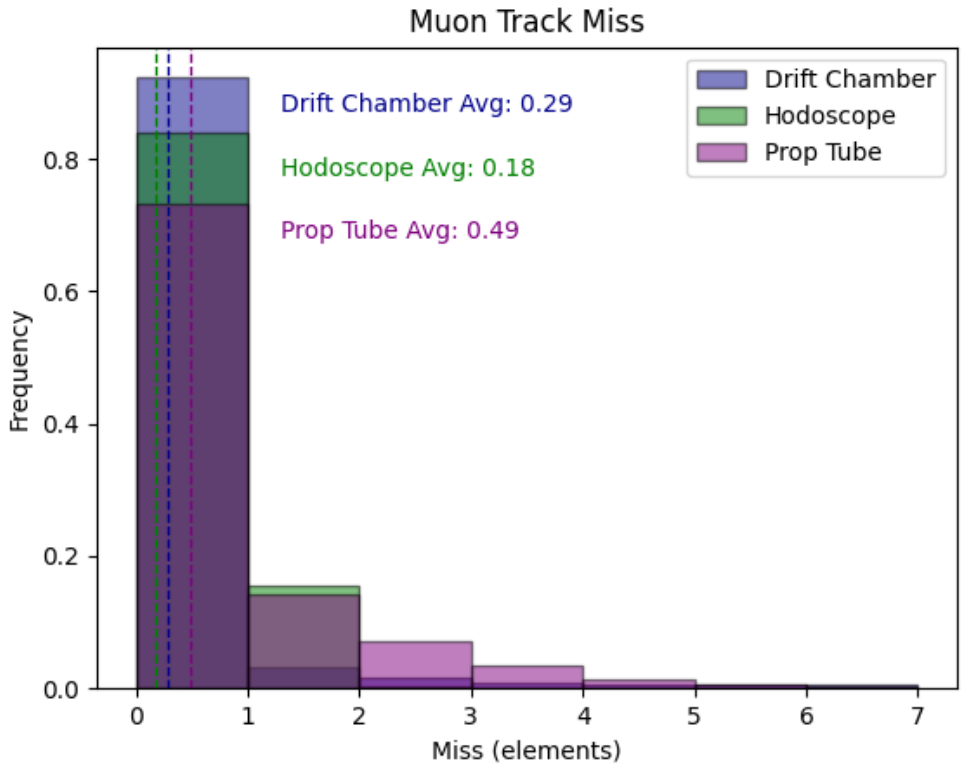


FIGURE 4.13: The average miss (in elements) for each detector type for the single-muon track finder. The average miss for the drift chambers is 0.29 elements, for the hodoscopes it is 0.18 elements, and for the proportional tubes it is 0.49 elements.

4.2.6 Single-Muon Vertex-Finder

Once we have found the tracks for each individual muon, the most important aspect for categorizing them as coincident muons or dimuons is the position of their intersection with the beamline. If the particle tracks cannot be traced back to the same region along the beamline, then we can conclude that either were not produced by the same process, or the quality of the tracks that they left are not high-quality enough to be used in analysis.

The single-muon vertex finders are feed-forward dense neural networks. They take the tracks that were found by the track finder and the hit-matching algorithm and outputs the z-position that the particle track approaches closest to the beamline. It does this by taking in the input and then passing the information through a sequence of dense layers, starting with 4096 neurons, and reducing by a factor of two down to 16, then a single-value output, corresponding to the z-position along the beamline. This correponds to 210,433 trainable parameters.

4.2.6.1 Training

The training data generation for the single-muon vertex finders networks followed a similar process to that of the single-muon track finders. The three-momenta and generation location were saved for each event, which was then used to calculate the distance of closest approach (DOCA) to the beamline, as well as the z-position at that point.

The events were then passed through the positive and negative muon track finders. The predicted tracks were then matched to the actual hits in the hit matrices and their associated drift times. A check for track quality was then performed, limiting the number of missing hits from each track to one or fewer per station.

This process produces input arrays for each network of shape 34×2. These input arrays, as well as the z-position of the muon origin, were saved into a file, and this process is repeated with more generated events until 10 million have been generated.

The vertex finding network was trained employing an MSE loss function and the Adam optimizer with a starting learning rate of 1e-5. Similar to the other networks, each batch of training data had a validation loss patience of 5 epochs, with a callback mechanism to select the best epoch.

4.2.6.2 Testing

The testing phase for the single-muon vertex-finder neural networks was conducted to validate their performance in predicting the z-position where particle tracks approach closest to the beamline. The accuracy of these predictions is critical for determining whether muons originate from the same process and ensuring high-quality data for further analysis.

The performance of the vertex-finders was assessed using the error distribution of the z-vertex predictions. Figure 4.14 illustrates the error for the predictions of the Z-vertex for single muons, where the central peak has a width of approximately 60 cm. This metric indicates the typical deviation of the predicted z-position from the actual position, reflecting the precision of the neural network in vertex reconstruction.

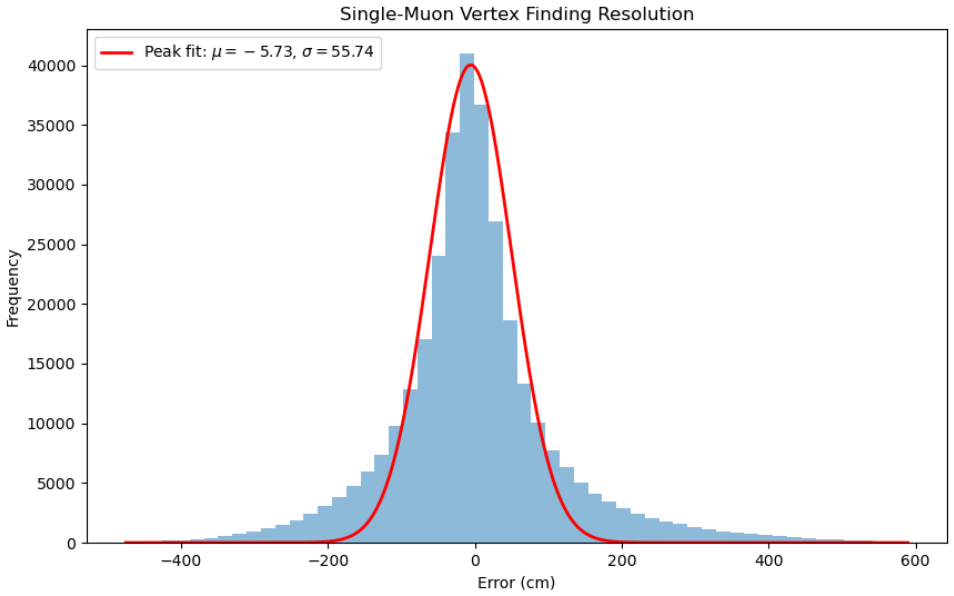


FIGURE 4.14: Error for the predictions of the Z-vertex for single muons. The central peak has width of approximately 60 cm.

The results from these tests demonstrate that the single-muon vertex-finder neural networks achieve a high level of precision in predicting the z-vertex positions. The error distribution's central peaks, with widths of 60 cm and 75 cm for single muons and dimuon separation respectively, coupled with the target-dump separation of 130 cm, indicate that they can differentiate between a dimuon pair and two muons originating at different locations.

4.2.7 Dimuon Track Finder

Once we have established that the triggered event has a high likelihood of containing a true dimuon pair, it is useful to refine the track found by the neural network, based on the assumption that the muons share a common vertex. For that purpose, we employ various dimuon track finders. These networks also employ CNNs, based on the LeNet infrastructure[70].

Similarly to the muon track finders, the dimuon track finder CNNs must have an input shape of 54×201, but since they are finding the track of two dimuons, they have an output dimension of 2×34. To accomplish this, we use an infrastructure that is a repeating sequence of convolutions, max pooling, and relu activations.

In total, the network has over 60 million trainable parameters and five convolutional layers. The network takes in the hit matrix image and outputs two lists of the elements activated by each muon in a dimuon pair as they pass through the detector array.

There are three versions of the track finder, with decreasing scopes but increasing accuracy. The first version makes no assumptions about the vertex position, the second version assumes that the vertex is along the beamline, and the final version assumes that the x, y, and z-positions are all in the target.

4.2.7.1 Training

To train the track finder, events were generated with a dimuon pair originating from the relevant vertex area. For each event, a 2×34 array was saved, corresponding to the tracks of the positive and negative muons as they passed through the detector array. The generated events were required to pass the event filter with a threshold of 75%, and the muon track finder and reconstruction was required to

Four versions of the track finder were trained on different vertex distributions:

1. All vertices along the beamline within 1 meter of the beam.
2. All z-vertices along the beamline.
3. Target vertices.
4. Dump vertices.

These versions allowed for recursive utilization of the track finder, increasing specificity and accuracy. During the initial pass, the track finder made no assumptions about the vertex position. The kinematic reconstruction and vertex-finding networks were subsequently employed to provide an estimate of the vertex position. If the vertex fell within an acceptable range, the event was forwarded to either the z-vertex finder or the xy-vertex finder, and the process iterated. If the vertex remained within an acceptable range, it was finally sent to the target vertex finder, enabling high-precision reconstruction of the kinematic variables.

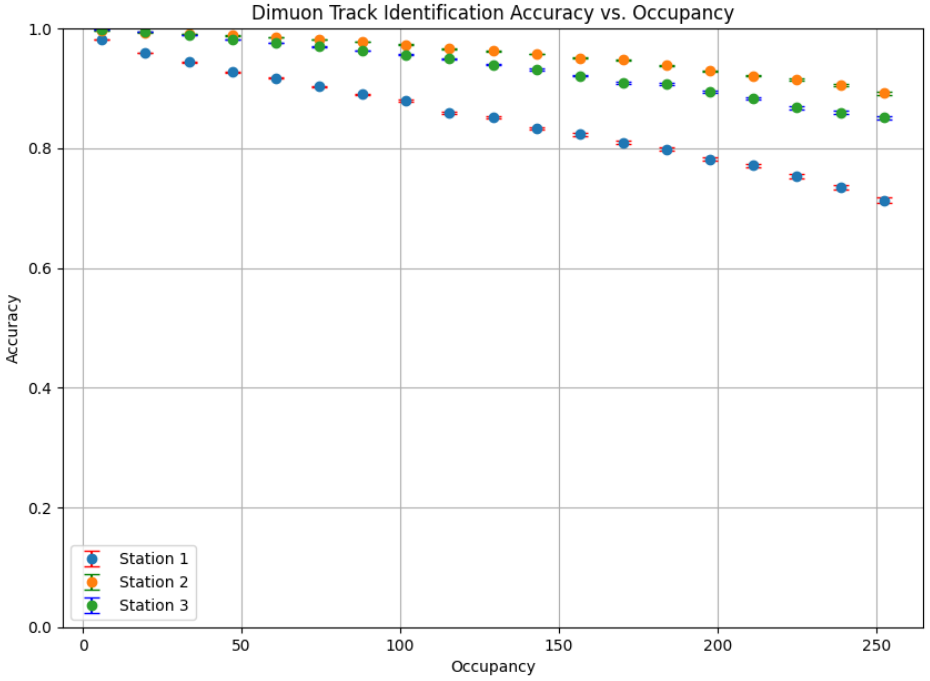


FIGURE 4.15: Track finding accuracy for the target vertex dimuon track finder as a function of occupancy for each of the three drift chamber stations. Finding is classified as accurate if the hit is within 1 element of the true track.

4.2.7.2 Testing

We performed the same tests on the dimuon track finders as were performed on the single-muon track finders. As may be expected, the decreased parameter space of possible outputs, as well as the requirement to pass single-muon track finding cuts improved the performance of the finders substantially.

To perform the tests, we generated dimuon events with occupancies that mimic FPGA-1 occupancies. The events were generated with a dimuon pair from the appropriate origin vertex. The reconstructed tracks were then checked for their quality (number of interpolated hits), and evaluated for accuracy and precision.

Although seperate tests were done for each version of the track finder, we will only show the tests for the target vertex finder to avoid repetition.

Figure 4.15 shows the track finding accuracy for each of the three drift chamber tracking stations as a function of that station's occupancy.

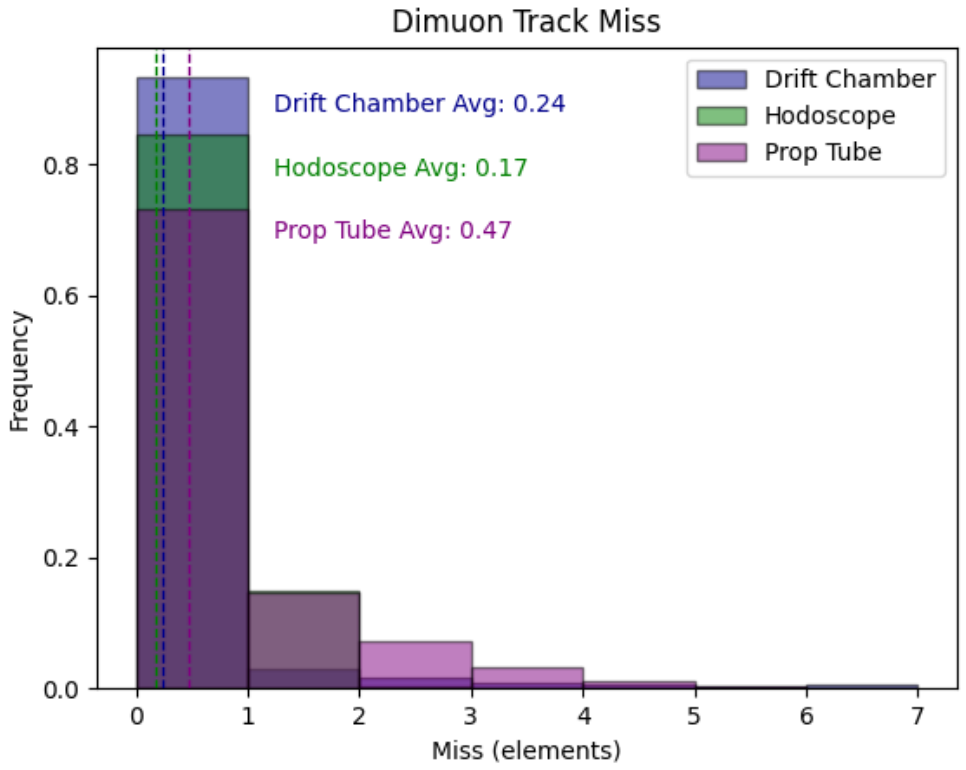


FIGURE 4.16: The average miss (in elements) for each detector type for the target vertex dimuon track finder. The average miss is for drift chambers is 0.24 elements, for hodoscopes it is 0.17 elements, and for proportional tubes it is 0.47 elements.

As with the single-muon finders, station 2 has a better accuracy than stations 1 or 3, with station 1 having the lowest accuracy of the three.

Again, we also want to look at is the average miss of the track finding. Those plots are shown in Figures 4.16. All three types of detectors have misses of less than 1 element, on average.

4.2.8 Momentum Reconstruction

The majority of the work of QTracker has been finished up to this point, and the remaining task is to translate the detector information into particle characteristics. The momentum reconstruction is actually the simplest of the DNNs that we use in QTracker.

The momentum reconstruction is a simple feed-forward dense neural network. It takes in the tracks that were found by the track finder and the hit=-matching

algorithm and outputs the three-momentum (p_x, p_y, and p_z) for each of the muons in the dimuon pair. It does this by taking in the $2 \times 34 \times 2$ input and then passing the information through a sequence of dense layers, starting with 4096 neurons, and reducing by a factor of two down to 16, then a six-value output, corresponding to the three-momenta of the reconstructed muons. The dimuon reconstruction networks have 245,334 trainable parameters.

There are three versions of the kinematic reconstruction, with decreasing scopes but increasing accuracy. The first version makes no assumptions about the vertex position, the second version assumes that the vertex z-position is in the target region, the third version assumes that the vertex is along the beamline, and the final version assumes that the x, y, and z-positions are all in the target.

4.2.8.1 Training

The training data generation for the momentum reconstruction networks followed a similar process to that of the track finder. However, instead of saving track information, the x, y, and z momenta of the positive and negative muons injected were saved, resulting in a 6-element array: $[p_x^+, p_y^+, p_z^+, p_x^-, p_y^-, p_z^-]$.

The events were filtered through the event filter with the same 75% threshold and the muon track finder cuts, before having their dimuon tracks found by the pre-trained track finder appropriate for their vertex configuration. The predicted tracks were then matched to the actual hits in the hit matrices and their associated drift times. This process produced a $2 \times 34 \times 2$ input for the momentum reconstruction.

The momentum reconstruction network was again trained with 10 million training events, employing an MSE loss function and the Adam optimizer with a starting learning rate of 1e-5. Because the size of the inputs for these networks are much smaller than those of the event filter and track finders, all training data was pre-generated, rather than using a generator while training.

Again, four versions of this reconstruction were trained on different vertex distributions:

1. All vertices along the beamline within 1 meter of the beam.

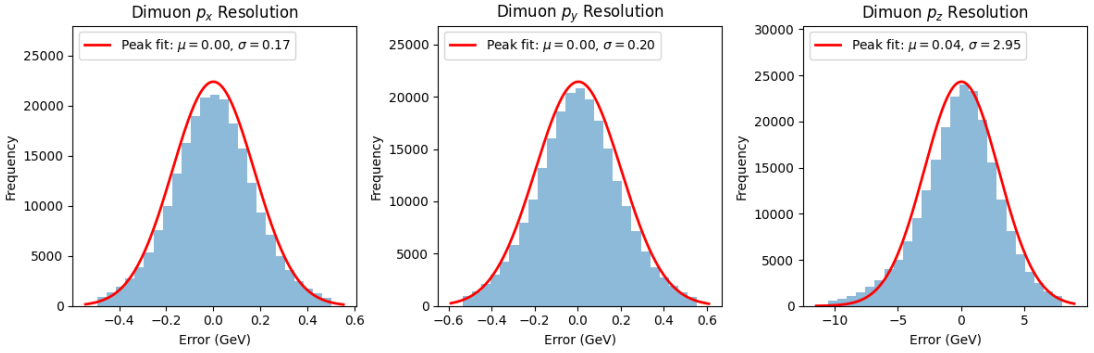


FIGURE 4.17: The precision of momentum reconstruction for target region dimuons in p_x, p_y, and p_z.

2. All z-vertices along the beamline.

3. Target vertices.

4. Dump vertices.

4.2.8.2 Testing

We are interested in examining the precision of the reconstruction of the momentum of dimuons. In order to do this, we generated MC events with embedded dimuons from the target, using smeared drift distances and appropriate detector efficiencies.

In addition, we evaluate the precision of key kinematic parameters, including x_1, x_2, x_F, M, p_T, $\cos(\theta)$, and ϕ, which are vital for characterizing the particles' behavior and interactions within the event. The precision of these variables is essential for drawing meaningful conclusions about the physics processes under investigation.

First, we will look at the resolution for p_x, p_y, and p_z with typical FPGA-1 trigger events. Figure 4.17 shows these plots, along with a Gaussian fit to the central peak of the error.

Next, we are interested in looking at the precision of the kinematic variables x_1, x_2, M, p_T, $\cos(\theta)$, and ϕ. These variables are the most important for the analysis of Drell-Yan scattering, as the SeaQuest ans SpinQuest experiments were designed to measure the angular dependence of the differential cross-section in bins of the x, M,

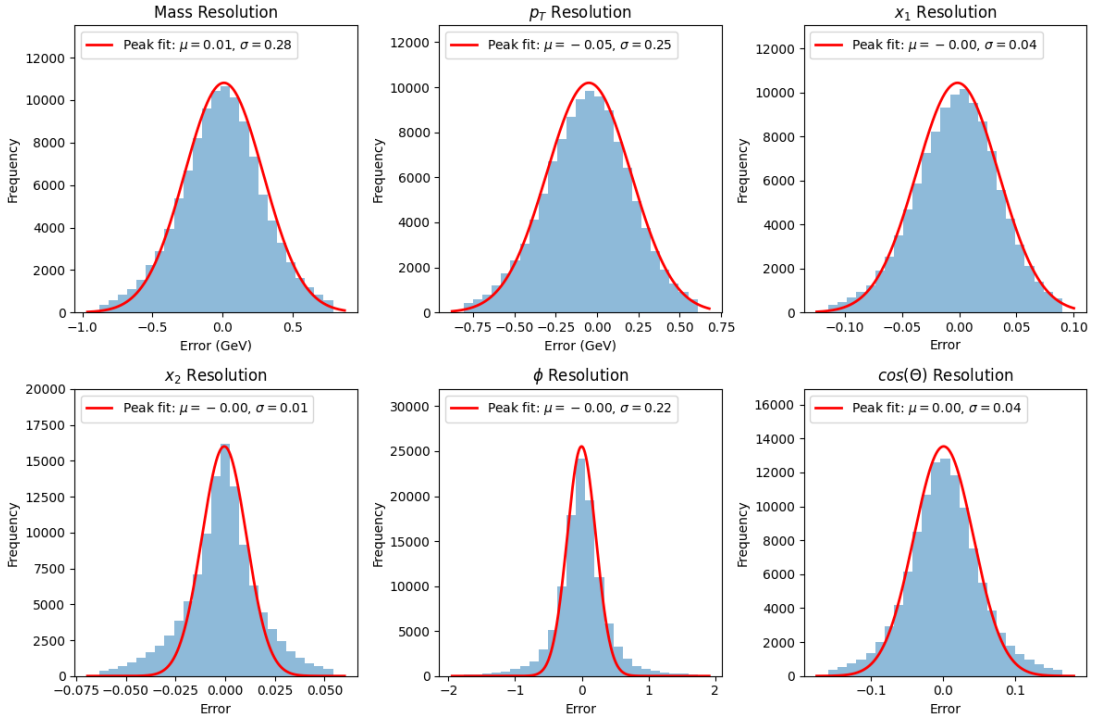


FIGURE 4.18: Error for the derived values important to physics analysis in E906 and E1039.

and p_T kinematic variables. Figure 4.18 shows the precision of each reconstructed value, along with a Gaussian fit to the central peak of the errors.

4.2.9 Vertex Finding

The vertex finding network, though similar in structure to the momentum reconstruction network, processes tracks of muons and their reconstructed momenta to predict the location that the dimuon originated. The network input has a shape of 2×71. This information is then fed through a network nearly identical to the one used for momentum reconstruction. The vertex finding networks have 248,355 trainable parameters.

There are two versions of the vertex finder. One version outputs the x-, y-, and z-positions of the dimuon vertex, and one outputs the z-position, assuming that the dimuon originated along the beamline.

4.2.9.1 Training

The training data generation for the vertex finder was nearly identical to that for the moomentum reconstruction, only replacing the output of six momentum values with three position values, v_x, v_y, and v_z.

The events were filtered through the event filter with the same 75% threshold and the muon track finder cuts, before having their dimuon tracks found by the pre-trained track finder appropriate for their vertex configuration. The predicted tracks were then matched to the actual hits in the hit matrices and their associated drift times. Their momentum was reconsructed using the appropriate momentum reconstruction network, which was then combined with the track information for the input of the vertex finder. This process produced a 2×71 input.

The momentum reconstruction network was again trained with 10 million training events, employing an MSE loss function and the Adam optimizer with a starting learning rate of 1e-5. Because the size of the inputs for these networks are much smaller than those of the event filter and track finders, all training data was pre-generated, rather than using a generator while training. The vertex-finding network was again trained with 10 million training events, employing an MSE loss function and the Adam optimizer with a starting learning rate of 1e-5.

There were two versions of the vertex-finder trained, those being:

1. All vertices along the beamline within 1 meter of the beam.
2. All z-vertices along the beamline.

4.2.9.2 Testing

As discussed when testing the individual muon resolution, it is very important to accurately reconstruct the vertices of the particle tracks in order to isolate the target events from the dump events. As such, we want to use all of the information that we have at our disposal, so we combine the predictions of the two vertex finders, as well as those of the single-muon vertex finders to provide more precise results.

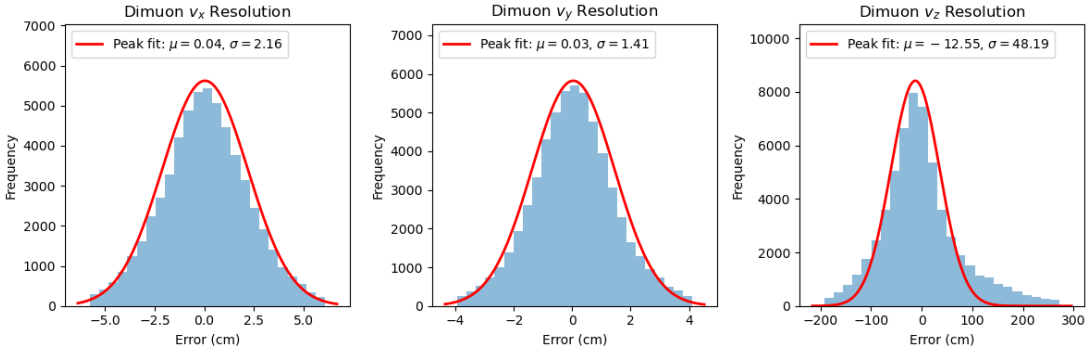


FIGURE 4.19: Vertex finding error for dimuons along the beamline for the v_x, v_y, and v_z values.

Although the z-vertex is the most important for isolating target dimuons from dump dimuons, the x and y vertices also play an important role for event selection. The probability of a dimuon event being produced far from the beamline is very small, which means that dimuons identified as such are very likely to be a coincident pair of muons masquerading as a dimuon pair.

In order to determine the resolution of the vertex finding, we generated dimuon pairs from along the entire beamline, and reconstructed the events. The results for each version of vertex reconstruction were combined to produce results for v_x, v_y, and v_z, shown in Figure 4.19.

4.2.10 Target Dump Filter

Although the vertex reconstruction is quite precise, our testing shows that there are some dimuons from the dump that could trick the vertex reconstruction to be classified as target dimuons. As such, we need to use a last neural network to classify events as being either from the target or the dump. Although this will not have more information than the vertex reconstruction, the information being output is slightly different. While the vertex reconstruction must output a single precise value for the vertex position, a binary filter is able to output a probability, allowing for a more nuanced classification.

The target-dump filter is a feed-forward dense neural network that takes the reconstructed momenta and vertex positions from all of the momentum reconstruction and vertex finding network, along with the tracks output by the track finders, and outputs a probability that the event originated in the target.

4.2.10.1 Training

In order to train the target-dump filter, a total of 5,000,000 events from both the target-vertex and dump-vertex dimuons were reconstructed, saving the output tracks, momenta, and vertex information. Both sets of dimuons were resampled to have a flat mass distribution, and appropriate event filter and track quality cuts were made.

Once the 10,000,000 reconstructed events were produced, the network was trained using those events, with a categorical cross-entropy loss function and the Adam optimizer starting with a learning rate of 1e-4. This was done for a maximum of 10,000 epochs, with a patience of 100 on the validation loss.

4.2.10.2 Testing

The performance of the target-dump filter neural network was evaluated on a testing dataset comprising both target and dump data, generated similarly to the training and validation data used in training. The overall accuracy is 89%, indicating strong classification.

The Receiver Operating Characteristic (ROC) curve was plotted in Figure 4.20 to visualize the trade-off between the true positive rate (TPR) and false positive rate (FPR). The Area Under the Curve (AUC) was calculated to be 0.96.

The precision and recall as functions of the probability threshold were plotted to further analyze the model's performance:

The model demonstrates strong performance in distinguishing between target and dump data, with an overall accuracy of 89%. The high precision and recall for both classes indicate that the model is both precise and sensitive in its predictions. The F1-scores for both classes are close to 0.90, suggesting a good balance between precision and recall.

The ROC curve further supports the model's robustness, with an AUC of 0.94 indicating excellent discriminative ability. The precision-recall curve shows how precision and recall vary with different probability thresholds, providing insights into the trade-offs between these metrics at various decision points.

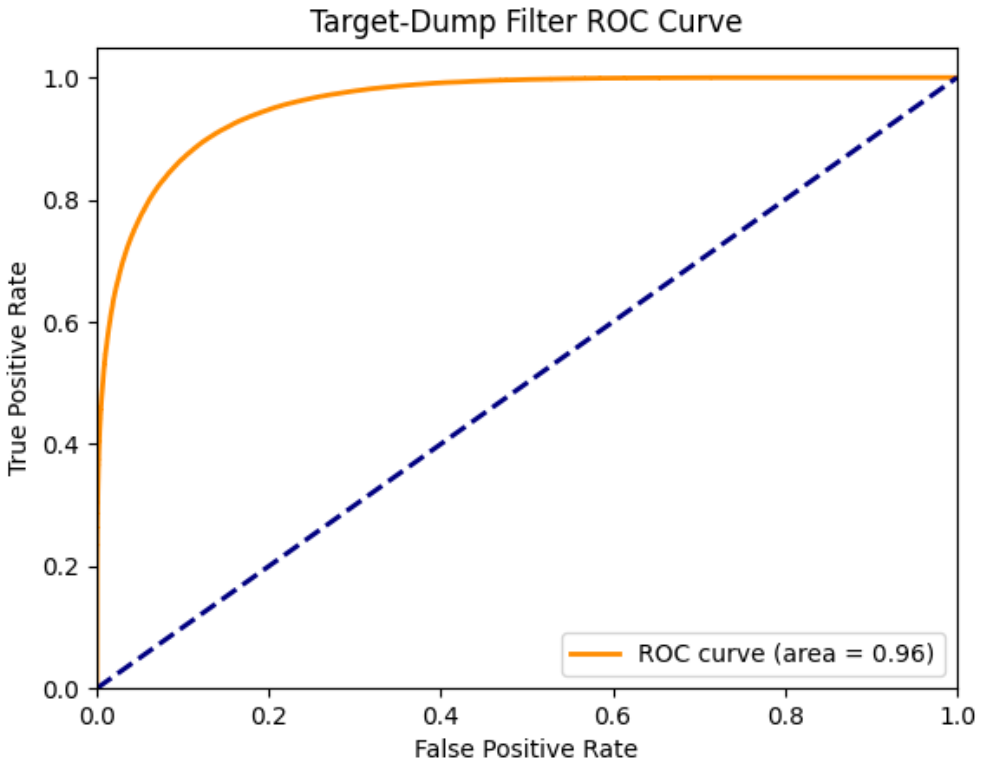


FIGURE 4.20: ROC Curve for Target-Dump Filter Neural Network

4.2.11 Processing Speed

Although less important for offline analysis than for online monitoring, the speed of QTracker is an important consideration when comparing it to KTracker. QTracker and KTracker were each tasked with reconstructing 10,000 dimuon pair events. KTracker completed the reconstruction in 23.5 hours, while QTracker was able to reconstruct the events in approximatley 30 seconds. This speed increase is so drastic, it required repeat tests to ensure that the results were not in error. In 100 tests, KTracker reconstructed the set of events in between 20 and 28 hours 90 times, while QTracker never exceeded 1 minute.

To test QTracker's ability to be integrated into an online monitoring system, a test was performed to reconstruct a spill of data. 500 spills from E906 were randomly selected for reconstruction, and the time to complete each was recorded. This was done using two methods, one with all of the code's libaries and functions pre-loaded, and one where they needed to be loaded each time a new file was evaluated.

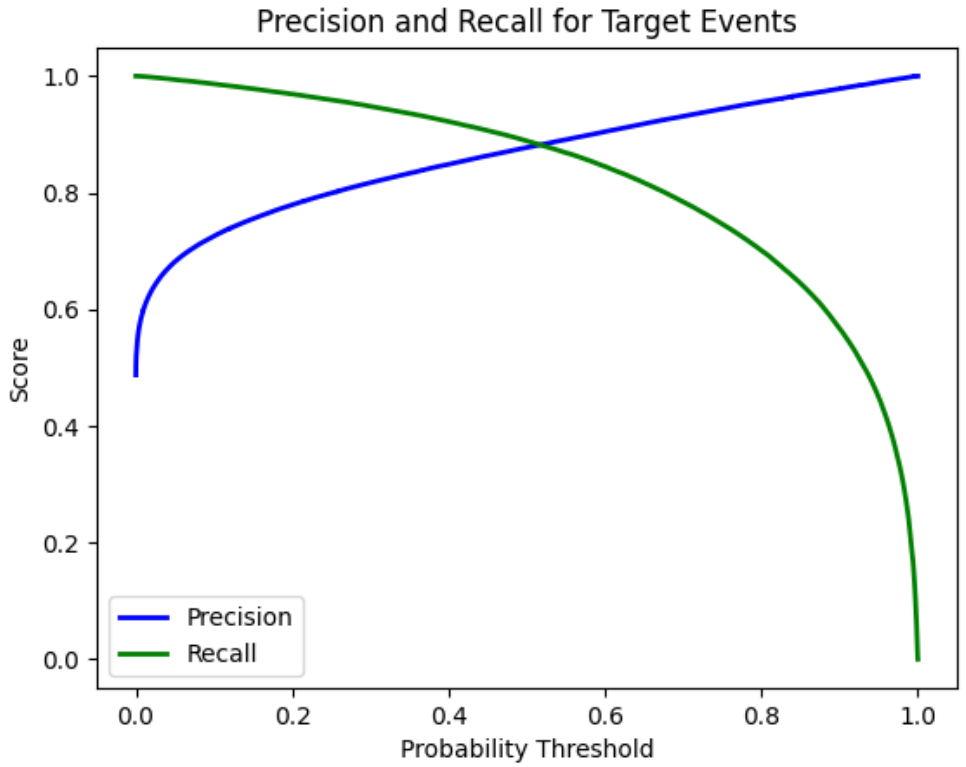


FIGURE 4.21: Precision and Recall as a Function of Probability Threshold

In both methods, the average time was comfortably under the 54-second threshold, which is defined as the time between spills from the main injector beam. The average time without function loading was 13.24 seconds, while the average time with function loading was 38.42 seconds.

This analysis shows that QTracker adapted for E1039 will be able to serve superbly as an online reconstruction algorithm, as it will be able to reconstruct the particle tracks for an entire spill between spills. This will help to improve data quality control and enable quicker turnaround for analysis of the observed physics.

4.3 Comparison to KTracker

In this section, we will compare the performance of QTracker to that of KTracker. We will examine the precision and accuracy of the two methods for vertex, momentum, and derived variable reconstruction.

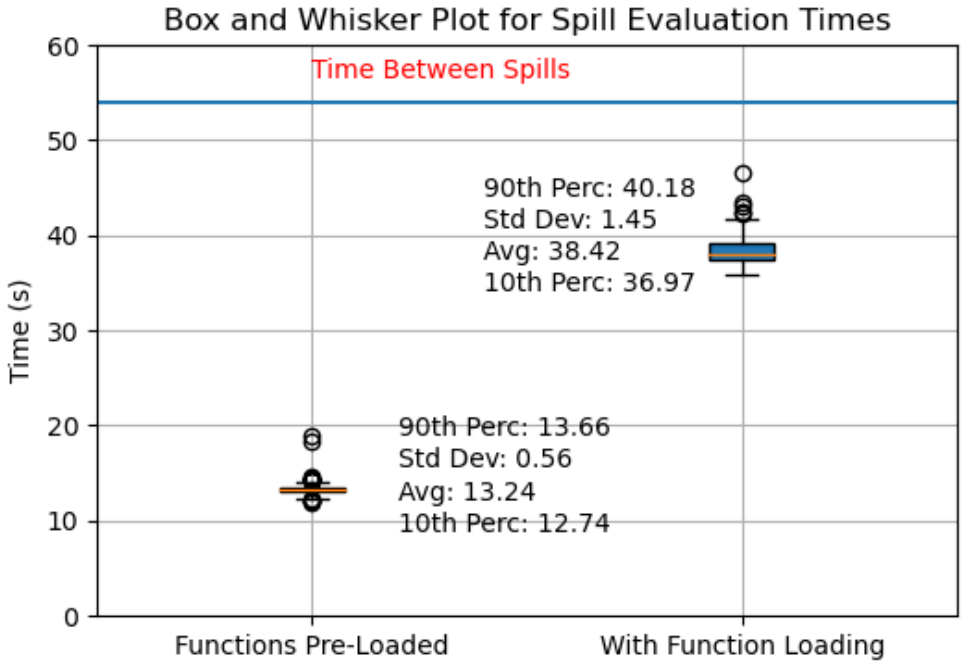


FIGURE 4.22: The time for QTracker to evaluate a single spill of SeaQuest E906 data, both with functions pre-loaded and with loading functions.

In order to compare the two methods, we will examine the precision of the reconstruction of the momentum of dimuons produced at the target, as well as the resolution of vertex reconstruction along the beamline. The precision of the reconstruction of the kinematic variables x_1 , x_2 , M, p_T, $\cos(\theta)$, and ϕ will also be examined.

Variable	KTracker σ	QTracker σ
p_x	0.30 GeV/c	0.19GeV/c
p_y	0.28 GeV/c	0.20 GeV/c
p_z	1.24 GeV/c	2.74 GeV/c
v_x	2.05 cm	2.16 cm
v_y	2.09 cm	1.41 cm
v_z	30.7 cm	48.2 cm
M	0.47 GeV	0.31 GeV
p_T	0.39 GeV/c	0.29 GeV/c
x_1	0.02	0.04
x_2	0.03	0.01
ϕ	0.33	0.28
$\cos(\theta)$	0.08	0.04

TABLE 4.3: Comparison of QTracker and KTracker variable resolution.

To run this comparison, we generated half a million Monte Carlo-based events embedded with background hits. These events contained dimuons originating along the beamline. Of those, approximately 40,000 were generated within the target region. All events were then reconstructed in both QTracker and KTracker, and their results analyzed.

The results of this comparison are shown in Table 4.3, with plots in Figure 4.23 and Figure 4.24. QTracker is able to improve on KTracker's performances for most variables, with the exception of those involving the z momenta and z vertex of particles (p_z, v_z, and x_1). Both KTracker and QTracker appear to exhibit certain biases in predictions, with KTracker having a bias in the Mass reconstruction and QTracker having a bias in the p_T reconstruction.

It is unclear why this should be the case, as QTracker is able to out-perform KTracker in the other dimensions, allowing for nearly double the resolution in the mass and $\cos(\theta)$ reconstruction. Future work will address these discrepancies.

Interestingly, as can be seen in the p_z and x_1 plots, KTracker has a narrower peak for the error, but much fatter tails, which are not taken into account in the Gaussian fit. When purely calculating the standard deviation of p_z error, we find that the standard deviation for KTracker error is 5.9 GeV, while for QTracker it is only 4.2 GeV. To use a baseball metaphor, KTracker is swinging for the fences, while QTracker is hitting for average.

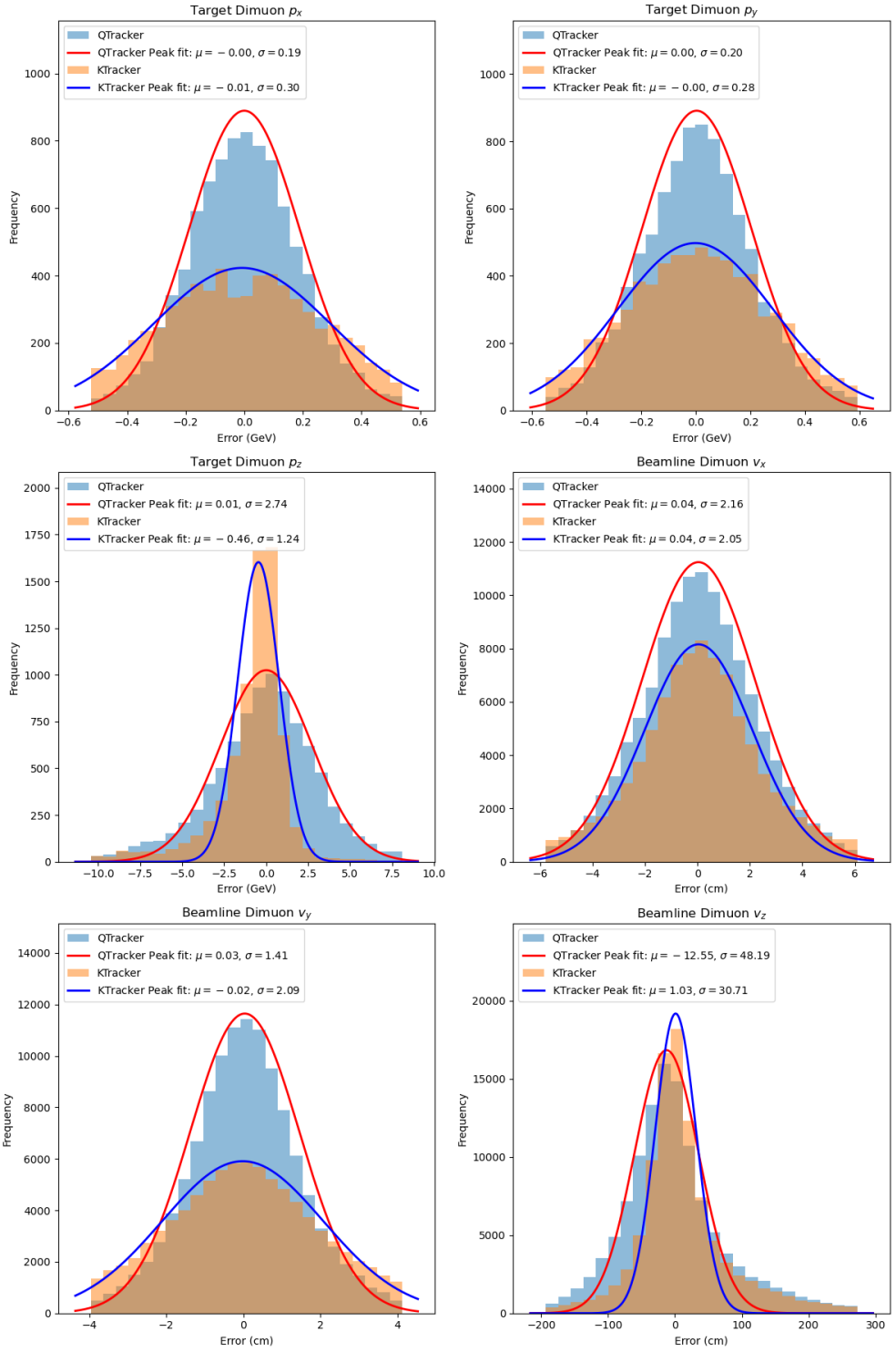


FIGURE 4.23: Momentum and vertex reconstruction precision for KTracker vs. QTracker.

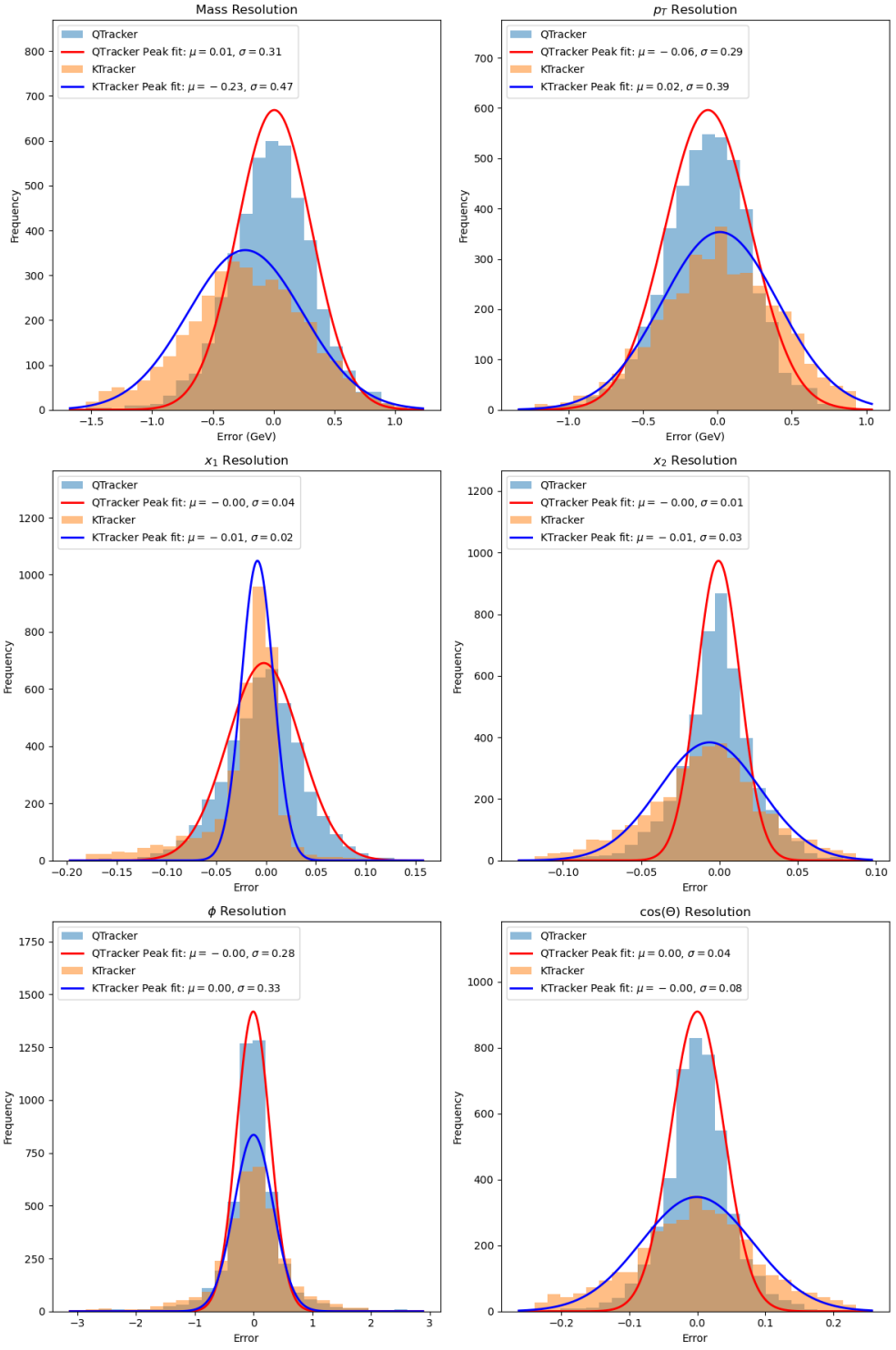


FIGURE 4.24: Precision of derived variables for KTracker vs. QTracker.

Chapter 5

Data Analysis

The QTracker neural network was trained on simulated SeaQuest data, which provided a controlled environment for optimizing the model architecture and hyperparameters. However, simulated data can only approximate the complexities of real experimental data. Therefore, extensive testing was conducted on real data collected by the SeaQuest spectrometer. This testing included using E906 data to reconstruct the angular dependence of the Drell-Yan dimuons originating from the iron dump of the experiment. The choice to analyze dump dimuons rather than target dimuons was motivated by two factors: a higher statistical level (the iron dump produced many more interactions than the liquid helium and deuterium targets), and simpler analysis due to reduced complexity of background selection. This analysis should not be understood as complete, but as a demonstration of the utility of QTracker.

5.1 Single-Event Reconstruction Path

To better understand the process of event reconstruction, we will follow the path of a single event through the reconstruction sequence.

We will follow Event 630753 from Run 26530, Spill 1301983. It is a relatively low occupancy even, allow us to better view the process on hit matrices. It starts with 549 hits across the entire spectrometer.

The saved detector information in SeaQuest data contain many hits that, for one reason or another, are not worth analyzing to find dimuon tracks. In general, those

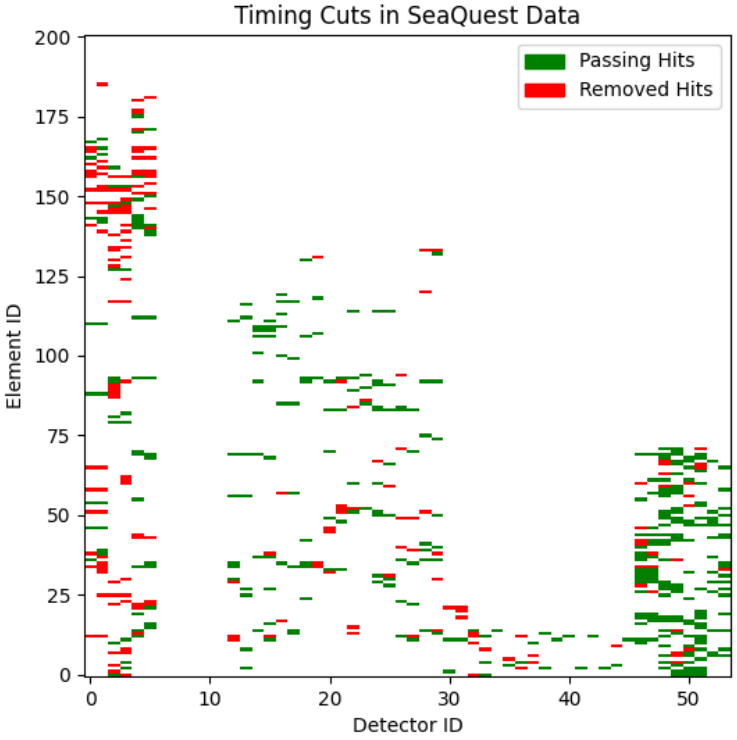


FIGURE 5.1: An example of an E906 event with timing cuts enforced. The timing cuts removed 34% of the detector hits.

irrelevant hits can be categorized into two groups: out-of-time hits and cluster hits. On average, this removes around 20% of detector hits, but can remove many more, depending on the event in question.

5.1.1 Timing Cuts

Out-of-time hits are hits that were within the TDC time window to be saved to an event, but are unlikely to have been produced by the same particles that set off the FPGA-1 trigger. Fortunately for the purposes of this analysis, the SRawEvent files that we analyze contain pre-configured timing cuts that allow us to remove out-of-time detector hits.

This removal has the effect of decreasing the complexity of the event, removing on average around 100 out-of time hits. Timing cuts are shown in Figure 5.1 for our example event.

5.1.2 Declustering

Even after applying timing cuts, there are still often obviously irrelevant detector hits present in the data. A 'declusterize' function focuses on refining the hit matrices by removing spurious hits, which often result from electronic noise or other artifacts. This function is important for enhancing the accuracy of track finding and reconstruction by ensuring that only genuine hits are considered.

The function scans the hits matrix to identify clusters of adjacent hits. For clusters consisting of two hits, the function performs an edge hit check, comparing their drift times; if one hit has a significantly higher drift time, it is removed. Additionally, the function conducts an electronic noise check for clusters where the difference in TDC times between adjacent hits is below a threshold (8 nanoseconds), removing such hits as noise. For larger clusters, the function calculates the mean difference in TDC times across the cluster and removes the entire cluster if this mean difference is below a set threshold (10 nanoseconds). This comprehensive hit removal process ensures that the hits, drift, and tdc matrices are cleansed of spurious data.

The declusterize function effectively removes noise and spurious hits from the hits, drift, and tdc matrices, resulting in refined data that is more reliable for track finding and reconstruction. This refined hit data significantly improves the accuracy of the subsequent reconstruction algorithms, ensuring that the final results are based on genuine particle interactions.

To illustrate this, Figure 5.2 shows our example event with the declusterize function performed on it. Additionally, Figure 5.3 shows the combined effects of these two processes on that event.

5.1.3 Dimuon Identification and Muon Finding

The hit-removed event is fed into the Event Filter neural network, which identifies the probability that this event contains a dimuon pair. Because of the low occupancy (and the fact that we have selected an event that will pass), the filter predicts a 99.99% chance that this event contains a dimuon.

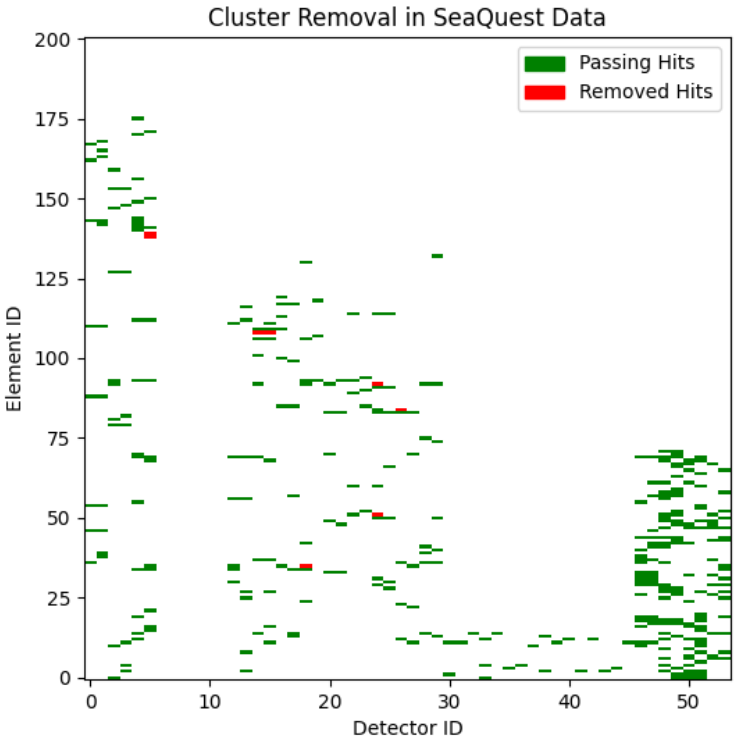


FIGURE 5.2: An example of an E906 event that has been 'declustered'. Declusterization removed an additional 2% of detector hits.

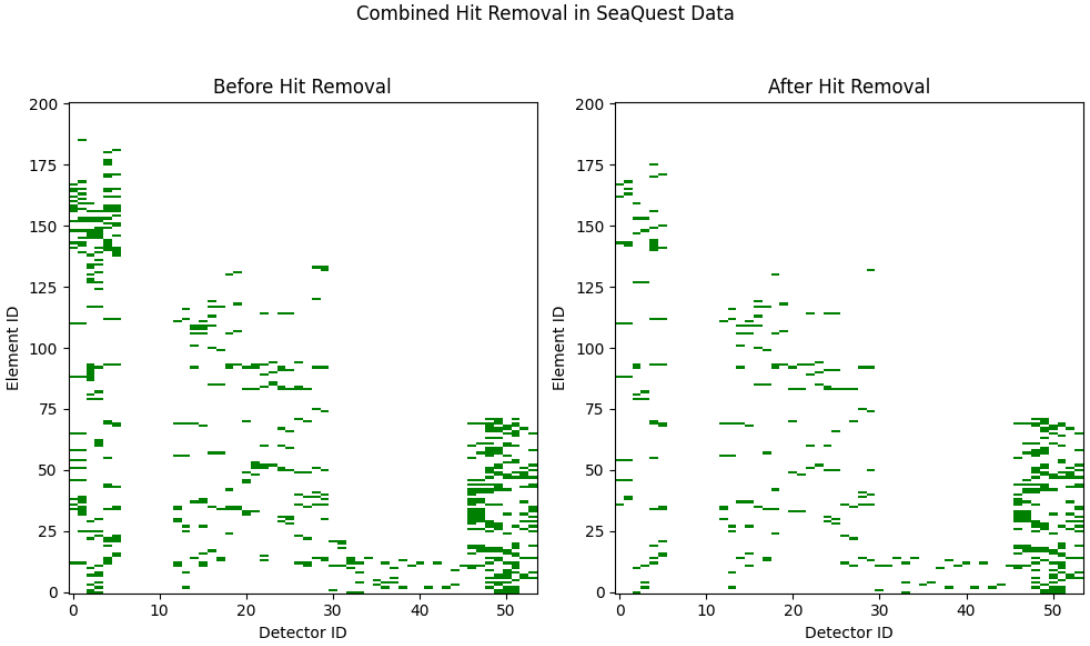


FIGURE 5.3: A comparison of an E906 event before and after hit removal. In this example, 196 hits were removed, reducing the total spectrometer occupancy by 36%.

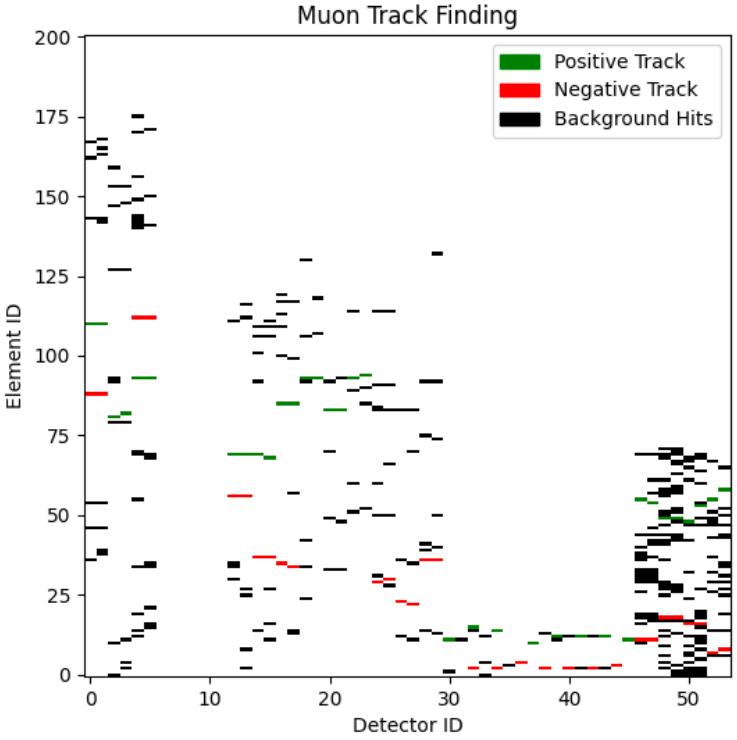


FIGURE 5.4: The tracks for positive and negative muons found in our example event.

Next, the event passes through the muon track finders, which attempt to identify the path that the individual muons took through the detector array. These reconstructed tracks are shown on the hit matrix in Figure 5.4.

These tracks are then used to determine the vertex seperation of these two events – in the case of our example event, the two tracks are found to be seperated by just 13 cm, well within the threshold to be considered a true dimuon.

We also check the quality of each track in terms of the number of drift chamber mismatches (how many hits in primed and unprimed chambers are seperated by more than 1 element) and how many interpolated hits there are (that is, track elements that do not correspond with a physical hit). In the case of our example event, there are no chamber mismatches or interpolated hits, indicating a strong likelihood of a real track.

If those quality cuts are met, then the event is passed on to the dimuon portion of QTracker.

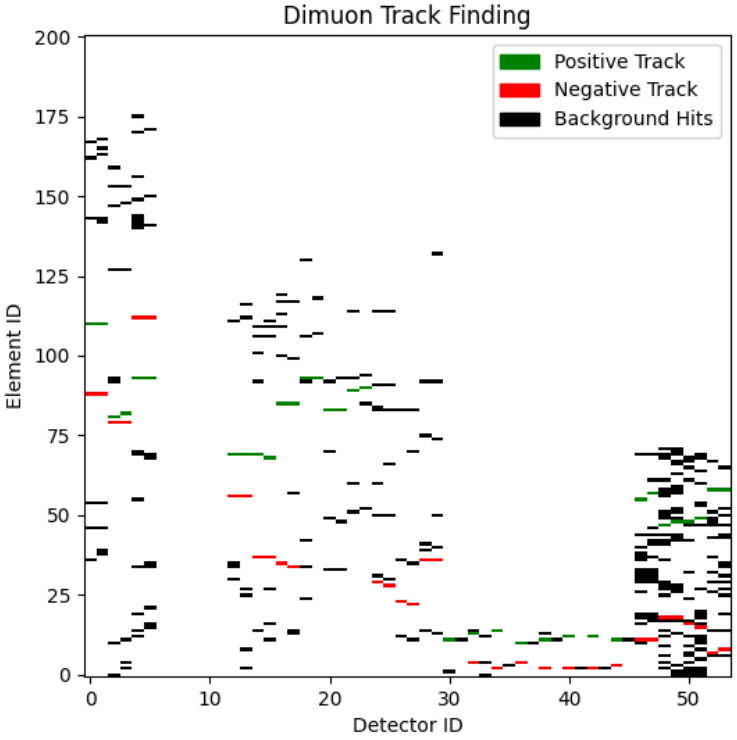


FIGURE 5.5: The tracks for positive and negative muons found in our example event by the Dump Track Finder.

5.1.4 Dimuon Reconstruction

Once our event is identified as a potential dimuon event, it passes through the four dimuon track finders, which each attempt to find the track contained in the event, based on the vertex assumptions that the finder was trained on. These generally produce very similar results, but can identify small but important differences. For brevity, we will only display one such dimuon track (the others differ by at most one hit). The dimuon-reconstructed track is shown in Figure 5.5

The four tracks are then fed into their corresponding momentum and vertex reconstruction networks, which each provide their own predictions for the momentum and location of origin for the dimuon pairs. The results from that are shown in Table 5.1.

Lastly, the identified tracks and these reconstructed values are fed into the Target-Dump classification network, which identifies which events likely originate in either the target region or dump region. In the case of our example event, it is not a

	All-Vertex	Beamline	Target	Dump
p_x^+	1.59 GeV	1.85 GeV	1.71 GeV	1.79 GeV
p_y^+	-2.18 GeV	-1.85 GeV	-2.09 GeV	1.83 GeV
p_z^+	38.2 GeV	31.6 GeV	38.6 GeV	31.2 GeV
p_x^-	-1.69 GeV	-1.80 GeV	-1.54 GeV	-1.72 GeV
p_y^-	2.16 GeV	2.15 GeV	2.53 GeV	2.14 GeV
p_z^-	45.8 GeV	43.7 GeV	46.1 GeV	42.5 GeV
v_x	0.23 cm			
v_y	0.74 cm			
v_z	25.4 cm	42.8 cm		

TABLE 5.1: Predictions for the four different vertex distributions QTracker was trained on. Blank spaces do not have predictions made. Based on the vertex information, the dimuon pair appears to have originated in the dump region.

difficult classification for the network, as it assigns a 99.7% probability of the dimuon originating in the dump.

The reconstructed particle characteristics, as well as information about the tracks and important metadata such as the run, spill, and event IDs, occupancy, and trigger configuration.

5.2 Cut Selection

In order to analyze the reconstructed data further, we first needed to adequately select reconstructed events from the SeaQuest E906 data. In this section, we outline the process of cut selection, a component of our data analysis method aimed at isolating relevant events from background noise. The application of these cuts significantly enhances the precision and reliability of experimental results.

The primary objective of cut selection is to refine the dataset to include only those events that are most likely to be of interest (i.e., dimuon events from the target), thereby effectively reducing the background noise.

There are generally four types of cuts that we can make to the data – quality cuts, kinematic cuts, geometric cuts, and probabalistic cuts.

Quality cuts are based on the reconstructed tracks themselves, without information coming from their predictions. These involved the number of hit mismatches, that is, places where the hits in a track do not align with their neighboring hits (namely,

hits in the primed and unprimed drift chamber planes being seperated by more than one element).

Kinematic cuts are based on the kinematic properties of the muons, such as momentum and energy. They are instrumental in excluding events that do not conform to the expected kinematic signatures of the particles under study.

Geometric cuts are based on the spatial properties of the particle interactions, including the vertex position and the angles between tracks. These cuts are vital for ensuring that the events originate from the intended interaction region and not from other regions of the detector.

Probabilistic cuts leverage the machine learning algorithms to assign probabilities to each event, indicating the likelihood of it originating from the signal or background processes. By setting a threshold probability, we can effectively filter out events with a low probability of being signal events.

The cut selection process for the E906 data analysis was iterative and involved careful consideration of various factors. Initial cuts were chosen based on theoretical expectations and previous experience. The data was then analyzed with these initial cuts, and the results were carefully evaluated. Based on these results, the cuts were refined, and the process was repeated until a satisfactory set of cuts was achieved.

This selection process also involved identifying events with a high probability of being correctly reconstructed, by randomly generating a large number of Monte Carlo events, with dimuon events from the target, dimuons from other areas, and combinations of single-muons that could combine to trick QTracker.

This involved generating over 200 million random combinations of events, which were then analyzed by QTracker, which allowed us to identify which cuts produced the greatest number of correctly identified dimuons from the target.

This iterative process of cut selection and Monte Carlo simulation refinement was necessary for ensuring the reliability and accuracy of our data analysis. By carefully selecting and optimizing our cuts, we were able to significantly reduce the background noise and isolate the dimuon events originating from the target with a high degree of confidence.

To verify the effectiveness of the cut selection process, we conducted a series of checks and comparisons. We compared the distributions of relevant variables (such as invariant mass, momentum, etc.) before and after the application of cuts. Significant differences in these distributions served as indicators of the successful reduction of background events. Additionally, we cross-validated our results with the output of the Monte Carlo simulations, ensuring that our selected cuts aligned with the predictions from the simulations.

The cut selection process, informed by Monte Carlo simulations, had a profound impact on the quality of our data analysis. It led to a substantial improvement in the signal-to-noise ratio of our data, enabling us to extract meaningful information about the dimuon events from the target with greater precision. This enhanced the statistical significance of our results and ultimately contributed to the reliability and accuracy of our experimental findings.

5.2.1 List of Cuts

In total, we used eleven cuts in our analysis. This included three quality hits, three kinematic cuts, three geometric cuts, and two probabilistic cuts.

The quality cuts included:

- A hit cut based on hit mismatches between primed and unprimed chamber hits.
- An spill-based cut based on the quality of beam during the spill.
- An event-based cut based on the trigger to select FPGA-1 events.

The kinematic cuts included:

- Muon momentum-based cuts to remove physically impossible particles.
- Dimuon momentum-based cuts to remove physically impossible dimuon pairs.
- Target and Beam x-based cuts to remove dimuon pairs known to be outside of the acceptance range of the spectrometer.

The geometric cuts included:

- Beamline cuts, ensuring that the dimuon pair was reconstructed along the experimental beamline.
- Seperation cuts, ensuring that the two muons were reconstructed to the same region of the beamline.
- Location cuts, ensuring that the reconstructed vertex of the dimuon pair is consistent with the desired area.

And finally, the probabilistic cuts were:

- Dimuon probability cut, ensuring that the dimuon pair was not simply a random coincidence of detector hits.
- Target-Dump probability cut, adding to the target-dump separation performed using the geometric cuts.

These cuts were selected based on knowledge from previous analyses of E906 data, as well as trial and error with QTracker-reconstructed data. The specifics of these cuts can be tweaked based on what is required for the analysis which they are used in.

5.2.2 Combinatoric Studies

In order to determine the effect of these cuts on data, we looked at the effects of the cuts on the signal to noise ratio of the passing events, as well as the type of non-dump events that triggered false-positives.

For this combinatoric study, we looked at the selection of target dimuons. Each Monte Carlo event was generated with between 1 and 4 of each positive and negative dimuons, and needed to pass QTracker's reconstruction algorithm. Because of the single-muon track cuts in QTracker.

We then categorized each event into one of four categories:

- Two Muons: Events whose muons do not come from the same region of the beamline.
- Dimuon-Passing: Events with uncorrelated positive and negative muons that originate in the same region of the beamline.
- Non-Dump Dimuon: Events with a dimuon pair, but no dimuon pair from the dump.
- Dump Dimuon: Events that contain a dimuon pair originating in the dump region.

We then applied the analytical cuts to these events and looked at the signal-to-noise ratio after applying each cut, as well as the origins of the muons and dimuons that tricked the filters.

The cuts are shown in Table

Filter Name	**Condition**
`Hit Cuts`	No interpolated hits and no hit mismatches.
`Trigger Cuts`	N/A.
`Spill Cuts`	N/A.
`Muon Momentum Cuts`	$\|p_y\| > 0.02)$
`Dimuon Momentum Cuts`	$(\|p_x\| < 2)$ & $\|p_y\| < 2$ & $38 < p_z < 116$
`X Cuts`	$x_F > -0.1$ & $x_F < 0.9$ & $x_2 > 0.05$
`Beamline Cuts`	$\sqrt{v_x^2 + v_y^2} < 5$
`Seperation Cuts`	$\|v_z^+ - v_z^-\| < 130$
`Dump Location Cuts`	$v_z > 0$
`Dump Probability Cut`	$p_{Dump} > 0.95$
`Dimuon Probability Cut`	$1 - p_{Dimu} < 1 \times 10^{-5}$

TABLE 5.2: Cuts used for the combinatoric study. Trigger cuts and spill cuts are not applicable for Monte Carlo events.

Figure 5.6 shows the vertex-dependence of filtering for events that do not contain target dimuons and yet pass the full set of cuts. Muons originating near the target region are the most likely to pass the filter, suggesting that the cuts are performing as expected, even on messy combinatoric data. Figure 5.7 shows the signal-to-noise ratio before and after each category of cut. The prevalence of non-Dump Dimuon events is reduced from 50% of the data to 10%, a five-fold reduction.

Using these cuts on reconstructed E906 data, we find 124553 dump dimuons. The mass curve is shown in Figure 5.8.

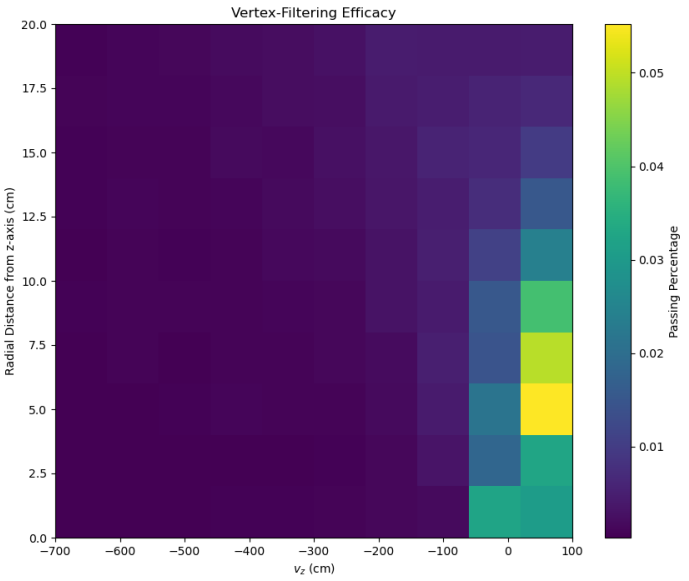


FIGURE 5.6: The filtering efficacy based on the vertices of muons in events not including a dump dimuon. Muons originating close to the dump region are the most likely to pass.

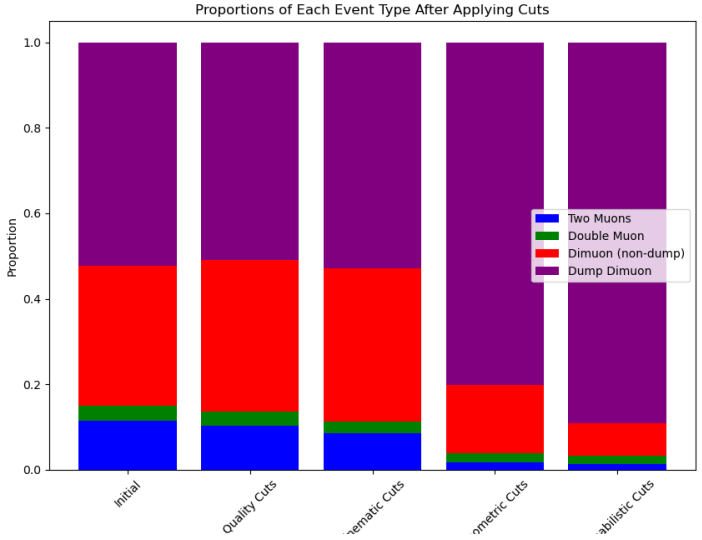


FIGURE 5.7: The effects of each type of cut on the signal-to-noise ratio in the combinatoric data.

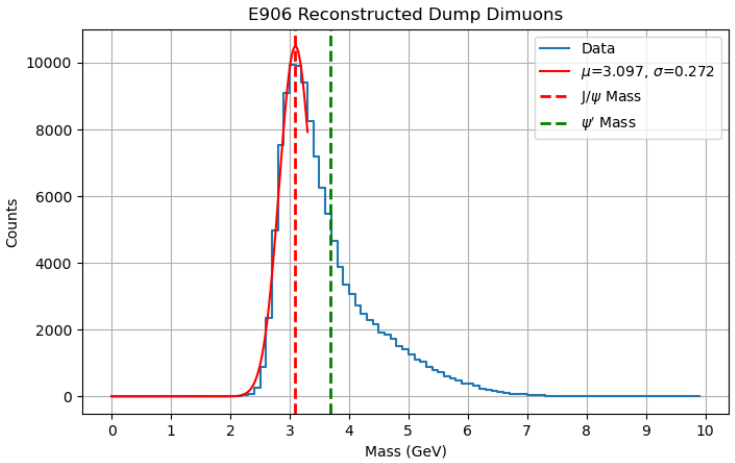


FIGURE 5.8: The reconstructed Mass curve for filtered dump-origin dimuons in Runs 5-6 of E906. The J/ψ and ψ' masses have been added for reference, as well as a Gaussian fit to the left side of the J/ψ peak in the data. The fit has a mean of 3.097 GeV, and a width of 0.27 GeV. This is a very good agreement with the known J/ψ mass.

5.3 Mass Curve Decomposition

Now that we have properly selected for dimuons originating from the target, we can decompose the reconstructed experimental data into the four potential sources of the dimuons. Those sources are

- J/ψ Dimuons,
- ψ' Dimuons,
- Drell-Yan Dimuons, and
- Combinatoric Background.

For J/ψ, ψ', and Drell-Yan dimuons, we are able to use Fun4Sim to generate Monte Carlo events that pass the trigger configuration to achieve a good representation of the data. This MC, while good, is not perfect, which will necessitate the use of smearing to accurately represent experimental data.

The combinatoric background, on the other hand, is harder to generate. Because we do not know the exact sources or distributions of single-muons that may combine to mimic a dimuon pair, starting from pure Monte Carlo is a tall order. Fortunately, this problem was known before data collection started. As shown

in Table 3.7, the FPGA-4 trigger was speifically created to collect single-muon events.

The upside of this is that we will be able to combine these single-muon events, then reconstruct them to accurately represent coincident muons. We can then proceed, knowing that these combined events are an accurate representation of the combinatoric events created during data taking.

5.3.1 Monte Carlo Smearing and Covariance Analysis

Due to the inherent imperfections in real-world data, the resolution of reconstructed MC data tends to be better than that of reconstructed experimental data. Therefore, to achieve accurate comparisons, we must apply a smearing function to the reconstructed MC data.

To determine the appropriate level of smearing, we analyze the width of the J/ψ peak in both the reconstructed E906 data and the MC data, along with their corresponding reconstructed energies. This allows us to apply an energy correction to the MC data, aligning the energy distributions of positive and negative muons with those observed in the real data. Additionally, we introduce smearing to the MC kinematics using a Gaussian distribution, adjusting the J/ψ peak width to match the implied width from the experimental data. This comprehensive process is referred to as the covariance analysis.

The covariance analysis involves a detailed study of the covariance matrix of the reconstructed variables to understand the differences in the residuals (kinematically dependent error distributions) between the experimental and MC data in the most fundamental variables of the feature space. This includes the momentum components p_x, p_y, p_z, and energy E of the muons.

First, we calculate the residuals for p_x, p_y, and p_z by determining the implied error based on the mass residual of the J/ψ peak. This was done using an estimate based on the partial derivative of the mass in terms of the individual parts of the momenta. The invariant mass M of a dimuon pair is given by:

$$M^2 = (E_1 + E_2)^2 - (\vec{p}_1 + \vec{p}_2)^2 \tag{5.1}$$

where E_1 and E_2 are the energies, and $\vec{p}_1$ and $\vec{p}_2$ are the momenta of the two muons.

The energy of each muon is given by the relativistic energy-momentum relation:

$$E_i = \sqrt{p_{i_x}^2 + p_{i_y}^2 + p_{i_z}^2 + m_\mu^2} \tag{5.2}$$

To find the partial derivative of M with respect to one of the spatial dimensions of the momentum of one of the muons, let's choose p_{1_x}.

We find that:

$$\frac{\partial M}{\partial p_{1_x}} = \frac{1}{M}((E_1 + E_2)\frac{p_{1_x}}{E_1}) - (p_{1_x} + p_{2_x})) \tag{5.3}$$

This estimate can then be used to give us an approximate error for p_{1_x} as follows:

$$\Delta p_{1_x}^0 = \frac{m_{reco} - m_{J/\psi}}{\frac{\partial M}{\partial p_{1_x}}|_{p_{1_x}=p_{1_x}^0}}. \tag{5.4}$$

Next, we develop smearing functions based on these residual distributions. These functions are modeled using Gaussian distributions where the widths are derived from the residuals. The smearing functions are then applied to the MC data to match the residual distributions of the experimental data, ensuring that the MC data accurately represents the experimental uncertainties. The plots of the residuals versus their corresponding variables are shown in Figure 5.9 for the experimental data, as well as for the Monte Carlo. The plots of the residuals versus the other kinematically reconstructed variables were also studied, but are not included for readability. There is a discrepancy in the width of the MC data versus the experimental data that necessitates a 7% Gaussian smearing function to the reconstructed MC data.

Finally, we verify the effectiveness of the smearing functions by comparing the widths of the J/ψ peaks in the smeared MC data and the experimental data. The goal is to achieve a close match between the two, indicating that the MC data has been appropriately adjusted to reflect the experimental conditions.

As illustrated in Figure 5.10, the experimental mass curve exhibits a width of 0.27 GeV, whereas the reconstructed J/ψ Monte Carlo data from the dump shows a

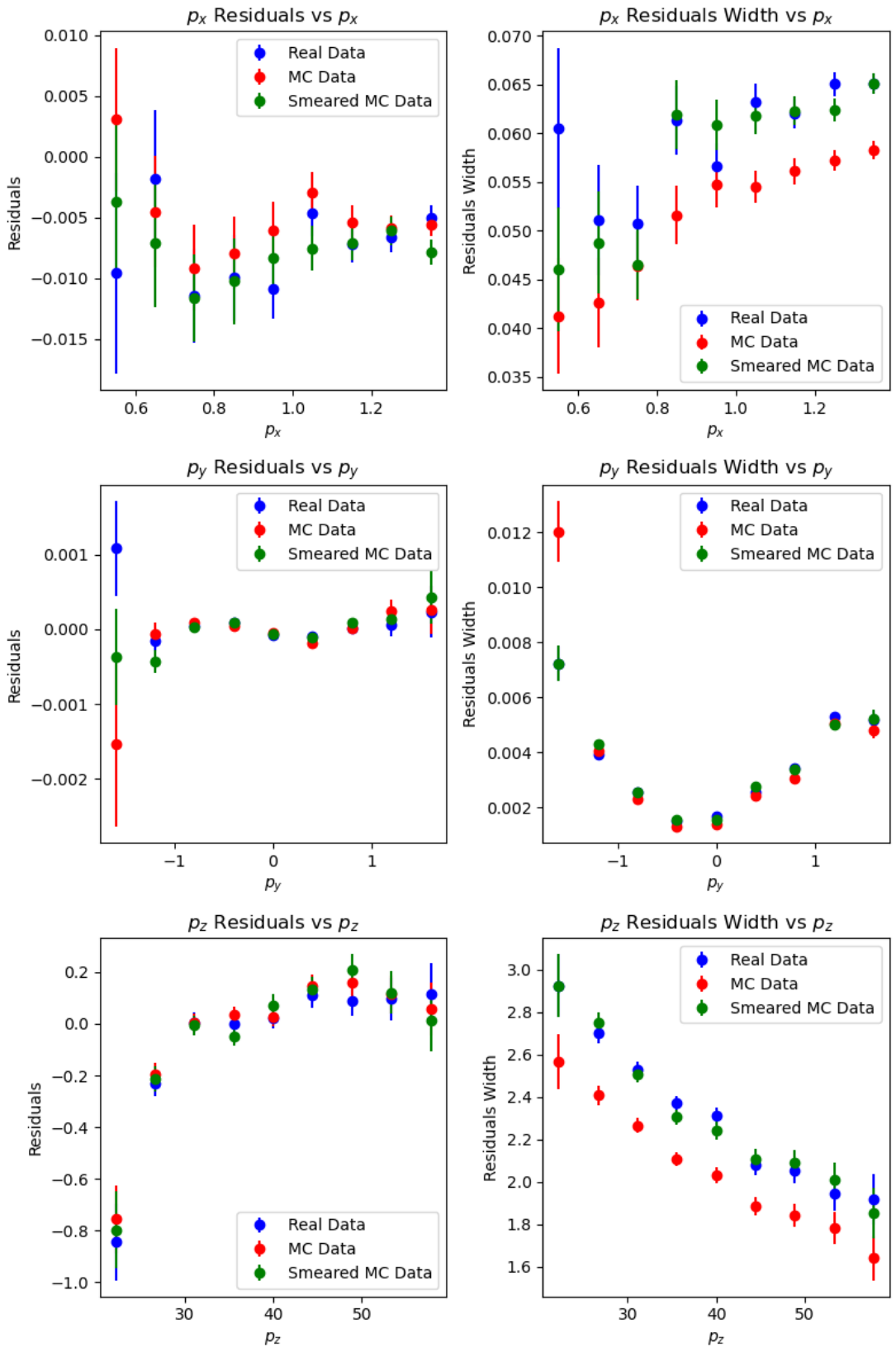


FIGURE 5.9: The covariance of the J/ψ peak for real and MC data both before and after a 7% Gaussian smearing is applied. After smearing, the patterns, especially of the width of the residual, have much closer agreement.

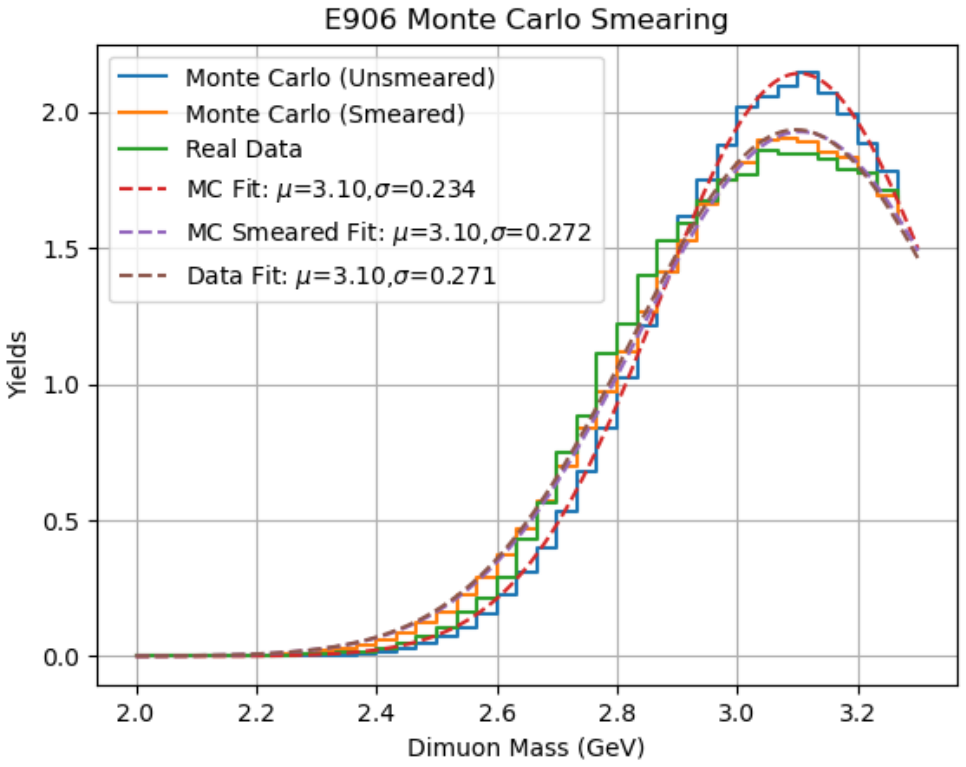


FIGURE 5.10: The unsmeared and smeared J/ψ peak compared to the experimental peak. This analysis demonstrates that a 7% Gaussian smearing function matches the precision of the MC data to the experimental data.

width of 0.23 GeV. After smearing, the widths of the J/ψ peaks in both the MC and experimental data align. This level of smearing is validated by the use of a 6% smearing value in previous SeaQuest analyses conducted using KTracker.

5.3.2 Combinatoric Estimate

Despite our best efforts with cuts, the rate of muon production in the beam dump means that a non-trivial amount of combinatoric background will pass through. This necessitates an accurate recreation of the combinatoric background based on the experimental data.

To achieve this, we can select events from the triggered experimental data that contain either a positive or a negative muon, but not both. This can be achieved by selecting FPGA-4 events, and using the single-muon track finders on the triggered events.

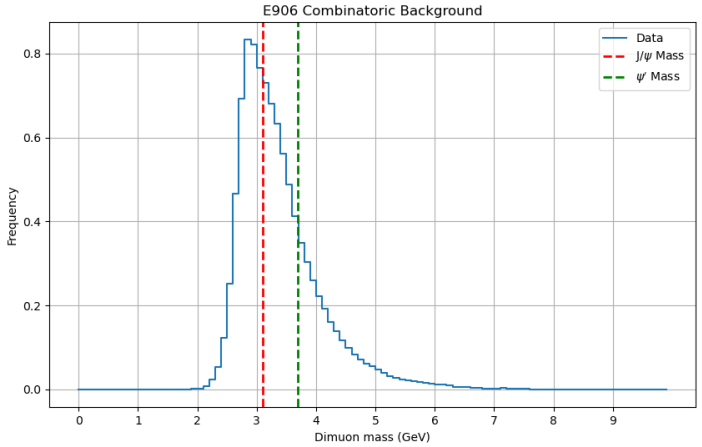


FIGURE 5.11: The combinatoric background for E906, based on an event-mixing method using positive and negative single-muon events.

If the track finders are able to identify either a positive or a negative muon, then that event's hit matrix is saved to an array – one array for positive muons, and one for negative muons.

Those events can then be combined by overlaying the hits of one onto the other. The combined events are then reconstructed using the full QTracker chain, which allows us to use the same cuts for the combinatoric estimate as we do for the Monte Carlo and real data.

Using the same cuts as those in Table 5.2, we can find the mass distribution of the combinatoric background shown in Figure 5.11.

Although the peak is at a similar position to the reconstructed data, it is slightly below the $J\psi$ mass, and falls off faster in the higher mass region. This is promising, as it means that the mass region most contaminated by the combinatoric background is the same as that contaminated by the charmonium-produced dimuons.

This estimate of the combinatoric background is especially important for extracting the angular dependence for dimuons originating in the dump, as the dump is a large source of combinatoric background.

The combinatoric background for QTracker will be fundamentally different from that of KTracker for a few reasons. Because QTracker searches for events with exactly one positive and one negative muon track, while KTracker accepts events with more than one of each, QTracker does not have as much of a likelihood of confusing independent muons in the same event as a dimuon. However, QTracker

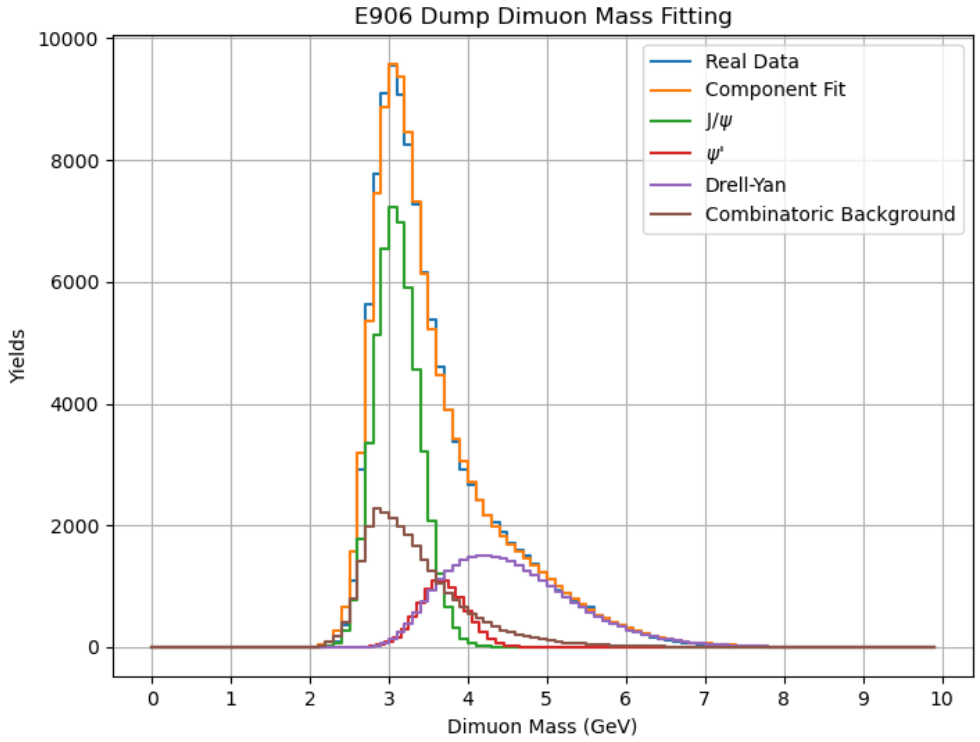


FIGURE 5.12: The mass curve fit for filtered dump-origin single-dimuon events in Runs 5-6 of E906, fit with Monte Carlo J/ψ, ψ', Drell-Yan, and combinatoric dimuons.

is able to, in a high-occupancy situation, find a poor quality track to match with a true track, which can also appear to be a dimuon under the right circumstances. As such, track quality is an important value to monitor when performing analyis using QTracker.

5.3.3 Fitting

To fit the experimental mass curve with the four components, we can use a curve fit. The fit imposes a relation between the J/ψ and ψ' prevalence in the data based on the branching ratio and trigger condition. This ratio is approximately 16%, but varies based on the specific trigger condition and cuts used.

Figure 5.12 shows the dump dimuon data from E906, fit with the smeared Monte Carlo and the estimated combinatoric data.

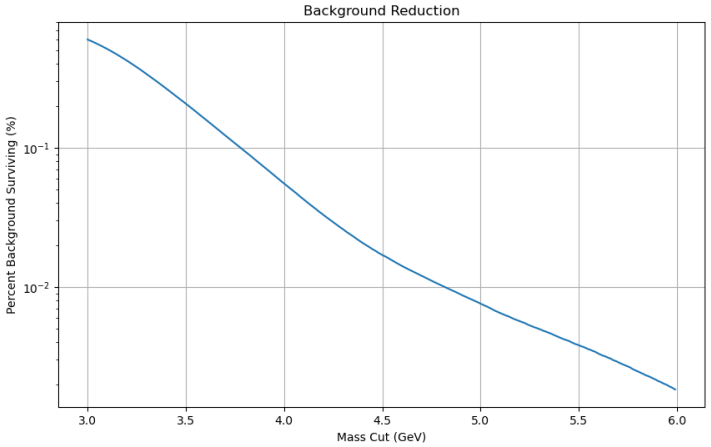


FIGURE 5.13: Percentage of non-Drell-Yan dimuons remaining as a function of mass cut. A mass cut of 4.8 GeV removes 99% of background events.

The fit finds that of the 124,000 dimuons, approximately 59,000 are J/ψ dimuons, 8,500 are ψ' dimuons, 34,000 are Drell-Yan dimuons, and 24,000 are combinatoric background.

5.4 Angular Dependence Extraction

Now that we have properly decomposed the experimental data into its components, we can extract the angular dependence of the Drell-Yan dimuons.

5.4.1 Mass Cut

In order to isolate the Drell-Yan dimuons, we can utilize a simple mass cut. To determine the level of that mass cut, we can use the mass curve fit to estimate the percentage of non-Drell Yan events that are removed by a simple mass cut.

This curve is shown in Figure 5.13. Based on this analysis, we should choose a mass cut of 4.8 GeV to remove 99% of the non-Drell Yan dimuons.

Once we have made this mass cut, we can examine the angular dependence of the dimuons in the data. Figure 5.14 shows the angular distribution of these dimuons.

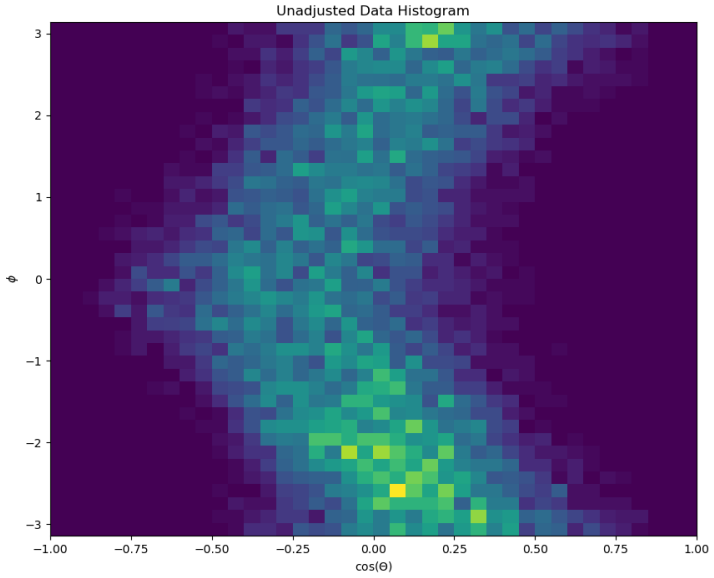


FIGURE 5.14: The uncorrected distribution of ϕ and $\cos\theta$ for dimuons from the dump with masses over 4.8 GeV.

5.4.2 Acceptance Correction and Data Matching

To accurately extract the angular dependence of the Drell-Yan process from the data, it is essential to correct for the combined acceptances of the spectrometer and QTracker. Acceptance correction involves adjusting the observed data to account for the detector's efficiency and geometric coverage, ensuring that the reconstructed distributions reflect the true physical distributions.

The process begins by generating a large sample of Drell-Yan MC data. These simulated events are then reconstructed using the same procedures applied to real data. To enhance the accuracy of the acceptance correction, we employ a data-matching technique by reweighting MC events. This involves adjusting the weights of individual MC events to better match the kinematic distributions observed in the actual data. The reweighting procedure uses a comparison between the distributions of key variables, such as the invariant mass and transverse momentum (p_T), in both the real and simulated datasets. By applying these weights, the MC sample more accurately reflects the conditions of the experimental data, thus improving the fidelity of the acceptance correction.

To compute the weights, we first generate histograms of the selected variables from the real data, background data, and MC data. The weights are then calculated as

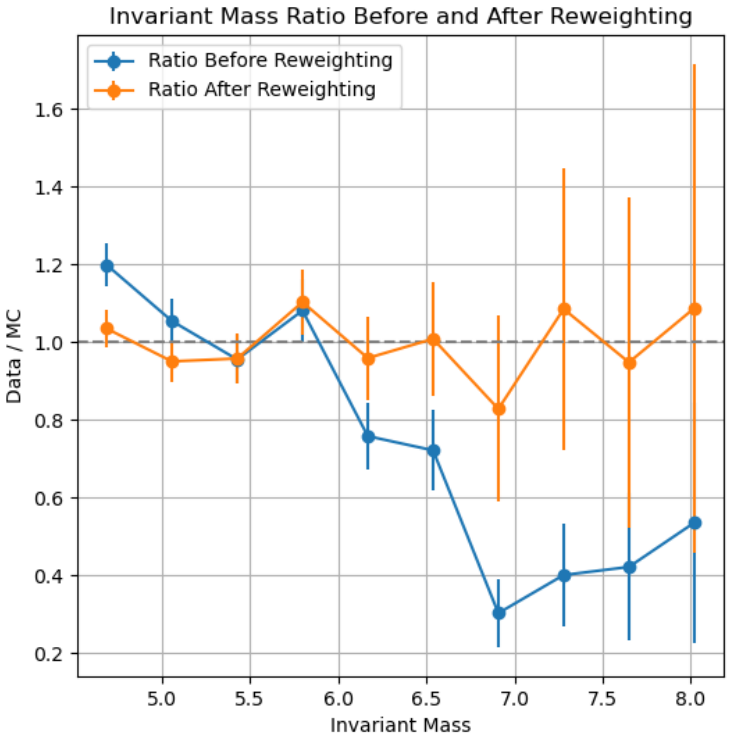


FIGURE 5.15: The ratio of real data to MC data for the invariant mass distribution before and after reweighting.

the ratio of the real data histogram to the MC data histogram, adjusted for the background. This ensures that the weighted MC data better matches the real data distribution across the full range of the variable. The weights are subsequently interpolated to each MC event to apply the correction.

To illustrate the reweighting procedure, consider the invariant mass distribution of the dimuon pairs. Figure 5.15 shows the ratio of the real data to the MC data before and after reweighting. Before reweighting, there are noticeable discrepancies between the real and simulated data, particularly at higher invariant mass values. After reweighting, the MC data more closely matches the real data across the entire range of invariant mass, demonstrating the improved accuracy of the reweighted MC sample.

After reweighting, the reconstructed MC data is further adjusted to have a flat angular distribution, removing the $1 + \cos^2\theta$ dependence generated by Pythia.

The naive Drell-Yan cross-section, as implemented in the Pythia event generator, is given by:

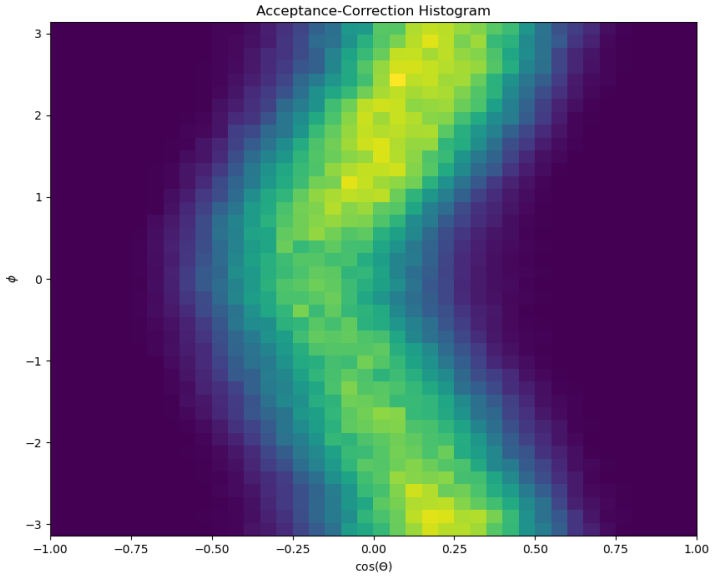


FIGURE 5.16: The normalized acceptance for the spectrometer and QTracker for Drell-Yan dimuons with mass greater than 4.8 GeV.

$$\frac{1}{\sigma}\frac{d\sigma}{d\Omega} = \frac{3}{4\pi}\frac{1}{\lambda+3}(1+\lambda\cos^2(\theta)) \tag{5.5}$$

This expression serves as a basis for calculating the acceptance correction. By comparing the reconstructed MC data to the known theoretical cross-section, we can derive the acceptance function. Specifically, a two-dimensional histogram of acceptance is created, where the acceptance is defined as the ratio of the reconstructed MC data to the theoretical cross-section.

Figure 5.16 illustrates the normalized acceptance for the spectrometer and QTracker for Drell-Yan dimuons with invariant mass greater than 4.8 GeV. The histogram reveals that the acceptance is concentrated in a band around $\cos\theta = 0$, indicating that the spectrometer's efficiency is highest near perpendicular angles.

Given this acceptance pattern, we apply the acceptance correction to the reconstructed data and restrict the range of $\cos(\theta)$ to between -0.4 and 0.4 for the Chi-Square fitting. This limitation ensures that we only use data where the acceptance is well understood and reliable.

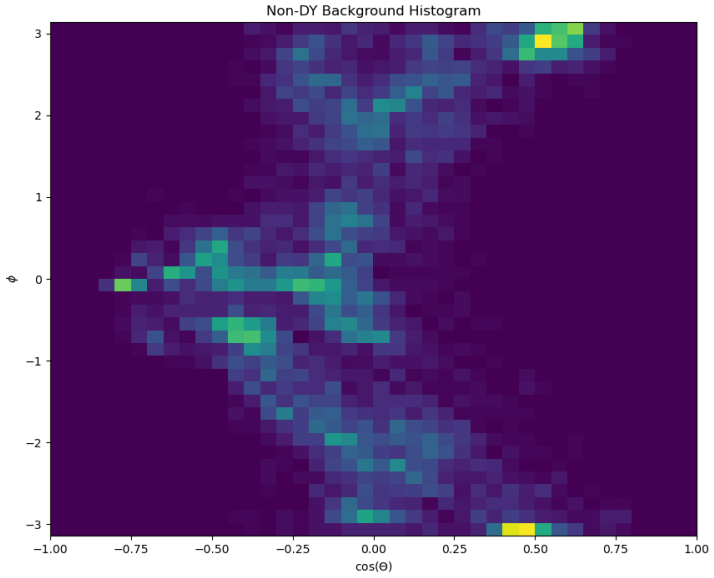


FIGURE 5.17: The angular distribution of background dimuons (before acceptance corrections).

5.4.3 Background Subtraction

To accurately fit the angular dependence of the Drell-Yan dimuons, we must account for the background contribution estimated from the mass fit. This is accomplished by creating histograms for the ϕ and $\cos\theta$ distributions from both the real data and the estimated background. The background histograms are then subtracted from the data histograms on a bin-by-bin basis.

The procedure involves generating corresponding histograms for the estimated background, using the same binning as the real data, then subtracting the background histograms from the data histograms for each bin.

By removing the estimated background events, this subtraction method refines the ϕ and $\cos\theta$ distributions, providing a clearer view of the true angular dependence of the signal.

The corrected histograms can then be analyzed to study the physical properties of the signal without the interference of background events. The angular dependence of these estimated background hits are shown in Figure 5.17

5.4.4 Fitting

Once the acceptance correction is applied and background subtraction is performed, the next step is to determine the angular dependence of the Drell-Yan process using a Chi-Square fit. The goal is to extract the parameters λ, μ, and ν from the angular distribution of the dimuons.

The angular differential cross-section for Drell-Yan data, incorporating the full angular dependence, is expressed as:

$$\frac{1}{\sigma}\frac{d\sigma}{d\Omega} = \frac{3}{4\pi}\frac{1}{\lambda+3}(1+\lambda cos^2(\theta)+\mu sin(2\theta)cos(\phi)+\frac{\nu}{2}sin^2(\theta)cos(2\phi)) \tag{5.6}$$

To perform the Chi-Square fit, we construct a histogram of the acceptance-corrected data in terms of $cos(\theta)$ and ϕ. This histogram is then used to fit the theoretical angular distribution, extracting the values of λ, μ, and ν.

We need to carefully select the number of bins used for the fit. We can base our choice on bin size on the precision found in 4.3. By choosing a bin width of 2σ, 95% of reconstructed events will fall within the correct bin. This corresponds to $\cos\theta$ bins having a width of 0.08, and ϕ bins having a width of approximately 0.6. This corresponds to 10 bins in ϕ, and 10 bins in $\cos\theta$.

Figure 5.18 shows the acceptance-corrected $cos(\theta)$ vs ϕ histogram along with a contour plot of the Chi-Square fit.

The fit yields the following angular parameters:

- $\lambda = 0.82 \pm 0.70$
- $\mu = 0.11 \pm 0.10$
- $\nu = -0.02 \pm 0.05$

The uncertainties in these parameters quite large, especially for the λ value. In order to reduce the statistical uncertainty, we can apply a bootstrapping method to the dataset. Bootstrapping involves repeatedly resampling the data with replacement and performing a regression fit on each resampled dataset. This technique

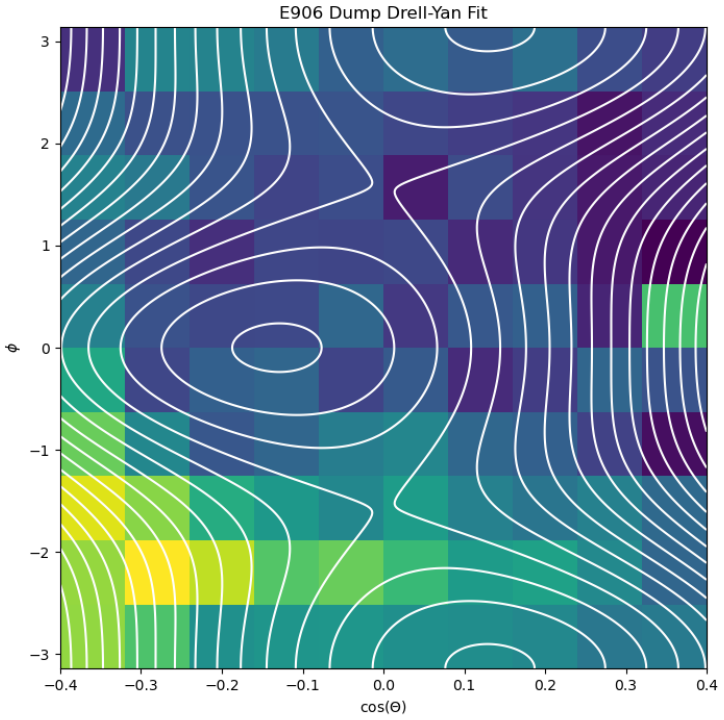


FIGURE 5.18: The Chi-Square fit for filtered dump-origin Drell-Yan single-dimuon events in Runs 5-6 of E906. The extracted values are $\lambda = 0.82 \pm 0.70$, $\mu = 0.11 \pm 0.10$, and $\nu = -0.02 \pm 0.05$ (statistical errors only).

provides a way to estimate the distribution of the fitted parameters and their uncertainties more robustly.

The procedure for bootstrapping is as follows:

Resample the Dataset: Create resampled datasets by randomly selecting events from the original dataset with replacement. Each resampled dataset is the same size as the original dataset.

Perform Regression Fits: For each resampled dataset, construct the $cos(\theta)$ vs ϕ histogram and perform a least-squares regression fit to extract the parameters λ, μ, and ν.

Analyze the Distribution: Collect the fitted values of λ, μ, and ν from all resampled datasets and analyze their distribution. Calculate the mean and standard deviation of these distributions to obtain the bootstrapped estimates of the parameters and their uncertainties.

The data was resampled and fitted 1,000 times. The bootstrapped results yield the following angular parameters:

- $\lambda = 0.84 \pm 0.39$
- $\mu = 0.12 \pm 0.06$
- $\nu = -0.02 \pm 0.02$

Comparing these values with the initial fit results, we observe an agreement in the values, but a reduction in statistical uncertainty. These bootstrapped results demonstrate the robustness of the Chi-Square fit and provide more precise estimates of the angular parameters. This enhanced precision is crucial for interpreting the Drell-Yan process and understanding the underlying parton dynamics.

Table 5.3 summarizes the effect of varying the selection cuts on the extracted angular parameters λ, μ, and ν. Understanding how these parameters shift with different selection criteria is crucial for validating the fit that we chose.

By comparing results across different cuts, we can assess how sensitive the extracted angular parameters are to variations in the selection criteria. This helps identify potential biases or systematic effects.

None of the different cuts that we modified created statistical disagreement with our intitial extraction. This suggests that the chosen cuts are not articifically creating the extracted values, but are actually a product of the underlying data. However, there is some variability of the extracted λ values especially. This is a known issue with SeaQuest data, and has led some previous analyses to fix the λ value to unity and only fit the μ and ν values.[71] The variablility, especially of λ, do indicate the presence of non-negligible systematic uncertainty.

The variability in the extracted angular distribution parameters can be explained by several factors. Firstly, statistical uncertainty plays a role, especially when dealing with limited data samples. Smaller datasets result in larger uncertainties in the parameters, as evidenced by the initial fits showing large error margins. Bootstrapping, by resampling the data, helps reduce statistical error, but the inherent randomness in resampling can still introduce some variability in the parameter estimates. Additionally, systematic errors such as those from detector calibration, particle identification, and event reconstruction can affect the accuracy of the measured distributions.

Original Cut	Less Stringent Cut		More Stringent Cut	
Classification Cut p_{dimuon}**>0.99**	p_{dimuon}>0.9	$\lambda = 0.88 \pm 0.37$ $\mu = 0.13 \pm 0.07$ $\nu = -0.02 \pm 0.02$	p_{dimuon}>0.999	$\lambda = 0.55 \pm 0.52$ $\mu = 0.13 \pm 0.09$ $\nu = 0.00 \pm 0.04$
Beamline Cut **Within 10 cm of beamline**	Within 20 cm of beamline	$\lambda = 0.76 \pm 0.38$ $\mu = 0.11 \pm 0.05$ $\nu = -0.01 \pm 0.02$	Within 5 cm of beamline	$\lambda = 1.17 \pm 0.59$ $\mu = 0.07 \pm 0.09$ $\nu = -0.04 \pm 0.03$
Muon Separation Cut $\lvert v_z^+ - v_z^- \rvert$**<130 cm**	$\lvert v_z^+ - v_z^- \rvert$<160 cm	$\lambda = 0.96 \pm 0.38$ $\mu = 0.18 \pm 0.06$ $\nu = -0.00 \pm 0.02$	$\lvert v_z^+ - v_z^- \rvert$<100 cm	$\lambda = 1.22 \pm 0.51$ $\mu = 0.15 \pm 0.09$ $\nu = -0.01 \pm 0.02$
Target-Dump Separation v_z**>0,** p_{dump} **>0.9**	v_z>-25 cm p_{dump} >0.75	$\lambda = 0.88 \pm 0.30$ $\mu = 0.07 \pm 0.06$ $\nu = -0.02 \pm 0.02$	v_z>25 cm p_{dump} >0.95	$\lambda = 0.56 \pm 0.43$ $\mu = 0.08 \pm 0.09$ $\nu = 0.00 \pm 0.04$
Mass Cut **m >4.8 GeV/**c^2	m >4.7 GeV/c^2	$\lambda = 0.78 \pm 0.38$ $\mu = 0.06 \pm 0.05$ $\nu = -0.03 \pm 0.02$	m >4.9 GeV/c^2	$\lambda = 0.65 \pm 0.38$ $\mu = 0.10 \pm 0.06$ $\nu = -0.03 \pm 0.03$
All Cuts	All above	$\lambda = 1.01 \pm 0.32$ $\mu = 0.13 \pm 0.04$ $\nu = -0.01 \pm 0.02$	All above	$\lambda = 1.44 \pm 0.67$ $\mu = 0.22 \pm 0.11$ $\nu = -0.00 \pm 0.04$

TABLE 5.3: The reconstructed angular dependence variables with modified cut selections of the data. None of the cut changes change the extracted values at a statistically significant level.

Another important factor is the process of acceptance corrections. Inaccuracies in the acceptance model or inefficiencies in the detector can cause fluctuations in the corrected distributions, and different methods or assumptions used in these corrections can lead to variations in the final extracted parameters. The choice of cuts influences the results, where looser cuts may include more background events, and tighter cuts may exclude some signal events, both affecting the extracted parameters.

Discrepancies between Monte Carlo simulations and actual data, along with inaccuracies in background subtraction methods, can further introduce variability. The stabilization of extracted parameters, especially μ and ν, with tighter cuts suggests that stringent cuts help reduce background contamination, thus improving the reliability of the results. However, these tighter cuts also increase the statistical uncertainty, requiring us to balance between the stability and precision of our extracted values.

5.5 Target Analysis

This entire process was also done for target dimuons, albeit with less rigor. In the data analyzed, there were approximately 39,000 dimuons found, with 20,000 J/ψ dimuons, 4,000 ψ' dimuons, 6,500 Drell-Yan dimuons, and 10,000 combinatoric background events. The mass curve fit is shown in Figure 5.19.

The proportion of target dimuons identified as Drell-Yan are lower compared to those found in the Dump. In the Dump, approximately 20% of the dimuons were identified as Drell-Yan, as opposed to only 17% for the target. In KTracker-reconstructed data, this number is approximately 18% in Liquid Hydrogen data.[71] A mass curve using a portion of the SeaQuest target data using KTracker is shown in Figure 5.20.

The data from runs where the target was liquid hydrogen was selected, and the process of background subtraction and acceptance correction was repeated using that data. The histogram was then fit using the Drell-Yan cross-section. The results of this fit are shown in Figure 5.21.

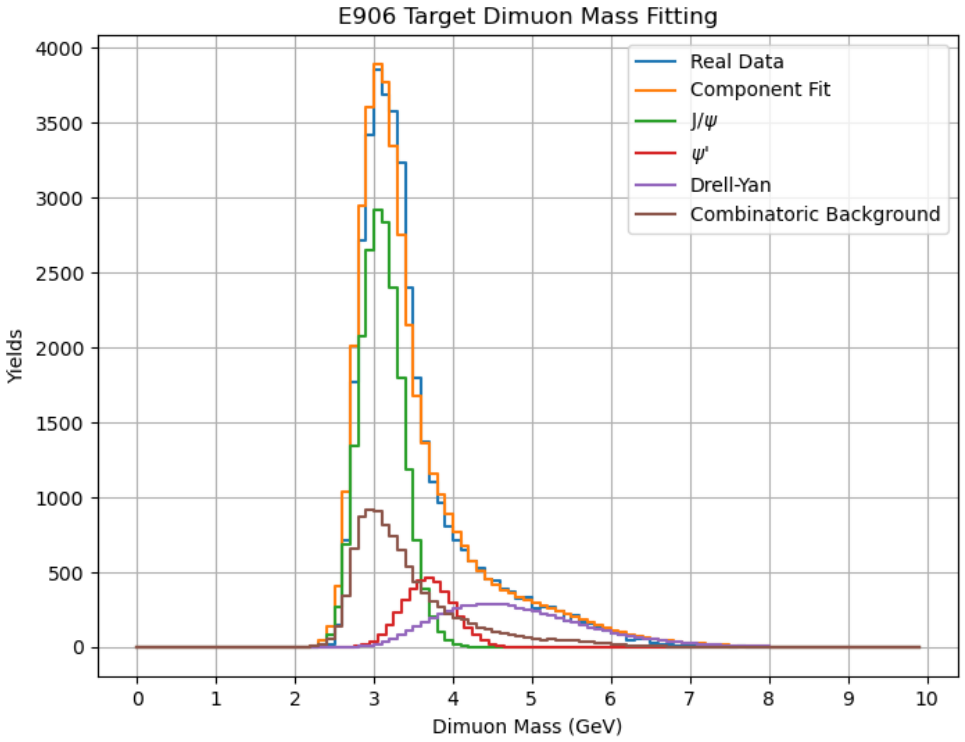


FIGURE 5.19: The mass curve fit for filtered target-origin Drell-Yan single-dimuon events in Runs 5-6 of E906, fit with Monte Carlo J/ψ, ψ', Drell-Yan, and combinatoric dimuons.

Because there were only approximately 16,500 dimuons identified from the hydrogen target, compared to 124,000 from the dump, the statistical uncertainty for this fit is much larger. The fit returned the following angular parameters:

- $\lambda = 1.08 \pm 1.24$
- $\mu = -0.13 \pm 0.18$
- $\nu = 0.13 \pm 0.08$

The given uncertainties are statistical only.

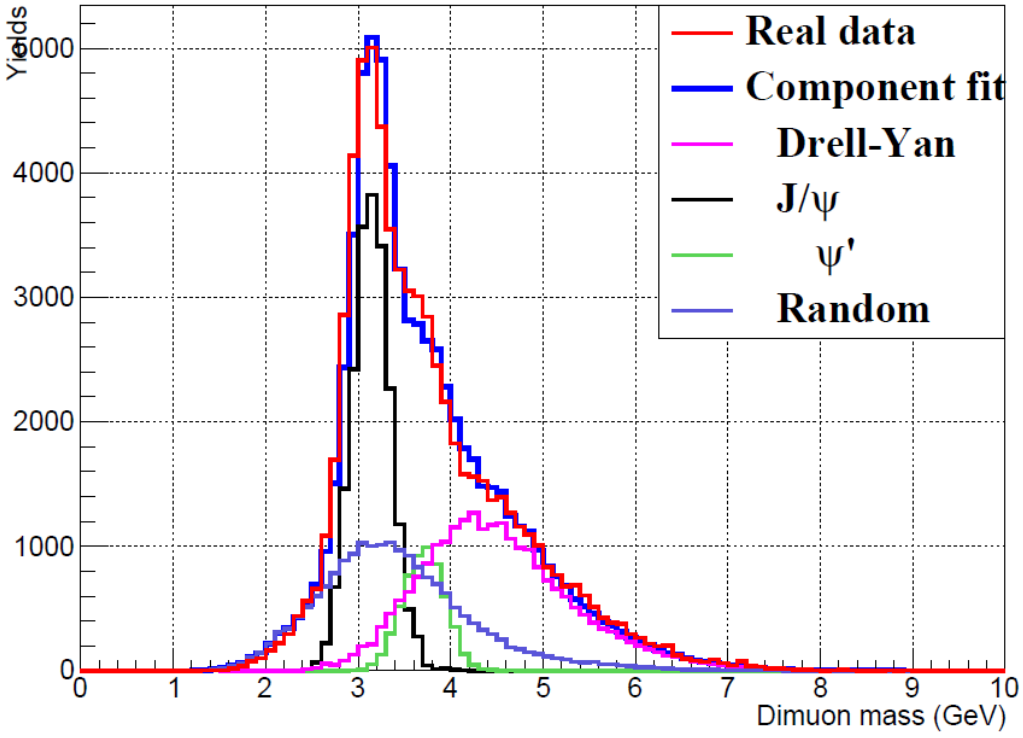


FIGURE 5.20: The KTracker-based mass curve fit for E906 target dimuons, fit with Monte Carlo J/ψ, ψ', Drell-Yan, and combinatoric dimuons. The data shows a more pronouced ψ' shoulder than is seen in the QTracker-based curve.[14]

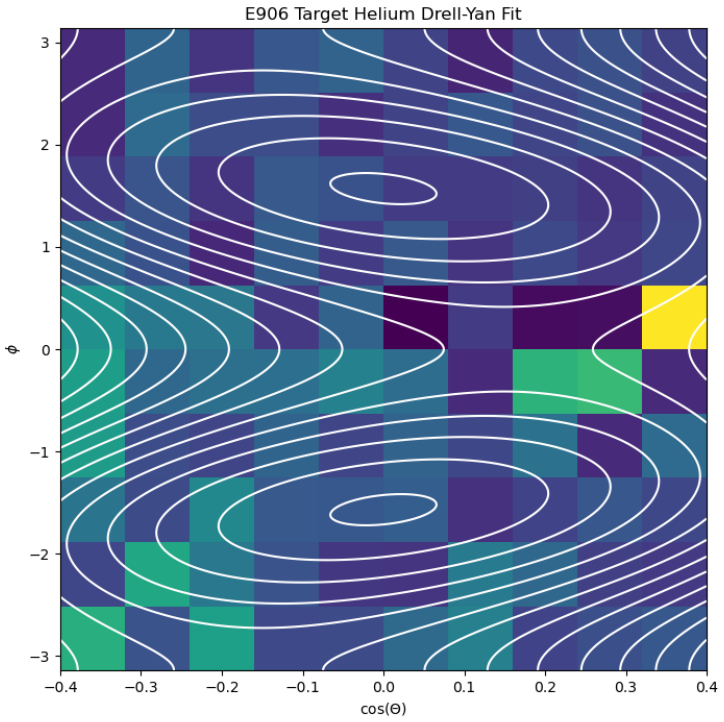


FIGURE 5.21: The Chi-Square fit for filtered target-origin Drell-Yan single-dimuon events in Runs 5-6 of E906 identified from the hydrogen target. The extracted values are $\lambda = 1.08 \pm 1.24$, $\mu = -0.13 \pm 0.18$, and $\nu = 0.13 \pm 0.08$ (statistical errors only).

Chapter 6

Discussion and Future Work

In the last chapter, we demonstrated that QTracker is able to perform analysis on experimental data. There is, however, still work to be done in the future. In this chapter we will discuss the results we found and the improvements still to be made.

6.1 Discussion

6.1.1 Discussion of Results

The Drell-Yan angular dependence has strong implications for multiple transverse momentum distributions, including the Boer-Mulders Function and the Sivers Function. Specifically, the E906 data that was reconstructed in the previous chapter has implicatons for the Boer-Mulders Function.

As was discussed in Chapter 2, a previous experiment at Fermilab, E866, measured the angular dependence of Drell-Yan scattering using an 800 GeV beam. Additionally, a previous analysis of the target data of E906 was reported. Table 6.1 shows the results we found compared to previous measured results.

Both of our values for λ agrees with all the results. Our values for μ do not agree with one another to one standard deviation, but both agree with one of the previously reported values for $p + p$ scattering. Our values for ν again disagree with each other, but our $p + p$ data agrees with both previously reported $p + p$

results. It is, however, important to keep in mind that these results do not include systematic uncertainties.

	$p+p$ E866 800 GeV/c	$p+d$ E866 800 GeV/c	$p+p$ E906 120 GeV/c	$p+d$ E906 120 GeV/c	$p+Fe$ QTracker E906 120 GeV/c	$p+p$ QTracker E906 120 GeV/c
λ	0.85 ± 0.10	1.07 ± 0.07	1 [fixed]	1 [fixed]	0.84 ± 0.39	1.08 ± 1.24
μ	-0.03 ± 0.02	0.00 ± 0.01	0.01 ± 0.08	-0.11 ± 0.04	0.12 ± 0.06	-0.13 ± 0.18
ν	0.04 ± 0.015	0.03 ± 0.01	0.21 ± 0.05	0.16 ± 0.04	-0.02 ± 0.02	0.13 ± 0.08

TABLE 6.1: Comparison of various extractions of the Drell-Yan angular dependence variables to our fit. All uncertainties shown are statistical. E866 angular variables are from [10], and non-QTracker E906 angular variables are from [71].

The value of ν is critical for interpreting this data with respect to the Boer-Mulders function. An existence of a non-zero Boer-Mulders function would predict a large value for the ν value in the angular dependence of Drell-Yan scattering.

Our analyses found values consistent with past results in E866, which suggests either that there are nuclear effects that cancel out the Boer-Mulders effect, or that there is no need for a non-zero Boer-Mulders effect for the sea quarks.

From a physical perspective, what this means is that our analyses found no evidence for preferential polarization of the sea quarks inside unpolarize nucleons, either in hydrogen or in iron nuclei. This insight gives a more complete view of the structure of the nucleon sea, which will be further explored in the SpinQuest experiment.

The Lam-Tung relation, which states that $1 - \lambda - 2\nu = 0$, is not violated by our findings. We find a values of 0.12 ± 0.39 and -0.34 ± 1.24, respectively. However, because of the large error bars on λ, these results are also consistent with findings of Lam-Tung relation breaking, such as those by Motyka et. al. [72]

It is important to note that these results are not conclusive. The consistency with zero for the ν value and the adherence to the Lam-Tung relation do not definitively rule out the Boer-Mulders effect or other transverse momentum-dependent distributions. The current analysis is limited by the statistical uncertainties and the systematic uncertainties that have not been fully accounted for.

Future work should focus on acquiring more statistical data to improve the precision of the measurements and on conducting a more careful examination of the systematic uncertainties involved. Reducing these uncertainties will be crucial for

drawing more definitive conclusions about the role of the Boer-Mulders function and other transverse momentum-dependent effects in Drell-Yan processes.

6.1.2 Systematic Uncertainty

Although the comparisons in the last section were limited to statistical error, the question of systematic uncertainty is an important one. Several potential sources of systematic uncertainty in our analysis need additional exploration. These sources include:

- **Detector Efficiency and Resolution**: Variations in the efficiency and resolution of the spectrometer and QTracker can introduce biases in the reconstructed data. The spectrometer's efficiency can vary due to changes in hardware performance, environmental conditions, or operational settings. QTracker's resolution depends on the precise calibration of its parameters and the quality of the input data. The effects of these variations need to be quantified to explore their impact on reconstructed physical quantities.

- **Background Subtraction**: The methods used to subtract background events may have inherent uncertainties that could affect the final results. The algorithms for background subtraction rely on modeling the background distribution and distinguishing it from the signal. Any inaccuracies in the background model or assumptions can introduce systematic biases in the measurement.

- **Monte Carlo Simulations**: The accuracy of the acceptance correction relies on the quality of the MC simulations. Any discrepancies between the simulated and real data can lead to systematic errors. Discrepancies between the simulated events and actual data can arise from incomplete physics models, incorrect detector descriptions, or statistical limitations of the simulations. Regular comparisons between MC simulations and real data are essential to identify and mitigate these systematic effects.

- **The QTracker Algorithm**: The QTracker algorithm might introduce biases if not perfectly calibrated. The performance of the algorithm depends on the training dataset, the architecture of the neural network, and the hyperparameters used. Any major differences between the training data and

the actual experimental conditions can introduce systematic biases in the track reconstruction. Continuous calibration and validation of QTracker with up-to-date experimental data and rigorous cross-validation techniques are necessary to ensure unbiased performance.

These effects are not explored in detail in this dissertation, but will be an important aspect of future work in the QTracker project.

6.2 Future Work

6.2.1 Quality Metric

Developing a robust quality metric for QTracker is essential for ensuring the reliability of the reconstructed data. This metric should assess various aspects of the tracking performance, such as:

- **Track Quality**: How precise the track reconstruction is, that is, how closely the reconstructed track represents the physical particle track.
- **Momentum Resolution**: The precision with which the momentum of the particles is measured.
- **Vertex Resolution**: The accuracy of the reconstructed vertex position.
- **False Positive Probability**: The probabilty that an event that has been identified as a dimuon event was incorrectly identified.

The goal of this metric is to fold all of these values into one Chi-Square-like metric to allow for a one-size-fits-all approach to quality cutting. The current method using a series of stringent cuts can provide high data purity but discards a good amount of useful data.

The process involves defining a set of benchmarks that QTracker must meet or exceed in each of the categories mentioned above. For example, the track quality can be quantified by comparing the reconstructed tracks to the true tracks obtained from Monte Carlo simulations, calculating the deviation, and normalizing it. A similar approach can be applied to the momentum and vertex resolutions.

The false positive probability is particularly important in high-multiplicity environments where the likelihood of hallucinated tracks increases.

The final quality metric will be a composite score derived from these individual assessments. By normalizing and weighting each component appropriately, we achieve a comprehensive metric that encapsulates the overall performance of QTracker on a specific event. This composite metric provides a discrete quality value for each event, enabling straightforward quality cuts and ensuring that only the highest quality tracks are used in subsequent analyses.

Ongoing work involves refining the metric, incorporating more complex event topologies, and validating the results with real data from the SeaQuest and SpinQuest experiments.

6.2.2 Adaptation for E1039

The SpinQuest E1039 experiment at Fermilab began beam and target comissioning in June of 2024. As discussed in Section 4.2.11, QTracker is ideal for adaptation as an online reconstruction algorithm for E1039 due to its high processing speed and accuracy.

To adapt QTracker for E1039, several steps need to be undertaken:

- **Integration with E1039 Data Acquisition**: Ensure that QTracker can seamlessly process data from the E1039 detectors.
- **Calibration and Validation**: Perform calibration and validation using E1039-specific MC simulations and early data to ensure that QTracker meets the experiment's requirements.
- **Systematic Studies**: Conduct detailed studies of systematic uncertainties specific to E1039 and develop mitigation strategies.

Integration with the E1039 data acquisition process is ongoing by the University of Virginia Spin Physics group. Fortunately, the data file formats are shared between E906 and E1039, allowing for minimal friction in this process.

Although extremely similar, there are several differences between the E906 and E1039 experimental setups. The shielding around the target has been changed to adequately protect researchers entering the experimental hall.

Additionally, as discussed extensively in Chapter 3, the target placement and design is radically different between the two experiments. This is especially important as the E1039 target uses a 5T magnetic field to polarize the target material, which has the added effect of bending the trajectories of muons that pass through it. This is a small effect, but should nevertheless be accounted for in Monte Carlo generation.

These changes necessitate fine-tuning of the model with updated Monte Carlo. The position of the target reconstruction network needs to be changed, as the E1039 target is three meters upstream of the beam dump, compared to only 130 cm for the E906 target.

6.2.3 DNN-Based Angular Dependence Extraction

Another potential use for neural networks in the analysis process of Drell-Yan scattering is using models specifically trained to extract the angular dependence variables, λ, μ, and ν. We have developed models to do this and performed a full MC-based extraction, but work is still ongoing to perfect the training process.

This type of method has several advantages over a Chi-Square fit. Neural networks can handle large datasets and complex patterns more efficiently, reducing the impact of noise and enhancing the precision of angular dependence measurements. This subsection outlines the methodology we are developing and the future directions for this approach.

To generate training data, we use the von Neumann rejection sampling method. This technique allows us to create synthetic datasets that replicate real experimental conditions, embedding known angular dependencies.

First, we reconstruct a large number of Drell-Yan, charmonium, and mixed events using QTracker, saving the true kinematic values for the Monte Carlo events. Following that, we randomly select λ, μ, and ν from expected ranges based on previous experimental results. Using these selected angular dependencies, we select approximately 10,000 Drell-Yan events via von Neumann rejection.

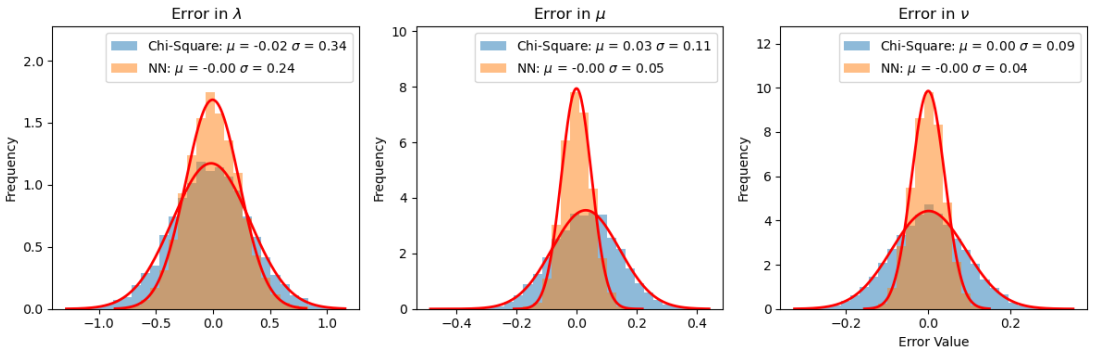


FIGURE 6.1: Absolute error of extraction of angular dependence variables for Chi-Square and Neural-Network based methods.

These Drell-Yan events, with the injected angular dependence, are then combined with the background events from charmonia and random processes, which give us representative samples of events from experimental data. We then perform analysis cuts on all of these events and create ϕ vs. $\cos\theta$ histograms.

These histograms are used to train the neural networks to extract the angular dependencies that we have injected. We designed the neural network as relatively simple Convolutional Neural Networks, but the design is still in progress.

Tests of the full MC Extraction have shown an improvement in reconstruction precision when compared to a Chi-Square extraction. To perform the test, we generated 10,000 angular dependences randomly, using ranges expected based on experimental data. The angular dependences were used to select the Drell-Yan dimuons included in the histograms to be reconstructed. Additionally, we add Monte Carlo dimuons resulting from charmonia processes and experimental background events.

These histograms are then fed into the neural network, which predicts λ, μ, and ν values based on the event prevalences. To test the performance compared to the Chi-Square fit method, we also perform the full background subtraction and acceptance correction discussed in Chapter 5 on each histogram and fit using the Drell-Yan angular differential cross-section.

Figure 6.1 shows the absolute error for the full Monte Carlo extraction of 10,000 injected dependences via both a Chi-Square fit and a Neural Network-based extraction.

The absolute error for all three values are reduced, with an approximately 30% reduction in uncertainty for λ, and over 50% reduction for μ and ν.

The challenging aspect for this method is the precise calibration of the training data, which is highly dependent on covariance and differences between Monte Carlo and experimental data. As such, more work is needed to precisely configure the training data. Future work will focus on optimizing neural network architecture and training procedures to enhance prediction accuracy further. We also plan to conduct systematic studies to ensure the model's robustness across different experimental conditions and datasets.

6.2.4 Applications Beyond SeaQuest and SpinQuest

The successful development and implementation of QTracker has impacts beyond the SeaQuest and SpinQuest experiments. As a neural network-based reconstruction algorithm, QTracker showcases the potential of deep learning techniques to enhance data analysis and interpretation in physics.

The principles and architecture of QTracker can be adapted for use in other particle tracking applications. Many experiments across physics disciplines require precise tracking of charged particles through complex detector environments. Traditional tracking algorithms can struggle with high track multiplicity and noise present in these environments. By integrating deep learning models like QTracker, future experiments can achieve more accurate and efficient particle track reconstruction, leading to better resolution of particle interactions and decay processes.

The use of neural networks for event filtering and track finding also offers a powerful tool for data preprocessing in large-scale experiments. With the increasing volume of data generated by modern detectors, efficient filtering of irrelevant or noisy events is crucial for optimizing data storage and analysis pipelines. Neural networks trained on representative datasets can perform this task with high accuracy, reducing the computational load and improving the overall quality of the data used for physics analysis.

Finally, the experience and knowledge gained from developing QTracker can inform the design of future experiments. As particle physics experiments become increasingly complex, the need for advanced data analysis tools will continue to grow. The successful integration of deep learning techniques can serve as a blueprint for

developing new algorithms and analysis frameworks tailored to the specific needs of upcoming experiments.

6.3 Conclusion

This dissertation has explored the implementation and impact of QTracker, a novel deep neural network-based reconstruction algorithm, within the context of the SeaQuest and SpinQuest experiments. Using QTracker, we analyzed the angular dependence of Drell-Yan scattering incident on both the liquid hydrogen target and the iron beam dump.

The angular distribution analyses revealed a relatively small $\cos 2\theta$ dependence of -0.02 ± 0.02 for $p + Fe$ scattering, and 0.13 ± 0.08 for $p + p$ scattering. These values are significant as they relate directly to the Boer-Mulders function, which describes the correlation between the transverse spin and transverse momentum of quarks inside the nucleon. Our findings support prior observations that sea quarks exhibit a smaller Boer-Mulders function compared to valence quarks. Specifically, the small $\cos 2\theta$ dependence observed for the $p + Fe$ scattering suggests that the sea quarks within the iron target are less polarized relative to the nucleon's motion. In contrast, the larger dependence seen in $p + p$ scattering aligns with the understanding that valence quarks exhibit more pronounced polarization effects.

These results contribute to the global dataset on Boer-Mulders functions and Drell-Yan angular distributions, enhancing our understanding of the intrinsic properties of quarks. The low $\cos 2\theta$ dependence implies a small associated Boer-Mulders function, and by extension, that within an unpolarized nucleon, the sea quarks are not preferentially aligned in relation to the nucleon's movement. This contrasts with the behavior of valence quarks, which show significant preferential polarization, reflecting the complex internal dynamics of the nucleon.

The development and application of QTracker have been pivotal in achieving these insights. Traditional reconstruction methods often struggle with the complex data produced in fixed-target experimental environments. By integrating advanced deep learning techniques, QTracker has addressed these challenges, enhancing the quality of track reconstruction and providing more precise measurements of key variables. The AI-driven approach of QTracker allowed for rapid and accurate

data analysis, maximizing the statistical power of our measurements and enabling the detailed extraction of angular distributions in Drell-Yan scattering.

The successful integration of machine learning into the particle tracking process not only impacts the SeaQuest and SpinQuest experiments but also sets a precedent for future physics research. The methodologies and results presented in this dissertation illustrate the potential of combining physics with cutting-edge computational techniques. As new experiments and detectors come online, the insights gained from the implementation of QTracker in SeaQuest and SpinQuest will inform and enhance the design and analysis strategies of future studies.

Future work will focus on the continued refinement of QTracker for use in the SpinQuest experiment, which aims to probe the spin structure of the nucleon in greater detail via a polarized target. By leveraging QTracker's advanced reconstruction capabilities, SpinQuest can achieve higher precision in measuring the transverse spin effects and further elucidate the role of sea quarks. Additionally, there is significant potential to adapt QTracker for use in other experiments with different experimental setups. This adaptability opens the door to a wide array of possibilities in experimental nuclear and particle physics. Continued development in this area will be fruitful, as the implementation of machine learning along with traditional physics methodologies can drive future advancements in our understanding of fundamental particles and their interactions.

In conclusion, this dissertation not only highlights the successful application of a deep neural network-based reconstruction algorithm but also demonstrates the impact that innovative computational techniques can have on advancing our understanding of the subatomic world.

Appendix A

Code

This appendix contains all of the code used to create, train, and use QTracker as of July 2024. Code has been slightly edited to fit on the page – indents may need to be fixed before the scripts are able to be executed.

A.1 Network Creation

```
import tensorflow as tf

#Event Filter

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(54, 201, 1)),
    tf.keras.layers.Conv2D(96, (11, 11), strides=(4, 4),
    padding='valid', activation='relu'),
    tf.keras.layers.MaxPooling2D((1, 6), strides=(2, 2),
    padding='valid'),
    tf.keras.layers.Conv2D(256, (5, 5), strides=(1, 1),
    padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((1, 6), strides=(2, 2),
    padding='valid'),
    tf.keras.layers.Conv2D(384, (3, 3), strides=(1, 1),
    padding='same', activation='relu'),
    tf.keras.layers.Conv2D(384, (3, 3), strides=(1, 1),
    padding='same', activation='relu'),
    tf.keras.layers.Conv2D(384, (3, 3), strides=(1, 1),
    padding='same', activation='relu'),
    tf.keras.layers.Conv2D(384, (3, 3), strides=(1, 1),
    padding='same', activation='relu'),
    tf.keras.layers.Conv2D(256, (3, 3), strides=(1, 1),
    padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((1, 6), strides=(2, 2),
    padding='valid'),
```

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(2)
])

# Save the model
model.save('Networks/event_filter')


#Track Finder Networks
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu',
        input_shape=(54,201,1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(2048, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(34, activation='linear')
    ])
#Save the individual muon track finders.
model.save('Networks/Track_Finder_Pos')
model.save('Networks/Track_Finder_Neg')

#Change the output layer to shape 68 to make it work for dimuon track finding.
model.pop()  # Remove the final layer
model.add(tf.keras.layers.Dense(68, activation='linear'))

#Save the dimuon track finders.
model.save('Networks/Track_Finder_All')
model.save('Networks/Track_Finder_Z')
```

```
model.save('Networks/Track_Finder_Target')
model.save('Networks/Track_Finder_Dump')

# Define Kinematic Reconstruction Networks
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(68,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dense(256,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(16,activation='relu'),
    tf.keras.layers.Dense(6)])

#Save a copy each for all vertices, z vertices, and for target vertices.
model.save('Networks/Reconstruction_All')
model.save('Networks/Reconstruction_Z')
model.save('Networks/Reconstruction_Target')
model.save('Networks/Reconstruction_Dump')

#Define the single muon vertex finding networks.
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(34,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dense(256,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(16,activation='relu'),
    tf.keras.layers.Dense(1)])

#Save the individual muon vertex finders.
model.save('Networks/Vertexing_Pos')
model.save('Networks/Vertexing_Neg')

#Define the dimuon vertex finding networks.
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(71,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dense(256,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(16,activation='relu'),
    tf.keras.layers.Dense(3)])

#Save a copy each for all vertices and for Z vertices.
model.save('Networks/Vertexing_All')
model.save('Networks/Vertexing_Z')


# Define the target-dump filter network.
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(757,)),
    tf.keras.layers.Dense(2048, activation='relu'),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(2)
])

#Save the model for training.
model.save('Networks/target_dump_filter')
```

LISTING A.1: Network defining code

A.2 Training

A.2.1 Shared Functions

```
    import numpy as np
import uproot
from numba import njit, prange
import random

#These are useful variables.
kin_means = np.array([2,0,35,-2,0,35])
kin_stds = np.array([0.6,1.2,10,0.6,1.2,10])

vertex_means=np.array([0,0,-300])
vertex_stds=np.array([10,10,300])

means = np.concatenate((kin_means,vertex_means))
stds = np.concatenate((kin_stds,vertex_stds))


max_ele = [200, 200, 168, 168, 200, 200, 128, 128,  112,  112, 128, 128, 134,
    134,
            112, 112, 134, 134,  20,  20,  16,  16,  16,  16,  16,  16,
        72,  72,  72,  72,  72,  72,  72,  72, 200, 200, 168, 168, 200, 200,
        128, 128,  112,  112, 128, 128, 134, 134, 112, 112, 134, 134,
        20,  20,  16,  16,  16,  16,  16,  16,  72,  72,  72,  72,  72,
        72,  72,  72]


#This Function takes the Track_QA_v2 format Root files and creates arrays
#for positive and negative element ids, drift, as well as kinematics.
def read_root_file(root_file):
```

```
print("Reading ROOT file...")
targettree = uproot.open(root_file+':QA_ana')
targetevents=len(targettree['n_tracks'].array(library='np'))
D0U_ele = targettree['D0U_ele'].array(library='np')
D0Up_ele = targettree['D0Up_ele'].array(library='np')
D0X_ele = targettree['D0X_ele'].array(library='np')
D0Xp_ele = targettree['D0Xp_ele'].array(library='np')
D0V_ele = targettree['D0V_ele'].array(library='np')
D0Vp_ele = targettree['D0Vp_ele'].array(library='np')

D2U_ele = targettree['D2U_ele'].array(library='np')
D2Up_ele = targettree['D2Up_ele'].array(library='np')
D2X_ele = targettree['D2X_ele'].array(library='np')
D2Xp_ele = targettree['D2Xp_ele'].array(library='np')
D2V_ele = targettree['D2V_ele'].array(library='np')
D2Vp_ele = targettree['D2Vp_ele'].array(library='np')

D3pU_ele = targettree['D3pU_ele'].array(library='np')
D3pUp_ele = targettree['D3pUp_ele'].array(library='np')
D3pX_ele = targettree['D3pX_ele'].array(library='np')
D3pXp_ele = targettree['D3pXp_ele'].array(library='np')
D3pV_ele = targettree['D3pV_ele'].array(library='np')
D3pVp_ele = targettree['D3pVp_ele'].array(library='np')

D3mU_ele = targettree['D3mU_ele'].array(library='np')
D3mUp_ele = targettree['D3mUp_ele'].array(library='np')
D3mX_ele = targettree['D3mX_ele'].array(library='np')
D3mXp_ele = targettree['D3mXp_ele'].array(library='np')
D3mV_ele = targettree['D3mV_ele'].array(library='np')
D3mVp_ele = targettree['D3mVp_ele'].array(library='np')

D0U_drift = targettree['D0U_drift'].array(library='np')
D0Up_drift = targettree['D0Up_drift'].array(library='np')
D0X_drift = targettree['D0X_drift'].array(library='np')
D0Xp_drift = targettree['D0Xp_drift'].array(library='np')
D0V_drift = targettree['D0V_drift'].array(library='np')
D0Vp_drift = targettree['D0Vp_drift'].array(library='np')

D2U_drift = targettree['D2U_drift'].array(library='np')
D2Up_drift = targettree['D2Up_drift'].array(library='np')
D2X_drift = targettree['D2X_drift'].array(library='np')
D2Xp_drift = targettree['D2Xp_drift'].array(library='np')
D2V_drift = targettree['D2V_drift'].array(library='np')
D2Vp_drift = targettree['D2Vp_drift'].array(library='np')

D3pU_drift = targettree['D3pU_drift'].array(library='np')
D3pUp_drift = targettree['D3pUp_drift'].array(library='np')
D3pX_drift = targettree['D3pX_drift'].array(library='np')
D3pXp_drift = targettree['D3pXp_drift'].array(library='np')
D3pV_drift = targettree['D3pV_drift'].array(library='np')
D3pVp_drift = targettree['D3pVp_drift'].array(library='np')

D3mU_drift = targettree['D3mU_drift'].array(library='np')
D3mUp_drift = targettree['D3mUp_drift'].array(library='np')
D3mX_drift = targettree['D3mX_drift'].array(library='np')
```

```
D3mXp_drift = targettree['D3mXp_drift'].array(library='np')
D3mV_drift = targettree['D3mV_drift'].array(library='np')
D3mVp_drift = targettree['D3mVp_drift'].array(library='np')

H1B_ele = targettree['H1B_ele'].array(library='np')
H1T_ele = targettree['H1T_ele'].array(library='np')
H1L_ele = targettree['H1L_ele'].array(library='np')
H1R_ele = targettree['H1R_ele'].array(library='np')

H2L_ele = targettree['H2L_ele'].array(library='np')
H2R_ele = targettree['H2R_ele'].array(library='np')
H2B_ele = targettree['H2B_ele'].array(library='np')
H2T_ele = targettree['H2T_ele'].array(library='np')

H3B_ele = targettree['H3B_ele'].array(library='np')
H3T_ele = targettree['H3T_ele'].array(library='np')

H4Y1L_ele = targettree['H4Y1L_ele'].array(library='np')
H4Y1R_ele = targettree['H4Y1R_ele'].array(library='np')
H4Y2L_ele = targettree['H4Y2L_ele'].array(library='np')
H4Y2R_ele = targettree['H4Y2R_ele'].array(library='np')
H4B_ele = targettree['H4B_ele'].array(library='np')
H4T_ele = targettree['H4T_ele'].array(library='np')

P1Y1_ele = targettree['P1Y1_ele'].array(library='np')
P1Y2_ele = targettree['P1Y2_ele'].array(library='np')
P1X1_ele = targettree['P1X1_ele'].array(library='np')
P1X2_ele = targettree['P1X2_ele'].array(library='np')

P2X1_ele = targettree['P2X1_ele'].array(library='np')
P2X2_ele = targettree['P2X2_ele'].array(library='np')
P2Y1_ele = targettree['P2Y1_ele'].array(library='np')
P2Y2_ele = targettree['P2Y2_ele'].array(library='np')

gpx = targettree['gpx'].array(library='np')
gpy = targettree['gpy'].array(library='np')
gpz = targettree['gpz'].array(library='np')
gvx = targettree['gvx'].array(library='np')
gvy = targettree['gvy'].array(library='np')
gvz = targettree['gvz'].array(library='np')

pid = targettree['pid'].array(library='np')

print('Done')

#This reads the dimuon tracks into an array
pos_events=np.zeros((targetevents,54))
pos_drift = np.zeros((targetevents,30))
pos_kinematics = np.zeros((targetevents,6))
neg_events=np.zeros((targetevents,54))
neg_drift = np.zeros((targetevents,30))
neg_kinematics = np.zeros((targetevents,6))
print("Reading events...")
for j in range(targetevents):
    #Determine which hits correspond to positive or negative muons
```

```
first=pid[j][0]
if(first>0):
    pos=0
    neg=1
else:
    pos=1
    neg=0
pos_kinematics[j][0] = gpx[j][pos]
pos_kinematics[j][1] = gpy[j][pos]
pos_kinematics[j][2] = gpz[j][pos]
pos_kinematics[j][3] = gvx[j][pos]
pos_kinematics[j][4] = gvy[j][pos]
pos_kinematics[j][5] = gvz[j][pos]
neg_kinematics[j][0] = gpx[j][neg]
neg_kinematics[j][1] = gpy[j][neg]
neg_kinematics[j][2] = gpz[j][neg]
neg_kinematics[j][3] = gvx[j][neg]
neg_kinematics[j][4] = gvy[j][neg]
neg_kinematics[j][5] = gvz[j][neg]
pos_events[j][0]=D0U_ele[j][pos]
neg_events[j][0]=D0U_ele[j][neg]
pos_events[j][1]=D0Up_ele[j][pos]
neg_events[j][1]=D0Up_ele[j][neg]
pos_events[j][2]=D0X_ele[j][pos]
neg_events[j][2]=D0X_ele[j][neg]
pos_events[j][3]=D0Xp_ele[j][pos]
neg_events[j][3]=D0Xp_ele[j][neg]
pos_events[j][4]=D0V_ele[j][pos]
neg_events[j][4]=D0V_ele[j][neg]
pos_events[j][5]=D0Vp_ele[j][pos]
neg_events[j][5]=D0Vp_ele[j][neg]
pos_events[j][16]=D2U_ele[j][pos]
neg_events[j][16]=D2U_ele[j][neg]
pos_events[j][17]=D2Up_ele[j][pos]
neg_events[j][17]=D2Up_ele[j][neg]
pos_events[j][15]=D2X_ele[j][pos]
neg_events[j][15]=D2X_ele[j][neg]
pos_events[j][14]=D2Xp_ele[j][pos]
neg_events[j][14]=D2Xp_ele[j][neg]
pos_events[j][12]=D2V_ele[j][pos]
neg_events[j][12]=D2V_ele[j][neg]
pos_events[j][13]=D2Vp_ele[j][pos]
neg_events[j][13]=D2Vp_ele[j][neg]
pos_events[j][23]=D3pU_ele[j][pos]
neg_events[j][23]=D3pU_ele[j][neg]
pos_events[j][22]=D3pUp_ele[j][pos]
neg_events[j][22]=D3pUp_ele[j][neg]
pos_events[j][21]=D3pX_ele[j][pos]
neg_events[j][21]=D3pX_ele[j][neg]
pos_events[j][20]=D3pXp_ele[j][pos]
neg_events[j][20]=D3pXp_ele[j][neg]
pos_events[j][19]=D3pV_ele[j][pos]
neg_events[j][19]=D3pV_ele[j][neg]
pos_events[j][18]=D3pVp_ele[j][pos]
neg_events[j][18]=D3pVp_ele[j][neg]
```

```
pos_events[j][29]=D3mU_ele[j][pos]
neg_events[j][29]=D3mU_ele[j][neg]
pos_events[j][28]=D3mUp_ele[j][pos]
neg_events[j][28]=D3mUp_ele[j][neg]
pos_events[j][27]=D3mX_ele[j][pos]
neg_events[j][27]=D3mX_ele[j][neg]
pos_events[j][26]=D3mXp_ele[j][pos]
neg_events[j][26]=D3mXp_ele[j][neg]
pos_events[j][25]=D3mV_ele[j][pos]
neg_events[j][25]=D3mV_ele[j][neg]
pos_events[j][24]=D3mVp_ele[j][pos]
neg_events[j][24]=D3mVp_ele[j][neg]
pos_events[j][30]=H1B_ele[j][pos]
neg_events[j][30]=H1B_ele[j][neg]
pos_events[j][31]=H1T_ele[j][pos]
neg_events[j][31]=H1T_ele[j][neg]
pos_events[j][32]=H1L_ele[j][pos]
neg_events[j][32]=H1L_ele[j][neg]
pos_events[j][33]=H1R_ele[j][pos]
neg_events[j][33]=H1R_ele[j][neg]
pos_events[j][34]=H2L_ele[j][pos]
neg_events[j][34]=H2L_ele[j][neg]
pos_events[j][35]=H2R_ele[j][pos]
neg_events[j][35]=H2R_ele[j][neg]
pos_events[j][36]=H2T_ele[j][pos]
neg_events[j][36]=H2T_ele[j][neg]
pos_events[j][37]=H2B_ele[j][pos]
neg_events[j][37]=H2B_ele[j][neg]
pos_events[j][38]=H3B_ele[j][pos]
neg_events[j][38]=H3B_ele[j][neg]
pos_events[j][39]=H3T_ele[j][pos]
neg_events[j][39]=H3T_ele[j][neg]
pos_events[j][40]=H4Y1L_ele[j][pos]
neg_events[j][40]=H4Y1L_ele[j][neg]
pos_events[j][41]=H4Y1R_ele[j][pos]
neg_events[j][41]=H4Y1R_ele[j][neg]
pos_events[j][42]=H4Y2L_ele[j][pos]
neg_events[j][42]=H4Y2L_ele[j][neg]
pos_events[j][43]=H4Y2R_ele[j][pos]
neg_events[j][43]=H4Y2R_ele[j][neg]
pos_events[j][44]=H4B_ele[j][pos]
neg_events[j][44]=H4B_ele[j][neg]
pos_events[j][45]=H4T_ele[j][pos]
neg_events[j][45]=H4T_ele[j][neg]
pos_events[j][46]=P1Y1_ele[j][pos]
neg_events[j][46]=P1Y1_ele[j][neg]
pos_events[j][47]=P1Y2_ele[j][pos]
neg_events[j][47]=P1Y2_ele[j][neg]
pos_events[j][48]=P1X1_ele[j][pos]
neg_events[j][48]=P1X1_ele[j][neg]
pos_events[j][49]=P1X2_ele[j][pos]
neg_events[j][49]=P1X2_ele[j][neg]
pos_events[j][50]=P2X1_ele[j][pos]
neg_events[j][50]=P2X1_ele[j][neg]
pos_events[j][51]=P2X2_ele[j][pos]
```

```
        neg_events[j][51]=P2X2_ele[j][neg]
        pos_events[j][52]=P2Y1_ele[j][pos]
        neg_events[j][52]=P2Y1_ele[j][neg]
        pos_events[j][53]=P2Y2_ele[j][pos]
        neg_events[j][53]=P2Y2_ele[j][neg]
        pos_drift[j][0]=D0U_drift[j][pos]
        neg_drift[j][0]=D0U_drift[j][neg]
        pos_drift[j][1]=D0Up_drift[j][pos]
        neg_drift[j][1]=D0Up_drift[j][neg]
        pos_drift[j][2]=D0X_drift[j][pos]
        neg_drift[j][2]=D0X_drift[j][neg]
        pos_drift[j][3]=D0Xp_drift[j][pos]
        neg_drift[j][3]=D0Xp_drift[j][neg]
        pos_drift[j][4]=D0V_drift[j][pos]
        neg_drift[j][4]=D0V_drift[j][neg]
        pos_drift[j][5]=D0Vp_drift[j][pos]
        neg_drift[j][5]=D0Vp_drift[j][neg]
        pos_drift[j][16]=D2U_drift[j][pos]
        neg_drift[j][16]=D2U_drift[j][neg]
        pos_drift[j][17]=D2Up_drift[j][pos]
        neg_drift[j][17]=D2Up_drift[j][neg]
        pos_drift[j][15]=D2X_drift[j][pos]
        neg_drift[j][15]=D2X_drift[j][neg]
        pos_drift[j][14]=D2Xp_drift[j][pos]
        neg_drift[j][14]=D2Xp_drift[j][neg]
        pos_drift[j][12]=D2V_drift[j][pos]
        neg_drift[j][12]=D2V_drift[j][neg]
        pos_drift[j][13]=D2Vp_drift[j][pos]
        neg_drift[j][13]=D2Vp_drift[j][neg]
        pos_drift[j][23]=D3pU_drift[j][pos]
        neg_drift[j][23]=D3pU_drift[j][neg]
        pos_drift[j][22]=D3pUp_drift[j][pos]
        neg_drift[j][22]=D3pUp_drift[j][neg]
        pos_drift[j][21]=D3pX_drift[j][pos]
        neg_drift[j][21]=D3pX_drift[j][neg]
        pos_drift[j][20]=D3pXp_drift[j][pos]
        neg_drift[j][20]=D3pXp_drift[j][neg]
        pos_drift[j][19]=D3pV_drift[j][pos]
        neg_drift[j][19]=D3pV_drift[j][neg]
        pos_drift[j][18]=D3pVp_drift[j][pos]
        neg_drift[j][18]=D3pVp_drift[j][neg]
        pos_drift[j][29]=D3mU_drift[j][pos]
        neg_drift[j][29]=D3mU_drift[j][neg]
        pos_drift[j][28]=D3mUp_drift[j][pos]
        neg_drift[j][28]=D3mUp_drift[j][neg]
        pos_drift[j][27]=D3mX_drift[j][pos]
        neg_drift[j][27]=D3mX_drift[j][neg]
        pos_drift[j][26]=D3mXp_drift[j][pos]
        neg_drift[j][26]=D3mXp_drift[j][neg]
        pos_drift[j][25]=D3mV_drift[j][pos]
        neg_drift[j][25]=D3mV_drift[j][neg]
        pos_drift[j][24]=D3mVp_drift[j][pos]
        neg_drift[j][24]=D3mVp_drift[j][neg]
print("Done")
```

```
    return pos_events, pos_drift, pos_kinematics, neg_events, neg_drift,
    neg_kinematics

#Removes the high values that Track_QA_v2 uses to initialize arrays.
@njit(parallel=True)
def clean(events):
    for j in prange(len(events)):
        for i in prange(54):
            if(events[j][i]>1000):
                events[j][i]=0
    return events


#This function is used to inject NIM3 partial tracks and random hits into the
    hit matrix.
#Modify this path to point to collection of NIM3 data.
nim3_path = '/project/ptgroup/QTracker_Training/NIM3/'
@njit()
def hit_matrix(detectorid,elementid,drifttime,hits,drift,station):
    for j in range (len(detectorid)):
        rand = random.random()
        #St 1
        if(station==1) and (rand<0.85):
            if ((detectorid[j]<7) or (detectorid[j]>30)) and (detectorid[j]<35):
                hits[int(detectorid[j])-1][int(elementid[j]-1)]=1
                drift[int(detectorid[j])-1][int(elementid[j]-1)]=drifttime[j]
        #St 2
        elif(station==2):
            if (detectorid[j]>12 and (detectorid[j]<19)) or
                ((detectorid[j]>34) and (detectorid[j]<39)):
                if((detectorid[j]<15) and (rand<0.76)) or ((detectorid[j]>14)
    and (rand<0.86)) or (detectorid[j]==17):
                    hits[int(detectorid[j])-1][int(elementid[j]-1)]=1
                    drift[int(detectorid[j])-1][int(elementid[j]-1)]=drifttime[j]
        #St 3
        elif(station==3) and (rand<0.8):
            if (detectorid[j]>18 and (detectorid[j]<31)) or
                ((detectorid[j]==39) or (detectorid[j]==40)):
                hits[int(detectorid[j])-1][int(elementid[j]-1)]=1
                drift[int(detectorid[j])-1][int(elementid[j]-1)]=drifttime[j]
        #St 4
        elif(station==4):
            if ((detectorid[j]>39) and (detectorid[j]<55)):
                hits[int(detectorid[j])-1][int(elementid[j]-1)]=1
                drift[int(detectorid[j])-1][int(elementid[j]-1)]=drifttime[j]
    return hits,drift

#This function builds the realistic background of messy hit matrix events.
#This can be modified to create different background configurations
def build_background(n_events):
    filelist=['output_part1.root:tree_nim3','output_part2.root:tree_nim3',
    'output_part3.root:tree_nim3','output_part4.root:tree_nim3',
    'output_part5.root:tree_nim3','output_part6.root:tree_nim3',
    'output_part7.root:tree_nim3','output_part8.root:tree_nim3',
    'output_part9.root:tree_nim3']
```

```
    targettree = uproot.open(nim3_path+random.choice(filelist))
    detectorid_nim3=targettree["det_id"].arrays(library="np")["det_id"]
    elementid_nim3=targettree["ele_id"].arrays(library="np")["ele_id"]

    driftdistance_nim3=targettree["drift_dist"].arrays(library="np")["drift_dist"]
    hits = np.zeros((n_events,54,201))
    drift = np.zeros((n_events,54,201))
    for n in range (n_events): #Create NIM3 events
        g=random.choice([1,2,3,4,5,6])#Creates realistic occupancies for FPGA-1
    events.
        for m in range(g):
            i=random.randrange(len(detectorid_nim3))
            hits[n],drift[n]=hit_matrix(detectorid_nim3[i],elementid_nim3[i]
                ,driftdistance_nim3[i],hits[n],drift[n],1)
            i=random.randrange(len(detectorid_nim3))
            hits[n],drift[n]=hit_matrix(detectorid_nim3[i],elementid_nim3[i],
                driftdistance_nim3[i],hits[n],drift[n],2)
            i=random.randrange(len(detectorid_nim3))
            hits[n],drift[n]=hit_matrix(detectorid_nim3[i],elementid_nim3[i],
                driftdistance_nim3[i],hits[n],drift[n],3)
            i=random.randrange(len(detectorid_nim3))
            hits[n],drift[n]=hit_matrix(detectorid_nim3[i],elementid_nim3[i],
                driftdistance_nim3[i],hits[n],drift[n],4)
    del detectorid_nim3, elementid_nim3,driftdistance_nim3
    return hits, drift

# Function to evaluate the Track Finder neural network and match to drift.
@njit(parallel=True)
def evaluate_finder(testin, testdrift, predictions):
    reco_in = np.zeros((len(testin), 68, 3))

    def process_entry(i, dummy, j_offset):
        j = dummy if dummy <= 5 else dummy + 6
        if dummy > 11:
            if predictions[i][12+j_offset] > 0:
                j = dummy + 6
            elif predictions[i][12+j_offset] < 0:
                j = dummy + 12

        if dummy > 17:
            if(predictions[i][2 * (dummy - 18) + 30 + j_offset] > 0):
                j = 2 * (dummy - 18) + 30
            else: 2 * (dummy - 18) + 31

        if dummy > 25:
            j = dummy + 20

        k = abs(predictions[i][dummy + j_offset])
        sign = k / predictions[i][dummy + j_offset] if k > 0 else 1
        if(dummy<6):window=15
        elif(dummy<12):window=5
        elif(dummy<18):window=5
        elif(dummy<26):window=1
        else:window=3
        k_sum = np.sum(testin[i][j][k - window:k + window-1])
```

```
        if k_sum > 0 and ((dummy < 18) or (dummy > 25)):
            k_temp = k
            n = 1
            while testin[i][j][k - 1] == 0:
                k_temp += n
                n = -n * (abs(n) + 1) / abs(n)
                if 0 <= k_temp < 201:
                    k = int(k_temp)

        reco_in[i][dummy + j_offset][0] = sign * k
        reco_in[i][dummy + j_offset][1] = testdrift[i][j][k - 1]
        if(testin[i][j][k - 1]==1):
            reco_in[i][dummy + j_offset][2]=1

    for i in prange(predictions.shape[0]):
        for dummy in prange(34):
            process_entry(i, dummy, 0)

        for dummy in prange(34):
            process_entry(i, dummy, 34)

    return reco_in

# Drift chamber mismatch calculation
def calc_mismatches(track):
    results = []
    for pos_slice, neg_slice in [(slice(0,6),slice(34,40)),(slice(6,12),
                                 slice(40,46)), (slice(12, 18), slice(46, 52))]:
        results.extend([
        np.sum(abs(track[:,pos_slice,::2,0] - track[:,pos_slice,1::2,0]) >
    1,axis=1),
        np.sum(abs(track[:,neg_slice,::2,0] - track[:,neg_slice,1::2,0]) >
    1,axis=1),
        np.sum(abs(track[:,pos_slice,:,2])== 0,axis=1),
        np.sum(abs(track[:,neg_slice,:,2])== 0,axis=1)
        ])
    return np.array(results)
```

Listing A.2: Common functions for training

A.2.2 Event Filter

```
import os
import numpy as np
import uproot
from numba import njit, prange
import random
import tensorflow as tf
import gc
from Common_Functions import *

@njit(parallel=True)
```

```
def track_injection(hits,pos_e,neg_e):
    # Inject tracks into the hit matrices
    category=np.zeros((len(hits)))
    for z in prange(len(hits)):
        m = random.randrange(0,2)
        j=random.randrange(len(pos_e))
        for k in range(54):
            if(pos_e[j][k]>0):
                if(random.random()<m*0.94) or ((k>29)&(k<45)):
                    hits[z][k][int(pos_e[j][k]-1)]=1
            if(neg_e[j][k]>0):
                if(random.random()<m*0.94) or ((k>29)&(k<45)):
                    hits[z][k][int(neg_e[j][k]-1)]=1
        category[z]=m

    return hits,category

def generate_hit_matrices(n_events, tvt):
    #Create the realistic background for events
    hits, _ = build_background(n_events)
    #Inject the reconstructable tracks
    if(tvt=="Train"):
        hits,category=track_injection(hits,pos_events,neg_events)
    if(tvt=="Val"):
        hits,category=track_injection(hits,pos_events_val,neg_events_val)
    return hits.astype(bool), category.astype(int)

# Read training and validation data from ROOT files
pos_events, pos_drift, pos_kinematics,
    neg_events, neg_drift, neg_kinematics =
        read_root_file('Root_Files/Target_Train_QA_v2.root')
pos_events_val, pos_drift_val, pos_kinematics_val,
    neg_events_val, neg_drift_val, neg_kinematics_val =
        read_root_file('Root_Files/Target_Val_QA_v2.root')

del pos_drift, neg_drift, pos_kinematics, neg_kinematics
del pos_drift_val, neg_drift_val, pos_kinematics_val, neg_kinematics_val

# Clean event data by setting values > 1000 to 0.
pos_events=clean(pos_events).astype(int)
neg_events=clean(neg_events).astype(int)
pos_events_val=clean(pos_events_val).astype(int)
neg_events_val=clean(neg_events_val).astype(int)

# Set learning rate and callback for early stopping
learning_rate_filter = 1e-6
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
    restore_best_weights=True)
n_train = 0

# Detect the number of GPUs available
gpus = tf.config.experimental.list_physical_devices('GPU')
num_gpus = len(gpus)
print(f"Number of GPUs available: {num_gpus}")
```

```
# Set up strategy for distributed training
if num_gpus > 1:
    strategy = tf.distribute.MirroredStrategy()
else:
    strategy = tf.distribute.get_strategy()

# Adjust batch size for the number of GPUs
batch_size_adjusted = 256 * num_gpus

print("Before while loop:", n_train)
while n_train < 1e7:
    # Generate training and validation data
    trainin, trainsignals = generate_hit_matrices(1000000, "Train")
    n_train += len(trainin)
    print("Generated Training Data")
    valin, valsignals = generate_hit_matrices(100000, "Val")
    print("Generated Validation Data")

    # Clear session and reset TensorFlow graph
    tf.keras.backend.clear_session()
    gc.collect()
    with strategy.scope():
        # Load and compile the model
        model = tf.keras.models.load_model('Networks/event_filter')
        optimizer = tf.keras.optimizers.Adam(learning_rate_filter)
        model.compile(optimizer=optimizer,
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])

        # Evaluate the model before training
        val_loss_before = model.evaluate(valin, valsignals,
            batch_size=batch_size_adjusted, verbose=2)[0]

        # Train the model
        history = model.fit(trainin, trainsignals,
                    epochs=1000, batch_size=batch_size_adjusted, verbose=2,
                    validation_data=(valin, valsignals), callbacks=[callback])

        # Check if the validation loss improved
        if min(history.history['val_loss']) < val_loss_before:
            model.save('Networks/event_filter')
            learning_rate_filter *= 2
        learning_rate_filter /= 2

    del trainsignals, trainin, valin, valsignals, model
    gc.collect()
    print(n_train)
```

Listing A.3: Event filter training.

A.2.3 Track Finder

```
import numpy as np
import uproot
from numba import njit, prange
import random
import tensorflow as tf
import gc
import sys
from Common_Functions import *

if len(sys.argv) != 2:
        print("Usage: python script.py <Charge or Vertex>")
        print("Options are Pos, Neg, All, Z, Target, or Dump")
        exit(1)

opt = sys.argv[1]

if(opt == 'Pos') or (opt == 'Neg'):
    single_muon=True

root_file_train = f"Root_Files/{opt}_Train_QA_v2.root"
root_file_val = f"Root_Files/{opt}_Val_QA_v2.root"
if single_muon:
    root_file_train = f"Root_Files/Z_Train_QA_v2.root"
    root_file_val = f"Root_Files/Z_Val_QA_v2.root"

model_name = f"Networks/Track_Finder_{opt}"

pos_events, pos_drift, pos_kinematics, neg_events, neg_drift, neg_kinematics =
    read_root_file(root_file_train)
pos_events_val, pos_drift_val, pos_kinematics_val, neg_events_val,
    neg_drift_val, neg_kinematics_val = read_root_file(root_file_val)

del pos_drift, neg_drift, pos_kinematics, neg_kinematics
del pos_drift_val, neg_drift_val, pos_kinematics_val, neg_kinematics_val

pos_events=clean(pos_events).astype(int)
neg_events=clean(neg_events).astype(int)
pos_events_val=clean(pos_events_val).astype(int)
neg_events_val=clean(neg_events_val).astype(int)

@njit(parallel=True)
def track_injection(hits, pos_e, neg_e):
    n_events = len(hits)
    track_real = np.zeros((n_events, 68), dtype=np.float32)
    for z in prange(n_events):
        #Randomly choose one positive and one negative event
        j = np.random.randint(len(pos_e))
        if single_muon:j2 = np.random.randint(len(neg_e))
        else: j2=j
        for k in range(54):
            pos_val = pos_e[j][k]
            neg_val = neg_e[j2][k]
            if pos_val > 0 and (np.random.random() < 0.94 or k > 29):
                hits[z][k][int(pos_val - 1)] = 1
```

```
            if neg_val > 0 and (np.random.random() < 0.94 or k > 29):
                hits[z][k][int(neg_val - 1)] = 1
        # Convert the hits into tracks to be reconstructed.
        track_real[z, :6] = pos_e[j, :6]
        track_real[z, 6:12] = pos_e[j, 12:18]
        track_real[z, 34:40] = neg_e[j2, :6]
        track_real[z, 40:46] = neg_e[j2, 12:18]
        # St. 3p gets positive values, St. 3m gets negative values.
        track_real[z, 12:18] = np.where((pos_e[j, 18]) > 0, pos_e[j, 18:24],
    -pos_e[j, 24:30])
        track_real[z, 46:52] = np.where(neg_e[j2, 18] > 0, neg_e[j2, 18:24],
    -neg_e[j2, 24:30])
        # Pairs of hodoscopes are mutually exclusive,
        #this gives positive or negative values depending on the array.
        track_real[z, 18:26] = np.where(pos_e[j, 30:45:2] > 0, pos_e[j,
    30:45:2], -pos_e[j, 31:46:2])
        track_real[z, 52:60] = np.where(neg_e[j2, 30:45:2] > 0 , neg_e[j2,
    30:45:2], -neg_e[j2, 31:46:2])
        track_real[z, 26:34] = pos_e[j, 46:54]
        track_real[z, 60:68] = neg_e[j2, 46:54]

    return hits, track_real

def generate_hit_matrices(n_events, tvt):
    #Create the realistic background for events
    hits, _ = build_background(n_events)
    #Place the full tracks that are reconstructable
    if(tvt=="Train"):
        hits,track=track_injection(hits,pos_events,neg_events)
    if(tvt=="Val"):
        hits,track=track_injection(hits,pos_events_val,neg_events_val)
    return hits.astype(bool), track.astype(int)

learning_rate_finder=1e-5
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
    restore_best_weights=True)
n_train=0

# Detect the number of GPUs available
gpus = tf.config.experimental.list_physical_devices('GPU')
num_gpus = len(gpus)
print(f"Number of GPUs available: {num_gpus}")

# Set up strategy for distributed training
if num_gpus > 1:
    strategy = tf.distribute.MirroredStrategy()
else:
    strategy = tf.distribute.get_strategy()

# Adjust batch size for the number of GPUs
batch_size_ef = 256 * num_gpus
batch_size_tf = 64 * num_gpus

print("Before while loop:", n_train)
while(n_train<5e6):
```

```
trainin, traintrack = generate_hit_matrices(750000, "Train")
print("Generated Training Data")
traintrack = traintrack/max_ele

valin, valtrack = generate_hit_matrices(75000, "Val")
print("Generated Validation Data")
valtrack = valtrack/max_ele

if(opt=='Pos'):
    traintrack = traintrack[:,:34]
    valtrack = valtrack[:,:34]
if(opt=='Neg'):
    traintrack = traintrack[:,34:]
    valtrack = valtrack[:,34:]

# Clear previous session
tf.keras.backend.clear_session()

with strategy.scope():
    probability_model =
tf.keras.Sequential([tf.keras.models.load_model('Networks/event_filter'),
tf.keras.layers.Softmax()])
    train_predictions = probability_model.predict(trainin,
batch_size=batch_size_ef, verbose=0)
    val_predictions = probability_model.predict(valin,
batch_size=batch_size_ef, verbose=0)
train_mask = train_predictions[:, 1] > 0.75
val_mask = val_predictions[:, 1] > 0.75

if single_muon==False: #If a dimuon finder, run generated events through
single-muon finders first.
    tf.keras.backend.clear_session()

    with strategy.scope():
        track_finder_pos =
tf.keras.models.load_model('Networks/Track_Finder_Pos')
        pos_predictions_val = (np.round(track_finder_pos.predict(valin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)
        pos_predictions_train = (np.round(track_finder_pos.predict(trainin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)

    tf.keras.backend.clear_session()

    with strategy.scope():
        track_finder_neg =
tf.keras.models.load_model('Networks/Track_Finder_Neg')
        neg_predictions_val = (np.round(track_finder_neg.predict(valin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)
        neg_predictions_train = (np.round(track_finder_neg.predict(trainin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)

    track_val = evaluate_finder(valin, valdrift,
np.column_stack((pos_predictions_val, neg_predictions_val)))
    results_val = calc_mismatches(track_val)
```

```
        val_mask &= ((results_val[0::4] < 2) & (results_val[1::4] < 2) &
    (results_val[2::4] < 3) & (results_val[3::4] < 3)).all(axis=0)

        track_train = evaluate_finder(trainin, traindrift,
    np.column_stack((pos_predictions_train, neg_predictions_train)))
        results_train = calc_mismatches(track_train)
        train_mask &= ((results_train[0::4] < 2) & (results_train[1::4] < 2) &
    (results_train[2::4] < 3) & (results_train[3::4] < 3)).all(axis=0)

    # Apply masks
    trainin = trainin[train_mask]
    traintrack = traintrack[train_mask]
    valin = valin[val_mask]
    valtrack = valtrack[val_mask]

    trainin = trainin[train_mask]
    traintrack = traintrack[train_mask]
    valin = valin[val_mask]
    valtrack = valtrack[val_mask]

    n_train += len(trainin)

    # Model Training
    tf.keras.backend.clear_session()
    with strategy.scope():
        model = tf.keras.models.load_model(model_name)
        optimizer = tf.keras.optimizers.Adam(learning_rate_finder)
        model.compile(optimizer=optimizer, loss='mse',
    metrics=['RootMeanSquaredError'])
        val_loss_before = model.evaluate(valin, valtrack,
    batch_size=batch_size_tf, verbose=2)[0]
        print(val_loss_before)
        history = model.fit(trainin, traintrack, epochs=1000,
    batch_size=batch_size_tf,
            verbose=2, validation_data=(valin, valtrack), callbacks=[callback])
        if min(history.history['val_loss']) < val_loss_before:
            model.save(model_name)  # Save only if improved
            learning_rate_finder *= 2
        learning_rate_finder /= 2

    del model  # Delete the model to free up memory
    gc.collect()  # Force garbage collection to release GPU memory
    print(n_train)
```

Listing A.4: Track Finder Training Code

A.2.4 Reconstruction Training Data Generation

```
import numpy as np
import uproot
from numba import njit, prange
import random
```

```
import tensorflow as tf
import gc
import sys
from Common_Functions import *

if len(sys.argv) != 2:
        print("Usage: python script.py <Charge or Vertex>")
        print("Options are Muon, All, Z, Target, or Dump")
        exit(1)

opt = sys.argv[1]

if(opt == 'Muon'):
    single_muon=True
else:single_muon=False

root_file_train = f"Root_Files/{opt}_Train_QA_v2.root"
root_file_val = f"Root_Files/{opt}_Val_QA_v2.root"
if single_muon:
    root_file_train = f"Root_Files/Z_Train_QA_v2.root"
    root_file_val = f"Root_Files/Z_Val_QA_v2.root"

model_name = f"Networks/Track_Finder_{opt}"

@njit(parallel=True)
def track_injection(hits, drift, pos_e, neg_e, pos_d, neg_d, pos_k, neg_k):
    kin = np.zeros((len(hits), 12))
    for z in prange(len(hits)):
        j = random.randrange(len(pos_e))
        if single_muon:j2 = random.randrange(len(neg_e))
        else:j2=j
        kin[z, :6] = pos_k[j]
        kin[z, 6:] = neg_k[j2]
        for k in range(54):
            if pos_e[j][k] > 0:
                if (random.random() < 0.94) and (k < 30):
                    hits[z][k][int(pos_e[j][k] - 1)] = 1
                    drift[z][k][int(pos_e[j][k] - 1)] = pos_d[j][k]
                if k > 29:
                    hits[z][k][int(pos_e[j][k] - 1)] = 1
            if neg_e[j2][k] > 0:
                if (random.random() < 0.94) and (k < 30):
                    hits[z][k][int(neg_e[j2][k] - 1)] = 1
                    drift[z][k][int(neg_e[j2][k] - 1)] = neg_d[j][k]
                if k > 29:
                    hits[z][k][int(neg_e[j2][k] - 1)] = 1
    return hits, drift, kin

def generate_hit_matrices(n_events, tvt):
    hits, drift = build_background(n_events)
    if tvt == "Train":
        hits, drift, kinematics = track_injection(hits, drift, pos_events,
    neg_events, pos_drift, neg_drift, pos_kinematics, neg_kinematics)
    if tvt == "Val":
```

```
        hits, drift, kinematics = track_injection(hits, drift, pos_events_val,
    neg_events_val, pos_drift_val, neg_drift_val, pos_kinematics_val,
    neg_kinematics_val)
    return hits.astype(bool), drift, kinematics

pos_events, pos_drift, pos_kinematics, neg_events, neg_drift, neg_kinematics =
    read_root_file(root_file_train)
pos_events_val, pos_drift_val, pos_kinematics_val, neg_events_val,
    neg_drift_val, neg_kinematics_val = read_root_file(root_file_val)

pos_events = clean(pos_events).astype(int)
neg_events = clean(neg_events).astype(int)
pos_events_val = clean(pos_events_val).astype(int)
neg_events_val = clean(neg_events_val).astype(int)

n_train = 0
train_input = []
val_input = []
train_kinematics = []
val_kinematics = []

# Detect the number of GPUs available
gpus = tf.config.experimental.list_physical_devices('GPU')
num_gpus = len(gpus)
print(f"Number of GPUs available: {num_gpus}")

# Set up strategy for distributed training
if num_gpus > 1:
    strategy = tf.distribute.MirroredStrategy()
else:
    strategy = tf.distribute.get_strategy()

# Adjust batch size for the number of GPUs
batch_size_ef = 256 * num_gpus
batch_size_tf = 64 * num_gpus

print(f"Number of devices: {strategy.num_replicas_in_sync}")

while n_train < 1e7:
    valin, valdrift, valkinematics = generate_hit_matrices(50000, "Val")
    trainin, traindrift, trainkinematics = generate_hit_matrices(500000, "Train")

    # Clear session and load the probability model for event filtering
    tf.keras.backend.clear_session()
    with strategy.scope():
        probability_model =
    tf.keras.Sequential([tf.keras.models.load_model('Networks/event_filter'),
    tf.keras.layers.Softmax()])
        train_predictions = probability_model.predict(trainin,
    batch_size=batch_size_ef, verbose=0)
        val_predictions = probability_model.predict(valin,
    batch_size=batch_size_ef, verbose=0)
        train_mask = train_predictions[:, 1] > 0.75
        val_mask = val_predictions[:, 1] > 0.75
```

```
# Clear session and load track finder models
tf.keras.backend.clear_session()
with strategy.scope():
    track_finder_pos =
tf.keras.models.load_model('Networks/Track_Finder_Pos')
    pos_predictions_val = (np.round(track_finder_pos.predict(valin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)
    pos_predictions_train = (np.round(track_finder_pos.predict(trainin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)

tf.keras.backend.clear_session()
with strategy.scope():
    track_finder_neg =
tf.keras.models.load_model('Networks/Track_Finder_Neg')
    neg_predictions_val = (np.round(track_finder_neg.predict(valin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)
    neg_predictions_train = (np.round(track_finder_neg.predict(trainin,
verbose=0, batch_size = batch_size_tf) * max_ele[:34])).astype(int)

# Update mask for validation data
track_val = evaluate_finder(valin, valdrift,
np.column_stack((pos_predictions_val, neg_predictions_val)))
results_val = calc_mismatches(track_val)
val_mask &= ((results_val[0::4] < 2) & (results_val[1::4] < 2) &
(results_val[2::4] < 3) & (results_val[3::4] < 3)).all(axis=0)

# Update mask for training data
track_train = evaluate_finder(trainin, traindrift,
np.column_stack((pos_predictions_train, neg_predictions_train)))
results_train = calc_mismatches(track_train)
train_mask &= ((results_train[0::4] < 2) & (results_train[1::4] < 2) &
(results_train[2::4] < 3) & (results_train[3::4] < 3)).all(axis=0)


if single_muon==False:
    # Apply masks
    trainin = trainin[train_mask]
    traindrift = traindrift[train_mask]
    trainkinematics = trainkinematics[train_mask]
    valin = valin[val_mask]
    valdrift = valdrift[val_mask]
    valkinematics = valkinematics[val_mask]

    # Clear session and load the track finder model
    tf.keras.backend.clear_session()
    with strategy.scope():
        track_finder_model = tf.keras.models.load_model(model_name)
        val_predictions = (np.round(track_finder_model.predict(valin,
verbose=0, batch_size = batch_size_tf) * max_ele)).astype(int)
        track_val = evaluate_finder(valin, valdrift, val_predictions)
        results_val = calc_mismatches(track_val)
        val_mask = ((results_val[0::4] < 2) & (results_val[1::4] < 2) &
(results_val[2::4] < 3) & (results_val[3::4] < 3)).all(axis=0)
```

```
            train_predictions = (np.round(track_finder_model.predict(trainin,
    verbose=0, batch_size = batch_size_tf) * max_ele)).astype(int)
            track_train = evaluate_finder(trainin, traindrift, train_predictions)
            results_train = calc_mismatches(track_train)
            train_mask = ((results_train[0::4] < 2) & (results_train[1::4] < 2)
    & (results_train[2::4] < 3) & (results_train[3::4] < 3)).all(axis=0)

    val_kinematics.append(valkinematics[val_mask])
    val_input.append(track_val[val_mask][:, :2])
    train_kinematics.append(trainkinematics[train_mask])
    train_input.append(track_train[train_mask][:, :2])

    n_train = len(np.concatenate(train_kinematics))

    # Save (use a consistent saving strategy to avoid repeated concatenation)
    np.save(f'Training_Data/{opt}_Val_In.npy', np.concatenate(val_input))
    np.save(f'Training_Data/{opt}_Val_Out.npy', np.concatenate(val_kinematics))
    np.save(f'Training_Data/{opt}_Train_In.npy', np.concatenate(train_input))
    np.save(f'Training_Data/{opt}_Train_Out.npy',
    np.concatenate(train_kinematics))
    gc.collect()
    print(n_train)
```

LISTING A.5: Training data generation code for muon and dimuon reconstruction and vertex finding.

A.2.5 Reconstruction

```
import numpy as np
import tensorflow as tf
import gc
import sys

if len(sys.argv) != 3:
    print("Usage: python script.py <Option> <Version>")
    print("Options are Vertex or Momentum")
    print("Version are Pos, Neg, All, Z, Target, or Dump")
    exit(1)

opt = sys.argv[1]
vers = sys.argv[2]
version = sys.argv[2]

if(vers == 'Pos') or (vers == 'Neg'):
    single_muon=True
else:single_muon=False

if opt == 'Vertex':
    model_name = f'Networks/Vertexing_{version}'
    mom_model_name = f'Networks/Reconstruction_{version}'

if opt == 'Momentum':
```

```
    model_name = f'Networks/Reconstruction_{version}'
    if single_muon:
        print('Momentum reconstruction not implemented for single-muons.')

if single_muon: version='Muon'

# Detect the number of GPUs available
gpus = tf.config.experimental.list_physical_devices('GPU')
num_gpus = len(gpus)
print(f"Number of GPUs available: {num_gpus}")

# Set up strategy for distributed training
if num_gpus > 1:
    strategy = tf.distribute.MirroredStrategy()
else:
    strategy = tf.distribute.get_strategy()

# Adjust batch size for the number of GPUs
batch_size_training = 1024 * num_gpus

#Define the means and standard deviations for output normalization
kin_means = np.array([2,0,35,-2,0,35])
kin_stds = np.array([0.6,1.2,10,0.6,1.2,10])
vertex_means=np.array([0,0,-300])
vertex_stds=np.array([10,10,300])

#Define the learning rate and callback
learning_rate=1e-6
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                            patience=100,
    restore_best_weights=True)

#Load the pre-generated training data
valin_reco = np.load(f"Training_Data/{version}_Val_In.npy")
valkinematics = np.load(f"Training_Data/{version}_Val_Out.npy")
filt = np.max(abs(valin_reco.reshape(len(valin_reco),(136))),axis=1)<1000
valin_reco = valin_reco[filt]
valkinematics = valkinematics[filt]

trainin_reco = np.load(f"Training_Data/{version}_Train_In.npy")
trainkinematics = np.load(f"Training_Data/{version}_Train_Out.npy")
filt = np.max(abs(trainin_reco.reshape(len(trainin_reco),(136))),axis=1)<1000
trainin_reco = trainin_reco[filt]
trainkinematics = trainkinematics[filt]

if opt == 'Vertex':
    if(vers=="Pos"):
        trainout = trainkinematics[:,0]
        valout = valkinematics[:,0]
    if(vers=="Neg"):
        trainout = trainkinematics[:,1]
        valout = valkinematics[:,1]
    if ~single_muon:
        trainout = trainkinematics[:,-3:]
        valout = valkinematics[:,-3:]
```

```
        trainout = (trainout-vertex_means)/vertex_stds
        valout = (valout-vertex_means)/vertex_stds
        with strategy.scope():
            model=tf.keras.models.load_model(mom_model_name)

            train_reco = model.predict(trainin_reco, verbose=0, batch_size =
    8192*num_gpus)
            val_reco = model.predict(valin_reco, verbose=0, batch_size =
    8192*num_gpus)

        trainin_reco=np.concatenate((train_reco.reshape((len(train_reco),3,2)),
                                     trainin_reco),axis=1)
        valin_reco=np.concatenate((val_reco.reshape((len(val_reco),3,2)),
                                   valin_reco),axis=1)


if opt == 'Momentum':
    trainout = np.column_stack((trainkinematics[:,:3],trainkinematics[:,-6:-3]))
    valout = np.column_stack((valkinematics[:,:3],valkinematics[:,-6:-3]))
    trainout = (trainout-kin_means)/kin_stds
    valout = (valout-kin_means)/kin_stds

tf.keras.backend.clear_session()
with strategy.scope():
    model=tf.keras.models.load_model(model_name)
    optimizer = tf.keras.optimizers.Adam(learning_rate)
    model.compile(optimizer=optimizer,
          loss=tf.keras.losses.mse,
          metrics=tf.keras.metrics.RootMeanSquaredError())
    history = model.fit(trainin_reco, trainout,
                epochs=10000, batch_size=batch_size_training, verbose=2,
                validation_data=(valin_reco,valout),callbacks=[callback])
    model.save(model_name)
```

Listing A.6: Code to load generated data from prior section and train either the momentum or vertex reconstruction networks.

A.2.6 Reconstructed Event Generation

```
    import os
import sys
import numpy as np
import uproot
from numba import njit, prange
import random
import tensorflow as tf
import gc
from Common_Functions import *

if len(sys.argv) != 2:
        print("Usage: python script.py <Vertex Distribution>")
        print("Currently supports All, Z, Target, and Dump")
```

```
        exit(1)

vertex = sys.argv[1]
root_file_train = f"Root_Files/{vertex}_Train_QA_v2.root"
root_file_val = f"Root_Files/{vertex}_Val_QA_v2.root"

pos_events, pos_drift, pos_kinematics, neg_events, neg_drift, neg_kinematics =
    read_root_file(root_file_train)
pos_events_val, pos_drift_val, pos_kinematics_val, neg_events_val,
    neg_drift_val, neg_kinematics_val = read_root_file(root_file_val)

pos_events=clean(pos_events).astype(int)
neg_events=clean(neg_events).astype(int)
pos_events_val=clean(pos_events_val).astype(int)
neg_events_val=clean(neg_events_val).astype(int)

@njit(parallel=True)
def track_injection(hits,drift,pos_e,neg_e,pos_d,neg_d,pos_k,neg_k):
    #Start generating the events
    kin=np.zeros((len(hits),9))
    for z in prange(len(hits)):
        j=random.randrange(len(pos_e))
        kin[z, :3] = pos_k[j, :3]
        kin[z, 3:9] = neg_k[j]
        for k in range(54):
            if(pos_e[j][k]>0):
                if(random.random()<0.94) and (k<30):
                    hits[z][k][int(pos_e[j][k]-1)]=1
                    drift[z][k][int(pos_e[j][k]-1)]=pos_d[j][k]
                if(k>29):
                    hits[z][k][int(pos_e[j][k]-1)]=1
            if(neg_e[j][k]>0):
                if(random.random()<0.94) and (k<30):
                    hits[z][k][int(neg_e[j][k]-1)]=1
                    drift[z][k][int(neg_e[j][k]-1)]=neg_d[j][k]
                if(k>29):
                    hits[z][k][int(neg_e[j][k]-1)]=1

    return hits,drift,kin

def generate_hit_matrices(n_events, tvt):
    #Create the realistic background for events
    hits, drift = build_background(n_events)
    #Place the full tracks that are reconstructable
    if(tvt=="Train"):
        hits,drift,kinematics=
    track_injection(hits,drift,pos_events,neg_events,pos_drift,neg_drift,
    pos_kinematics,neg_kinematics)
    if(tvt=="Val"):
        hits,drift,kinematics=
    track_injection(hits,drift,pos_events_val,neg_events_val,pos_drift_val,
    neg_drift_val,pos_kinematics_val,neg_kinematics_val)
    return hits.astype(bool), drift, kinematics

# Detect the number of GPUs available
```

```
gpus = tf.config.experimental.list_physical_devices('GPU')
num_gpus = len(gpus)
print(f"Number of GPUs available: {num_gpus}")

# Set up strategy for distributed training
if num_gpus > 1:
    strategy = tf.distribute.MirroredStrategy()
else:
    strategy = tf.distribute.get_strategy()

EF_batch = 256 * num_gpus
TF_batch = 64 * num_gpus
DN_batch = 8192 * num_gpus

def run_qtracker(hits, drift, kinematics):
    tf.keras.backend.clear_session()
    tf.compat.v1.reset_default_graph()
    with strategy.scope():
        model = tf.keras.models.load_model('Networks/event_filter')
        probability_model = tf.keras.Sequential([model,
    tf.keras.layers.Softmax()])
        event_classification_probabilies =
    probability_model.predict(hits,batch_size=EF_batch, verbose=0)

    mask = event_classification_probabilies[:,1]>=0.75
    hits = hits[mask]
    drift = drift[mask]
    kinematics=kinematics[mask]
    event_classification_probabilies = event_classification_probabilies[mask]

    tf.keras.backend.clear_session()
    with strategy.scope():
        model = tf.keras.models.load_model('Networks/Track_Finder_Pos')
        pos_predictions = model.predict(hits, verbose=0, batch_size = TF_batch)
    tf.keras.backend.clear_session()
    with strategy.scope():
        model = tf.keras.models.load_model('Networks/Track_Finder_Neg')
        neg_predictions =model.predict(hits, verbose=0, batch_size = TF_batch)
    predictions = (np.round(np.column_stack((pos_predictions,
    neg_predictions))*max_ele)).astype(int)

    muon_tracks=evaluate_finder(hits,drift,predictions)

    tf.keras.backend.clear_session()
    with strategy.scope():
        model = tf.keras.models.load_model('Networks/Vertexing_Pos')
        pos_pred = model.predict(muon_tracks[:,:34,:2], verbose=0, batch_size =
    DN_batch)
    tf.keras.backend.clear_session()
    with strategy.scope():
        model = tf.keras.models.load_model('Networks/Vertexing_Neg')
        neg_pred = model.predict(muon_tracks[:,34:,:2], verbose=0, batch_size =
    DN_batch)

    muon_track_quality = calc_mismatches(muon_tracks)
```

```
mask = ((muon_track_quality[0::4] < 2) & (muon_track_quality[1::4] < 2) &
(muon_track_quality[2::4] < 3) & (muon_track_quality[3::4] < 3)).all(axis=0)

# Apply the final filter to event_classification_probabilities
hits = hits[mask]
drift = drift[mask]
muon_tracks = muon_tracks[mask]
pos_pred = pos_pred[mask]
neg_pred = neg_pred[mask]
kinematics = kinematics[mask]
muon_track_quality = muon_track_quality.T[mask]
event_classification_probabilies = event_classification_probabilies[mask]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_All')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
all_vtx_track = evaluate_finder(hits,drift,predictions)[:,:,:2]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_All')
    reco_kinematics =
model.predict(all_vtx_track,batch_size=DN_batch,verbose=0)


vertex_input=np.concatenate((reco_kinematics.reshape((len(reco_kinematics),3,2)),
all_vtx_track),axis=1)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Vertexing_All')
    reco_vertex = model.predict(vertex_input,batch_size=DN_batch,verbose=0)

all_vtx_reco=np.concatenate((reco_kinematics,reco_vertex),axis=1)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Z')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
z_vtx_track = evaluate_finder(hits,drift,predictions)[:,:,:2]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_Z')
    reco_kinematics =
model.predict(z_vtx_track,batch_size=DN_batch,verbose=0)
```

```
vertex_input=np.concatenate((reco_kinematics.reshape((len(reco_kinematics),3,2))
,z_vtx_track),axis=1)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Vertexing_Z')
    reco_vertex = model.predict(vertex_input,batch_size=DN_batch,verbose=0)

z_vtx_reco=np.concatenate((reco_kinematics,reco_vertex),axis=1)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Target')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
target_track = evaluate_finder(hits,drift,predictions)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_Target')
    target_vtx_reco =
model.predict(target_track[:,:,:2],batch_size=DN_batch,verbose=0)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Dump')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
dump_track = evaluate_finder(hits,drift,predictions)[:,:,:2]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_Dump')
    dump_vtx_reco = model.predict(dump_track,batch_size=DN_batch,verbose=0)

dimuon_track_quality = calc_mismatches(target_track)

mask = ((dimuon_track_quality[0::4] < 2) & (dimuon_track_quality[1::4] < 2)
& (dimuon_track_quality[2::4] < 3) & (dimuon_track_quality[3::4] <
3)).all(axis=0)

predictions = np.column_stack((event_classification_probabilies[:,1],
pos_pred, neg_pred, all_vtx_reco, z_vtx_reco, target_vtx_reco,
dump_vtx_reco, muon_track_quality, dimuon_track_quality.T))
tracks = np.column_stack((muon_tracks[:,:,:2], all_vtx_track[:,:,:2],
z_vtx_track[:,:,:2], target_track[:,:,:2], dump_track[:,:,:2]))

predictions = predictions[mask]
tracks = tracks[mask]
```

```
    target_dump_input =
    np.column_stack((predictions,tracks.reshape((len(tracks),(68*2*5)))))

    return predictions, tracks

# Initialize lists to store the data
dimuon_probability=[]
all_predictions = []
tracks = []
truth = []
total_entries = 0
#Generate training data
while(total_entries<10000000):
    try:
        hits, drift, kinematics = generate_hit_matrices(500000,"Train")

        all_predictions, tracks = run_qtracker(hits, drift, kinematics)


    np.save(f'Training_Data/{vertex}_Tracks_Train.npy',np.concatenate(tracks,
    axis=0))

    np.save(f'Training_Data/{vertex}_Reco_Train.npy',np.concatenate(all_predictions,
    axis=0))

        total_entries += len(hits)
        print(total_entries)
        del hits, drift, all_predictions, tracks
    except Exception as e:
        pass

# Initialize lists to store the data
dimuon_probability=[]
all_predictions = []
tracks = []
truth = []
total_entries = 0


#Generate validation data
while(total_entries<1000000):
    try:
        hits, drift, kinematics = generate_hit_matrices(500000,"Val")

        all_predictions, tracks = run_qtracker(hits, drift, kinematics)

        np.save(f'Training_Data/{vertex}_Tracks_Val.npy',np.concatenate(tracks,
    axis=0))

    np.save(f'Training_Data/{vertex}_Reco_Val.npy',np.concatenate(all_predictions,
    axis=0))

        total_entries += len(hits)
        print(total_entries)
```

```
        del hits, drift, all_predictions, tracks
    except Exception as e:
        pass
```

LISTING A.7: Code to generate reconstructed target and dump events for use in the Target-Dump filter training.

A.2.7 Target-Dump Filter

```
import os
import sys
import numpy as np
import uproot
from numba import njit, prange
import random
import tensorflow as tf
import gc
from Common_Functions import *

if len(sys.argv) != 2:
        print("Usage: python script.py <Vertex Distribution>")
        print("Currently supports All, Z, Target, and Dump")
        exit(1)

vertex = sys.argv[1]
root_file_train = f"Root_Files/{vertex}_Train_QA_v2.root"
root_file_val = f"Root_Files/{vertex}_Val_QA_v2.root"

pos_events, pos_drift, pos_kinematics, neg_events, neg_drift, neg_kinematics =
    read_root_file(root_file_train)
pos_events_val, pos_drift_val, pos_kinematics_val, neg_events_val,
    neg_drift_val, neg_kinematics_val = read_root_file(root_file_val)

pos_events=clean(pos_events).astype(int)
neg_events=clean(neg_events).astype(int)
pos_events_val=clean(pos_events_val).astype(int)
neg_events_val=clean(neg_events_val).astype(int)

@njit(parallel=True)
def track_injection(hits,drift,pos_e,neg_e,pos_d,neg_d,pos_k,neg_k):
    #Start generating the events
    kin=np.zeros((len(hits),9))
    for z in prange(len(hits)):
        j=random.randrange(len(pos_e))
        kin[z, :3] = pos_k[j, :3]
        kin[z, 3:9] = neg_k[j]
        for k in range(54):
            if(pos_e[j][k]>0):
                if(random.random()<0.94) and (k<30):
                    hits[z][k][int(pos_e[j][k]-1)]=1
                    drift[z][k][int(pos_e[j][k]-1)]=pos_d[j][k]
                if(k>29):
```

```
                    hits[z][k][int(pos_e[j][k]-1)]=1
            if(neg_e[j][k]>0):
                if(random.random()<0.94) and (k<30):
                    hits[z][k][int(neg_e[j][k]-1)]=1
                    drift[z][k][int(neg_e[j][k]-1)]=neg_d[j][k]
                if(k>29):
                    hits[z][k][int(neg_e[j][k]-1)]=1

    return hits,drift,kin

def generate_hit_matrices(n_events, tvt):
    #Create the realistic background for events
    hits, drift = build_background(n_events)
    #Place the full tracks that are reconstructable
    if(tvt=="Train"):
        hits,drift,kinematics=
    track_injection(hits,drift,pos_events,neg_events,pos_drift,neg_drift,
    pos_kinematics,neg_kinematics)
    if(tvt=="Val"):
        hits,drift,kinematics=
    track_injection(hits,drift,pos_events_val,neg_events_val,pos_drift_val,
    neg_drift_val,pos_kinematics_val,neg_kinematics_val)
    return hits.astype(bool), drift, kinematics

# Detect the number of GPUs available
gpus = tf.config.experimental.list_physical_devices('GPU')
num_gpus = len(gpus)
print(f"Number of GPUs available: {num_gpus}")

# Set up strategy for distributed training
if num_gpus > 1:
    strategy = tf.distribute.MirroredStrategy()
else:
    strategy = tf.distribute.get_strategy()

EF_batch = 256 * num_gpus
TF_batch = 64 * num_gpus
DN_batch = 8192 * num_gpus

def run_qtracker(hits, drift, kinematics):
    tf.keras.backend.clear_session()
    tf.compat.v1.reset_default_graph()
    with strategy.scope():
        model = tf.keras.models.load_model('Networks/event_filter')
        probability_model = tf.keras.Sequential([model,
    tf.keras.layers.Softmax()])
        event_classification_probabilies =
    probability_model.predict(hits,batch_size=EF_batch, verbose=0)

    mask = event_classification_probabilies[:,1]>=0.75
    hits = hits[mask]
    drift = drift[mask]
    kinematics=kinematics[mask]
    event_classification_probabilies = event_classification_probabilies[mask]
```

```
tf.keras.backend.clear_session()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Pos')
    pos_predictions = model.predict(hits, verbose=0, batch_size = TF_batch)
tf.keras.backend.clear_session()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Neg')
    neg_predictions =model.predict(hits, verbose=0, batch_size = TF_batch)
predictions =
(np.round(np.column_stack((pos_predictions,neg_predictions))*max_ele)).astype(int)

muon_tracks=evaluate_finder(hits,drift,predictions)

tf.keras.backend.clear_session()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Vertexing_Pos')
    pos_pred = model.predict(muon_tracks[:,:34,:2], verbose=0, batch_size =
DN_batch)
tf.keras.backend.clear_session()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Vertexing_Neg')
    neg_pred = model.predict(muon_tracks[:,34:,:2], verbose=0, batch_size =
DN_batch)

muon_track_quality = calc_mismatches(muon_tracks)
mask = ((muon_track_quality[0::4] < 2) & (muon_track_quality[1::4] < 2) &
(muon_track_quality[2::4] < 3) & (muon_track_quality[3::4] < 3)).all(axis=0)

# Apply the final filter to event_classification_probabilities
hits = hits[mask]
drift = drift[mask]
muon_tracks = muon_tracks[mask]
pos_pred = pos_pred[mask]
neg_pred = neg_pred[mask]
kinematics = kinematics[mask]
muon_track_quality = muon_track_quality.T[mask]
event_classification_probabilies = event_classification_probabilies[mask]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_All')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
all_vtx_track = evaluate_finder(hits,drift,predictions)[:,:,:2]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_All')
    reco_kinematics =
model.predict(all_vtx_track,batch_size=DN_batch,verbose=0)


vertex_input=np.concatenate((reco_kinematics.reshape((len(reco_kinematics),3,2)),all_vtx_tra
```

```
tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Vertexing_All')
    reco_vertex = model.predict(vertex_input,batch_size=DN_batch,verbose=0)

all_vtx_reco=np.concatenate((reco_kinematics,reco_vertex),axis=1)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Z')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
z_vtx_track = evaluate_finder(hits,drift,predictions)[:,:,:2]

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_Z')
    reco_kinematics =
model.predict(z_vtx_track,batch_size=DN_batch,verbose=0)


vertex_input=np.concatenate((reco_kinematics.reshape((len(reco_kinematics),3,2)),z_vtx_track

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Vertexing_Z')
    reco_vertex = model.predict(vertex_input,batch_size=DN_batch,verbose=0)

z_vtx_reco=np.concatenate((reco_kinematics,reco_vertex),axis=1)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Target')
    predictions = (np.round(model.predict(hits,verbose=0, batch_size =
TF_batch)*max_ele)).astype(int)
target_track = evaluate_finder(hits,drift,predictions)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model=tf.keras.models.load_model('Networks/Reconstruction_Target')
    target_vtx_reco =
model.predict(target_track[:,:,:2],batch_size=DN_batch,verbose=0)

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
with strategy.scope():
    model = tf.keras.models.load_model('Networks/Track_Finder_Dump')
```

```
        predictions = (np.round(model.predict(hits,verbose=0, batch_size =
    TF_batch)*max_ele)).astype(int)
    dump_track = evaluate_finder(hits,drift,predictions)[:,:,:2]

    tf.keras.backend.clear_session()
    tf.compat.v1.reset_default_graph()
    with strategy.scope():
        model=tf.keras.models.load_model('Networks/Reconstruction_Dump')
        dump_vtx_reco = model.predict(dump_track,batch_size=DN_batch,verbose=0)

    dimuon_track_quality = calc_mismatches(target_track)

    mask = ((dimuon_track_quality[0::4] < 2) & (dimuon_track_quality[1::4] < 2)
    & (dimuon_track_quality[2::4] < 3) & (dimuon_track_quality[3::4] <
    3)).all(axis=0)

    predictions = np.column_stack((event_classification_probabilies[:,1],
    pos_pred, neg_pred, all_vtx_reco, z_vtx_reco, target_vtx_reco,
    dump_vtx_reco, muon_track_quality, dimuon_track_quality.T))
    tracks = np.column_stack((muon_tracks[:,:,:2], all_vtx_track[:,:,:2],
    z_vtx_track[:,:,:2], target_track[:,:,:2], dump_track[:,:,:2]))

    predictions = predictions[mask]
    tracks = tracks[mask]

    target_dump_input =
    np.column_stack((predictions,tracks.reshape((len(tracks),(68*2*5)))))

    return predictions, tracks

# Initialize lists to store the data
dimuon_probability=[]
all_predictions = []
tracks = []
truth = []
total_entries = 0
#Generate training data
while(total_entries<10000000):
    try:
        hits, drift, kinematics = generate_hit_matrices(500000,"Train")

        all_predictions, tracks = run_qtracker(hits, drift, kinematics)


    np.save(f'Training_Data/{vertex}_Tracks_Train.npy',np.concatenate(tracks,
    axis=0))

    np.save(f'Training_Data/{vertex}_Reco_Train.npy',np.concatenate(all_predictions,
    axis=0))

        total_entries += len(hits)
        print(total_entries)
        del hits, drift, all_predictions, tracks
    except Exception as e:
        pass
```

```
# Initialize lists to store the data
dimuon_probability=[]
all_predictions = []
tracks = []
truth = []
total_entries = 0


#Generate validation data
while(total_entries<1000000):
    try:
        hits, drift, kinematics = generate_hit_matrices(500000,"Val")

        all_predictions, tracks = run_qtracker(hits, drift, kinematics)

        np.save(f'Training_Data/{vertex}_Tracks_Val.npy',np.concatenate(tracks,
    axis=0))

    np.save(f'Training_Data/{vertex}_Reco_Val.npy',np.concatenate(all_predictions,
    axis=0))

        total_entries += len(hits)
        print(total_entries)
        del hits, drift, all_predictions, tracks
    except Exception as e:
        pass
```

Listing A.8: Code to train the Target-Dump Filter.

A.3 Execution

A.3.1 Functions Library

These are the functions used to execute QTracker.

```
import os
import numpy as np
import uproot
import numba
from numba import njit, prange
import tensorflow as tf

network_path = '/scratch/acc5dn/QTracker_Run_Refactor/Networks/'

def save_explanation():
    explanation = []
    n_columns = 0
    if event_prob_output:
```

```
    explanation.append(f"Event Filter Probabilites: Columns
{n_columns}:{n_columns+6}")
    n_columns+=2
explanation.append(f"Muon Z-Vertex: Columns {n_columns}:{n_columns+1}")
n_columns+=2
explanation.append(f"All Vertex Kinematic Predictions: Columns
{n_columns}:{n_columns+6}")
n_columns+=6
explanation.append(f"All Vertex Vertex Predictions: Columns
{n_columns}:{n_columns+3}")
n_columns+=3
explanation.append(f"Z Vertex Kinematic Predictions: Columns
{n_columns}:{n_columns+6}")
n_columns+=6
explanation.append(f"Z Vertex Vertex Predictions: Columns
{n_columns}:{n_columns+3}")
n_columns+=3
explanation.append(f"Target Vertex Kinematic Predictions: Columns
{n_columns}:{n_columns+6}")
n_columns+=6
explanation.append(f"Dump Vertex Kinematic Predictions: Columns
{n_columns}:{n_columns+6}")
n_columns+=6
if target_prob_output:
    explanation.append(f"Target Probability: Column {n_columns}")
    n_columns+=1
if track_quality_output:
    explanation.append(f"Muon Track Quality: Colums
{n_columns}:{n_columns+12}")
    n_columns+=12
    explanation.append(f"Dimuon Track Quality: Colums
{n_columns}:{n_columns+12}")
    n_columns+=12
if tracks_output:
    explanation.append(f"Pos Muon Track: {n_columns}:{n_columns+34}")
    n_columns+=34
    explanation.append(f"Neg Muon Track: {n_columns}:{n_columns+34}")
    n_columns+=34
    explanation.append(f"All Vertex Track: {n_columns}:{n_columns+68}")
    n_columns+=68
    explanation.append(f"Z Vertex Track: {n_columns}:{n_columns+68}")
    n_columns+=68
    explanation.append(f"Target Vertex Track: {n_columns}:{n_columns+68}")
    n_columns+=68
    explanation.append(f"Dump Vertex Track: {n_columns}:{n_columns+68}")
    n_columns+=68
if metadata_output:
    if runid_output:
        explanation.append(f"Run ID: Column {n_columns}")
        n_columns+=1
    if eventid_output:
        explanation.append(f"Event ID: Column {n_columns}")
        n_columns+=1
    if spillid_output:
        explanation.append(f"Spill ID: Column {n_columns}")
```

```
            n_columns+=1
        if triggerbit_output:
            explanation.append(f"Trigger Bits: Column {n_columns}")
            n_columns+=1
        if target_pos_output:
            explanation.append(f"Target Positions: Column {n_columns}")
            n_columns+=1
        if turnid_output:
            explanation.append(f"Turn ID: Column {n_columns}")
            n_columns+=1
        if rfid_output:
            explanation.append(f"RFID: Column {n_columns}")
            n_columns+=1
        if intensity_output:
            explanation.append(f"Cherenkov Information: Columns
    {n_columns}:{n_columns+32}")
            n_columns+=32
        if trigg_rds_output:
            explanation.append(f"Number of Trigger Roads: Column {n_columns}")
            n_columns+=1
        if occ_output:
            if occ_before_cuts:explanation.append(f"Detector Occupancies before
    cuts: Columns {n_columns}:{n_columns+54}")
            else:explanation.append(f"Detector Occupancies after cuts: Columns
    {n_columns}:{n_columns+54}")
            n_columns+=54

    filename= f'reconstructed_columns.txt'
    with open(filename,'w') as file:
        file.write('Explanation of Columns:\n\n')
        for info in explanation:
            file.write(f"{info}\n")

save_explanation()

def save_output():
    # After processing through all models, the results are aggregated based on
    options at top,
    # and the final dataset is prepared.

    # The reconstructed kinematics and vertex information are normalized
    # using predefined means and standard deviations before saving.
    row_count = 0
    definitions_string = []
    if file_extension == '.root':
        metadata = []
        metadata_row_count = 0
        metadata_string = []
        if runid_output:metadata.append(runid)
        if eventid_output:metadata.append(eventid)
        if spillid_output:metadata.append(spill_id)
        if triggerbit_output:metadata.append(trigger_bit)
        if target_pos_output:metadata.append(target_position)
        if turnid_output:metadata.append(turnid)
        if rfid_output:metadata.append(rfid)
```

```
        if intensity_output:metadata.append(intensity)
        if trigg_rds_output:metadata.append(n_roads)
        if occ_output:
            if occ_before_cuts:metadata.append(n_hits)
            else:metadata.append(np.sum(hits,axis=2))#Calculates the occupanceis
    from the Hit Matrix
        metadata = np.column_stack(metadata)
    if file_extension == '.npz':
        metadata = truth
    output = []
    if event_prob_output: output.append(event_classification_probabilies)
        output.append(all_predictions)
    if target_prob_output:
        output.append(target_dump_prob[:,1])
    if track_quality_output:
        output.append(muon_track_quality)
        output.append(dimuon_track_quality)
    if tracks_output: output.append(tracks)
    if metadata_output: output.append(metadata)
    output_data = np.column_stack(output)

    base_filename = 'Reconstructed/' + os.path.basename(root_file).split('.')[0]
    os.makedirs("Reconstructed", exist_ok=True)  # Ensure the output directory
    exists.
    np.save(base_filename + '_reconstructed.npy', output_data)  # Save the final
    dataset.
    print(f"File {base_filename}_reconstructed.npy has been saved successfully.")


kin_means = np.array([ 2.00, 0.00, 35.0, -2.00, -0.00, 35.0 ])
kin_stds = np.array([ 0.6, 1.2, 10.00, 0.60, 1.20, 10.00 ])

vertex_means=np.array([0,0,-300])
vertex_stds=np.array([10,10,300])

means = np.concatenate((kin_means,vertex_means))
stds = np.concatenate((kin_stds,vertex_stds))

# Function to convert raw detector data into a structured hit matrix.
@njit()
def hit_matrix(detectorid,elementid,drifttime,tdctime,intime,hits,drift,tdc):
    #Convert into hit matrices
    if(timing_cuts==False):intime[:]=2
    for j in prange (len(detectorid)):
        if ((detectorid[j]<7) or (detectorid[j]>12)) and (intime[j]>0):
            if (tdc[int(detectorid[j])-1][int(elementid[j]-1)]==0) or
    (tdctime[j]<tdc[int(detectorid[j])-1][int(elementid[j]-1)]):
                hits[int(detectorid[j])-1][int(elementid[j]-1)]=1
                drift[int(detectorid[j])-1][int(elementid[j]-1)]=drifttime[j]
                tdc[int(detectorid[j])-1][int(elementid[j]-1)]=tdctime[j]
    return hits,drift,tdc

max_ele = [200, 200, 168, 168, 200, 200, 128, 128,  112,  112, 128, 128, 134,
    134,
           112, 112, 134, 134,  20,  20,  16,  16,  16,  16,  16,  16,
```

```
        72,  72,  72,  72,  72,  72,  72,  72, 200, 200, 168, 168, 200, 200,
        128, 128,  112,  112, 128, 128, 134, 134, 112, 112, 134, 134,
        20,  20,  16,  16,  16,  16,  16,  16,  72,  72,  72,  72,  72,
        72,  72,  72]

# Function to evaluate the Track Finder neural network.
@njit(parallel=True)
def evaluate_finder(testin, testdrift, predictions):
    # The function constructs inputs for the neural network model based on test
    data
    # and predictions, processing each event in parallel for efficiency.
    reco_in = np.zeros((len(testin), 68, 3))

    def process_entry(i, dummy, j_offset):
        j = dummy if dummy <= 5 else dummy + 6
        if dummy > 11:
            if predictions[i][12+j_offset] > 0:
                j = dummy + 6
            elif predictions[i][12+j_offset] < 0:
                j = dummy + 12

        if dummy > 17:
            j = 2 * (dummy - 18) + 30 if predictions[i][2 * (dummy - 18) + 30 +
    j_offset] > 0 else 2 * (dummy - 18) + 31

        if dummy > 25:
            j = dummy + 20

        k = abs(predictions[i][dummy + j_offset])
        sign = k / predictions[i][dummy + j_offset] if k > 0 else 1
        if(dummy<6):window=15
        elif(dummy<12):window=5
        elif(dummy<18):window=5
        elif(dummy<26):window=1
        else:window=3
        k_sum = np.sum(testin[i][j][k - window:k + window-1])
        if k_sum > 0 and ((dummy < 18) or (dummy > 25)):
            k_temp = k
            n = 1
            while testin[i][j][k - 1] == 0:
                k_temp += n
                n = -n * (abs(n) + 1) / abs(n)
                if 0 <= k_temp < 201:
                    k = int(k_temp)

        reco_in[i][dummy + j_offset][0] = sign * k
        reco_in[i][dummy + j_offset][1] = testdrift[i][j][k - 1]
        if(testin[i][j][k - 1]==1):
            reco_in[i][dummy + j_offset][2]=1

    for i in prange(predictions.shape[0]):
        for dummy in prange(34):
            process_entry(i, dummy, 0)

        for dummy in prange(34):
```

```
            process_entry(i, dummy, 34)

    return reco_in

# Function to remove closely spaced hits that are likely not real particle
    interactions (cluster hits).
@njit(parallel=True, forceobj=True)
def declusterize(hits, drift, tdc):
    # This function iterates over hits and removes clusters of hits that are too
    close
    # together, likely caused by noise or multiple hits from a single particle
    passing
    # through the detector. It's an important step in cleaning the data for
    analysis.
    for k in prange(len(hits)):
        for i in range(54):
            if(i<30 or i>45):
                for j in range(100):#Work from both sides
                    if(hits[k][i][j]==1 and hits[k][i][j+1]==1):
                        if(hits[k][i][j+2]==0):#Two hits
                            if(drift[k][i][j]>0.4 and
    drift[k][i][j+1]>0.9):#Edge hit check
                                hits[k][i][j+1]=0
                                drift[k][i][j+1]=0
                                tdc[k][i][j+1]=0
                            elif(drift[k][i][j+1]>0.4 and
    drift[k][i][j]>0.9):#Edge hit check
                                hits[k][i][j]=0
                                drift[k][i][j]=0
                                tdc[k][i][j]=0
                            if(abs(tdc[k][i][j]-tdc[k][i][j+1])<8):#Electronic
    Noise Check
                                hits[k][i][j+1]=0
                                drift[k][i][j+1]=0
                                tdc[k][i][j+1]=0
                                hits[k][i][j]=0
                                drift[k][i][j]=0
                                tdc[k][i][j]=0

                        else:#Check larger clusters for Electronic Noise
                            n=2
                            while(hits[k][i][j+n]==1):n=n+1
                            dt_mean = 0
                            for m in range(n-1):
                                dt_mean += (tdc[k][i][j+m]-tdc[k][i][j+m+1])
                            dt_mean = dt_mean/(n-1)
                            if(dt_mean<10):
                                for m in range(n):
                                    hits[k][i][j+m]=0
                                    drift[k][i][j+m]=0
                                    tdc[k][i][j+m]=0
                    if(hits[k][i][200-j]==1 and hits[k][i][199-j]):
                        if(hits[k][i][198-j]==0):
                            if(drift[k][i][200-j]>0.4 and
    drift[k][i][199-j]>0.9):  # Edge hit check
```

```
                                hits[k][i][199-j]=0
                                drift[k][i][199-j]=0
                            elif(drift[k][i][199-j]>0.4 and
    drift[k][i][200-j]>0.9):  # Edge hit check
                                hits[k][i][200-j]=0
                                drift[k][i][200-j]=0
                            if(abs(tdc[k][i][200-j]-tdc[k][i][199-j])<8):  #
    Electronic Noise Check
                                hits[k][i][199-j]=0
                                drift[k][i][199-j]=0
                                tdc[k][i][199-j]=0
                                hits[k][i][200-j]=0
                                drift[k][i][200-j]=0
                                tdc[k][i][200-j]=0
                        else:  # Check larger clusters for Electronic Noise
                            n=2
                            while(hits[k][i][200-j-n]==1): n=n+1
                            dt_mean = 0
                            for m in range(n-1):
                                dt_mean +=
    abs(tdc[k][i][200-j-m]-tdc[k][i][200-j-m-1])
                            dt_mean = dt_mean/(n-1)
                            if(dt_mean<10):
                                for m in range(n):
                                    hits[k][i][200-j-m]=0
                                    drift[k][i][200-j-m]=0
                                    tdc[k][i][200-j-m]=0


# Specify the directory containing the root files
i = 0

# Drift chamber mismatch calculation
def calc_mismatches(track):
    results = []
    for pos_slice, neg_slice in [(slice(0, 6), slice(34, 40)), (slice(6, 12),
    slice(40, 46)), (slice(12, 18), slice(46, 52))]:
        # Compare even indices with odd indices for the 0th component of the
    final dimension
        even_pos_indices = track[:, pos_slice, 0].reshape(track.shape[0], -1,
    2)[:, :, 0]
        odd_pos_indices = track[:, pos_slice, 0].reshape(track.shape[0], -1,
    2)[:, :, 1]
        even_neg_indices = track[:, neg_slice, 0].reshape(track.shape[0], -1,
    2)[:, :, 0]
        odd_neg_indices = track[:, neg_slice, 0].reshape(track.shape[0], -1,
    2)[:, :, 1]

        results.extend([
            np.sum(abs(even_pos_indices - odd_pos_indices) > 1, axis=1),
            np.sum(abs(even_neg_indices - odd_neg_indices) > 1, axis=1),
            np.sum(track[:, pos_slice, 2] == 0, axis=1),
            np.sum(track[:, neg_slice, 2] == 0, axis=1)
        ])
```

```
    return np.array(results)

def load_model(network):
    tf.keras.backend.clear_session()
    tf.compat.v1.reset_default_graph()
    return tf.keras.models.load_model(network_path+network)

def process_file(file_path, file_extension, max_ele, dimuon_prob_threshold,
    means, stds, kin_means, kin_stds):
    try:
        if file_extension == '.root':
            # Read in data from the ROOT file.
            targettree = uproot.open(file_path + ":save")
            detectorid =
    targettree["fAllHits.detectorID"].arrays(library="np")["fAllHits.detectorID"]
            elementid =
    targettree["fAllHits.elementID"].arrays(library="np")["fAllHits.elementID"]
            driftdistance =
    targettree["fAllHits.driftDistance"].arrays(library="np")["fAllHits.driftDistance"]
            tdctime =
    targettree["fAllHits.tdcTime"].arrays(library="np")["fAllHits.tdcTime"]
            intime =
    targettree["fAllHits.flag"].arrays(library="np")["fAllHits.flag"]

            hits = np.zeros((len(detectorid), 54, 201), dtype=bool)
            drift = np.zeros((len(detectorid), 54, 201))
            tdc = np.zeros((len(detectorid), 54, 201), dtype=int)

            for n in range(len(detectorid)):
                hits[n], drift[n], tdc[n] = hit_matrix(detectorid[n],
    elementid[n], driftdistance[n], tdctime[n], intime[n], hits[n], drift[n],
    tdc[n])

            declusterize(hits, drift, tdc)

        elif file_extension == '.npz':
            generated = np.load(file_path)
            hits = generated["hits"]
            drift = generated["drift"]
            truth = generated["truth"]

        print("Loaded events")

        model = load_model('Networks/event_filter')
        probability_model = tf.keras.Sequential([model,
    tf.keras.layers.Softmax()])
        event_classification_probabilities = probability_model.predict(hits,
    batch_size=256, verbose=0)

        filt = event_classification_probabilities[:, 1] > dimuon_prob_threshold
        hits, drift = hits[filt], drift[filt]

        pos_model = load_model('Track_Finder_Pos')
        pos_predictions = pos_model.predict(hits, verbose=0)
        neg_model = load_model('Track_Finder_Neg')
```

```
    neg_predictions = neg_model.predict(hits, verbose=0)
    predictions = (np.round(np.column_stack((pos_predictions,
neg_predictions)) * max_ele)).astype(int)

    muon_track = evaluate_finder(hits, drift, predictions)

    pos_recon_model = load_model('Reconstruction_Pos')
    pos_pred = pos_recon_model.predict(muon_track[:, :34, :2], verbose=0)
    neg_recon_model = load_model('Reconstruction_Neg')
    neg_pred = neg_recon_model.predict(muon_track[:, 34:, :2], verbose=0)

    muon_track_quality = calc_mismatches(muon_track).T
    filt1 = ((muon_track_quality[0::4] < 2) & (muon_track_quality[1::4] < 2)
& (muon_track_quality[2::4] < 3) & (muon_track_quality[3::4] <
3)).all(axis=0)

    hits, drift = hits[filt1], drift[filt1]

    if file_extension == '.root':
        runid =
targettree["fRunID"].arrays(library="np")["fRunID"][filt][filt1]
        eventid =
targettree["fEventID"].arrays(library="np")["fEventID"][filt][filt1]
        spill_id =
targettree["fSpillID"].arrays(library="np")["fSpillID"][filt][filt1]
        trigger_bit =
targettree["fTriggerBits"].arrays(library="np")["fTriggerBits"][filt][filt1]
        target_position =
targettree["fTargetPos"].arrays(library="np")["fTargetPos"][filt][filt1]
        turnid =
targettree["fTurnID"].arrays(library="np")["fTurnID"][filt][filt1]
        rfid = targettree["fRFID"].arrays(library="np")["fRFID"][filt][filt1]
        intensity =
targettree["fIntensity[33]"].arrays(library="np")["fIntensity[33]"][filt][filt1]
        n_roads =
targettree["fNRoads[4]"].arrays(library="np")["fNRoads[4]"][filt][filt1]
        n_hits =
targettree["fNHits[55]"].arrays(library="np")["fNHits[55]"][filt][filt1]

    elif file_extension == '.npz':
        truth = truth[filt][filt1]

    print("Filtered Events")
    if len(hits) > 0:
        track_finder_all_model = load_model('Track_Finder_All')
        predictions = (np.round(track_finder_all_model.predict(hits,
verbose=0) * max_ele)).astype(int)
        all_vtx_track = evaluate_finder(hits, drift, predictions)[:, :, :2]

        reco_all_model = load_model('Reconstruction_All')
        reco_kinematics = reco_all_model.predict(all_vtx_track,
batch_size=8192, verbose=0)
```

```
        vertex_input =
np.concatenate((reco_kinematics.reshape((len(reco_kinematics), 3, 2)),
all_vtx_track), axis=1)

        vertexing_all_model = load_model('Vertexing_All')
        reco_vertex = vertexing_all_model.predict(vertex_input,
batch_size=8192, verbose=0)

        all_vtx_reco = np.concatenate((reco_kinematics, reco_vertex), axis=1)

        track_finder_z_model = load_model('Track_Finder_Z')
        predictions = (np.round(track_finder_z_model.predict(hits,
verbose=0) * max_ele)).astype(int)
        z_vtx_track = evaluate_finder(hits, drift, predictions)[:, :, :2]

        reco_z_model = load_model('Reconstruction_Z')
        reco_kinematics = reco_z_model.predict(z_vtx_track, batch_size=8192,
verbose=0)

        vertex_input =
np.concatenate((reco_kinematics.reshape((len(reco_kinematics), 3, 2)),
z_vtx_track), axis=1)

        vertexing_z_model = load_model('Vertexing_Z')
        reco_vertex = vertexing_z_model.predict(vertex_input,
batch_size=8192, verbose=0)

        z_vtx_reco = np.concatenate((reco_kinematics, reco_vertex), axis=1)

        track_finder_target_model = load_model('Track_Finder_Target')
        predictions = (np.round(track_finder_target_model.predict(hits,
verbose=0) * max_ele)).astype(int)
        target_track = evaluate_finder(hits, drift, predictions)

        reco_target_model = load_model('Reconstruction_Target')
        target_vtx_reco = reco_target_model.predict(target_track[:, :, :2],
batch_size=8192, verbose=0)

        track_finder_dump_model = load_model('Track_Finder_Dump')
        predictions = (np.round(track_finder_dump_model.predict(hits,
verbose=0) * max_ele)).astype(int)
        dump_track = evaluate_finder(hits, drift, predictions)[:, :, :2]

        reco_dump_model = load_model('Reconstruction_Dump')
        dump_vtx_reco = reco_dump_model.predict(dump_track, batch_size=8192,
verbose=0)

        dimuon_track_quality = calc_mismatches(target_track).T

        reco_kinematics =
np.concatenate((event_classification_probabilities[:, 1], pos_pred,
neg_pred, all_vtx_reco, z_vtx_reco, target_vtx_reco, dump_vtx_reco,
muon_track_quality, dimuon_track_quality), axis=1)
```

```
            tracks = np.column_stack((muon_track[:, :, :2], all_vtx_track,
    z_vtx_track, target_track[:, :, :2], dump_track))

            target_dump_input = np.column_stack((reco_kinematics,
    tracks.reshape((len(tracks), (68 * 2 * 5)))))

            target_dump_filter_model = load_model('target_dump_filter')
            target_dump_pred =
    target_dump_filter_model.predict(target_dump_input, batch_size=512,
    verbose=0)
            target_dump_prob = np.exp(target_dump_pred) /
    np.sum(np.exp(target_dump_pred), axis=1, keepdims=True)

            all_predictions = np.column_stack((pos_pred, neg_pred, all_vtx_reco
    * stds + means, z_vtx_reco * stds + means, target_vtx_reco * kin_stds +
    kin_means, dump_vtx_reco * kin_stds + kin_means))

            print("Found", len(all_predictions), "Dimuons in file.")

            save_output()
        else:
            print("No events meeting dimuon criteria.")
    except Exception as e:
        print(f"Error processing file {file_path}: {e}")
```

LISTING A.9: These functions are used in evaluating files using QTracker.

A.3.2 Looping QTracker

```
###QTracker Rivanna###
# This script is used to reconstruct large amount of data on Rivanna via Slurm
    job submission.

#####Parent path Directory#####
root_directory = '/project/ptgroup/seaquest/data/digit/02/'

#####Import Functions to Run QTracker#####
from QTracker_Run_Library import *

#####Reconstruction Options#####
dimuon_prob_threshold = 0.75 #Minimum dimuon probability to reconstruct.
timing_cuts = True #Use SRawEvent intime flag for hit filtering

#####Output Options#####
event_prob_output = True #Output the event filter probabilites for reconstructed
    events
track_quality_output = True #Output the number of drift chamber mismatches for
    each chamber
target_prob_output = True #Output the probability that the dimuon pair is from
    the target.
tracks_output = False #Output the element IDs for the identified tracks for all
    three track finders
```

```
metadata_output = True #Output metadata

####Metadata Options#####
#Select which values from the SRawEvent file should be saved to the
    reconstructed .npy file
runid_output = True #Output the run id
eventid_output = True #Output the event id
spillid_output = True #Output the spill id
triggerbit_output = True #Output the trigger bit for the event
target_pos_output = True #Output the target type (hydrogen, deuterium, etc.)
turnid_output = True #Output the turn id
rfid_output = True #Output the RF ID
intensity_output = True #Output Cherenkov information
trigg_rds_output = True #Output the number of trigger roads activated
occ_output = True #Output the occupancy information
occ_before_cuts = False #If set to true, counts number of hits before timing
    cuts, if false, outputs occupancies after hit reduction.

import os
import numpy as np
import uproot
import numba
from numba import njit, prange
import tensorflow as tf

root_files = [file for file in os.listdir(root_directory) if
    file.endswith('.root')]

for i, root_file in enumerate(root_files):
    process_file(root_file, root_directory, i, max_ele, dimuon_prob_threshold,
    means, stds, kin_means, kin_stds)
```

LISTING A.10: This code is used to evaluate all of the files in a specified directory.

A.3.3 Single File QTracker

```
###QTracker Execution###
#This script is used for reconstructing a single root or numpy file.
#Usage:
"""""""""""""""""
python QTracker_Run.py /path/to/file.root|.npz
"""""""""""""""""

#####Import Functions to Run QTracker#####
from QTracker_Run_Library import *

#####Reconstruction Options#####
dimuon_prob_threshold = 0.75 #Minimum dimuon probability to reconstruct.
timing_cuts = True #Use SRawEvent intime flag for hit filtering

#####Output Options#####
```

```
event_prob_output = True #Output the event filter probabilites for reconstructed
    events
track_quality_output = True #Output the number of drift chamber mismatches for
    each chamber
target_prob_output = True #Output the probability that the dimuon pair is from
    the target.
tracks_output = False #Output the element IDs for the identified tracks for all
    three track finders
metadata_output = True #Output metadata

####Metadata Options#####
#Select which values from the SRawEvent file should be saved to the
    reconstructed .npy file
#Only affects output if using .root file.
runid_output = True #Output the run id
eventid_output = True #Output the event id
spillid_output = True #Output the spill id
triggerbit_output = True #Output the trigger bit for the event
target_pos_output = True #Output the target type (hydrogen, deuterium, etc.)
turnid_output = True #Output the turn id
rfid_output = True #Output the RF ID
intensity_output = True #Output Cherenkov information
trigg_rds_output = True #Output the number of trigger roads activated
occ_output = True #Output the occupancy information
occ_before_cuts = False #If set to true, counts number of hits before timing
    cuts, if false, outputs occupancies after hit reduction.

import os
import numpy as np
import uproot  # For reading ROOT files, a common data format in particle
    physics.
import numba  # Just-In-Time (JIT) compiler for speeding up Python code.
from numba import njit, prange  # njit for compiling functions, prange for
    parallel loops.
import tensorflow as tf  # For using machine learning models.
import sys

# Check if the script is run without a ROOT file or with the script name as
    input.
if len(sys.argv) != 2:
    print("Usage: python script_name.py <input_file.root|.npz>")
    quit()

root_file = sys.argv[1]  # Takes the first command-line argument as the input
    file path.

# Check if the input file has a valid extension
valid_extensions = ('.root', '.npz')
file_extension = os.path.splitext(root_file)[1]
if file_extension not in valid_extensions:
    print("Invalid input file format. Supported formats: .root, .npy")
    quit()

process_file(root_file, '', i, max_ele, dimuon_prob_threshold, means, stds,
    kin_means, kin_stds)
```

LISTING A.11: This code is used to evaluate a single specified file.

Bibliography

[1] Cush. Standard model of elementary particles, 2024. URL https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg. Public domain, via Wikimedia Commons.

[2] Cedric Lorce, Barbara Pasquini, and Marc Vanderhaeghen. Unified framework for generalized and transverse-momentum dependent parton distributions within a 3Q light-cone picture of the nucleon. *JHEP*, 05:041, 2011. doi: 10.1007/JHEP05(2011)041.

[3] A. Accardi, J. L. Albacete, M. Anselmino, N. Armesto, E. C. Aschenauer, A. Bacchetta, D. Boer, W. Brooks, et al. Electron ion collider: The next qcd frontier - understanding the glue that binds us all? 2012.

[4] Daniël Boer. Boer-mulders function. 2022. URL http://www.physics4all.nl/BoerMulders.html.

[5] L. A. Harland-Lang, A. D. Martin, P. Motylinski, and R. S. Thorne. Parton distributions in the lhc era: Mmht 2014 pdfs. *The European Physical Journal C*, 75(5), May 2015. ISSN 1434-6052. doi: 10.1140/epjc/s10052-015-3397-6. URL http://dx.doi.org/10.1140/epjc/s10052-015-3397-6.

[6] Christian Wiese, Constantia Alexandrou, Martha Constantinou, Kyriakos Hadjiyiannakou, Christos Kallidonis, Giannis Koutsou, Karl Jansen, Haralambos Panagopoulos, Fernanda Steffens, and Alejandro Vaquero. Recent results for the proton spin decomposition from lattice qcd. *Proceedings of XXIV International Workshop on Deep-Inelastic Scattering and Related Subjects — PoS(DIS2016)*, Nov 2016. doi: 10.22323/1.265.0240. URL http://dx.doi.org/10.22323/1.265.0240.

[7] Mark Thomson. *Modern particle physics*. Cambridge University Press, New York, 2013. ISBN 978-1-107-03426-6. doi: 10.1017/CBO9781139525367.

[8] No machine-readable author provided. E2m assumed (based on copyright claims). Drell-yan, 2017. URL https://commons.wikimedia.org/wiki/File:Drell-Yan.svg. Public domain, via Wikimedia Commons.

[9] Daniël Boer. Investigating the origins of transverse spin asymmetries at bnl rhic. *Phys. Rev. D*, 60:014012, Jun 1999. doi: 10.1103/PhysRevD.60.014012. URL https://link.aps.org/doi/10.1103/PhysRevD.60.014012.

[10] L. Y. Zhu, J. C. Peng, P. E. Reimer, T. C. Awes, M. L. Brooks, C. N. Brown, J. D. Bush, T. A. Carey, T. H. Chang, W. E. Cooper, C. A. Gagliardi, G. T. Garvey, D. F. Geesaman, E. A. Hawker, X. C. He, L. D. Isenhower, D. M. Kaplan, S. B. Kaufman, S. A. Klinksiek, D. D. Koetke, D. M. Lee, W. M. Lee, M. J. Leitch, N. Makins, P. L. McGaughey, J. M. Moss, B. A. Mueller, P. M. Nord, V. Papavassiliou, B. K. Park, G. Petitt, M. E. Sadler, W. E. Sondheim, P. W. Stankus, T. N. Thompson, R. S. Towell, R. E. Tribble, M. A. Vasiliev, J. C. Webb, J. L. Willis, D. K. Wise, and G. R. Young. Measurement of angular distributions of drell-yan dimuons in $p+d$ interactions at 800 GeV/c. *Phys. Rev. Lett.*, 99:082301, Aug 2007. doi: 10.1103/PhysRevLett.99.082301. URL https://link.aps.org/doi/10.1103/PhysRevLett.99.082301.

[11] Paul E. Reimer. Measurements of Drell-Yan Angular Distributions and the Transverse Boer-Mulders Structure Function. In *17th International Workshop on Deep-Inelastic Scattering and Related Subjects*, page 212, Berlin, Germany, 2009. Science Wise Publ.

[12] J. J. Aubert, U. Becker, P. J. Biggs, J. Burger, M. Chen, G. Everhart, P. Goldhagen, J. Leong, T. McCorriston, T. G. Rhoades, M. Rohde, Samuel C. C. Ting, Sau Lan Wu, and Y. Y. Lee. Experimental observation of a heavy particle j. *Phys. Rev. Lett.*, 33:1404–1406, Dec 1974. doi: 10.1103/PhysRevLett.33.1404. URL https://link.aps.org/doi/10.1103/PhysRevLett.33.1404.

[13] SeaQuest Collaboration, C. A. Aidala, J. R. Arrington, C. Ayuso, B. M. Bowen, M. L. Bowen, K. L. Bowling, A. W. Brown, C. N. Brown, R. Byrd, R. E. Carlisle, T. Chang, W. C. Chang, A. Chen, J. Y. Chen, D. C. Christian, X. Chu, B. P. Dannowitz, M. Daugherity, M. Diefenthaler, J. Dove, C. Durandet, L. El Fassi, E. Erdos, D. M. Fox, D. F. Geesaman, R. Gilman, Y. Goto, L. Guo, R. Guo, T. Hague, C. R. Hicks, R. J. Holt, D. Isenhower, X. Jiang, J. M. Katich, B. M. Kerns, E. R. Kinney, N. D. Kitts, A. Klein, D. Kleinjan, J. Kras, Y. Kudo, P. J. Lin, D. Liu, K. Liu, M. X. Liu, W. Lorenzon,

N. C. R. Makins, J. D. Martinez, R. E. McClellan, B. McDonald, P. L. McGaughey, S. E. McNease, M. M. Medeiros, B. Miller, A. J. Miller, S. Miyasaka, Y. Miyachi, I. A. Mooney, D. H. Morton, B. Nadim, K. Nagai, K. Nakahara, K. Nakano, S. Nara, S. Obata, J. C. Peng, S. Prasad, A. J. R. Puckett, B. J. Ramson, R. S. Raymond, P. E. Reimer, J. G. Rubin, R. Salinas, F. Sanftl, S. Sawada, T. Sawada, M. B. C. Scott, L. E. Selensky, T. A. Shibata, S. Shiu, D. Su, A. S. Tadepalli, M. Teo, B. G. Tice, C. L. Towell, R. S. Towell, S. Uemura, S. G. Wang, S. Watson, N. White, A. B. Wickes, M. R. Wood, J. Wu, Z. Xi, Z. Ye, and Y. Yin. The seaquest spectrometer at fermilab, 2019.

[14] Kei Nagai. *Recent Measurement of Flavor Asymmetry of Antiquarks in the Proton by Drell–Yan Experiment SeaQuest at Fermilab*. PhD thesis, Tokyo Institute of Technology, Department of Physics, Tokyo, Japan, February 2017.

[15] Andrew Chen, J. C. Peng, H. Leung, M. Tian, N. Makins, M. Brooks, A. Klein, D. Kleinjan, K. Liu, M. McCumber, P. McGaughey, J. Miraal-Martinez, C. Da Silva, Sho Uemura, M. Jen, X. Li, J. Arrington, D. Geesaman, P. E. Reimer, C. Brown, R. J. Tesarek, S. Sawada, W. Lorenzon, R. Raymond, K. Slifer, D. Ruth, Y. Goto, K. Nakano, T. A. Shibata, D. Crabb, D. Day, D. Keller, O. Rondon, Z. Akbar, G. Dodge, S. Bueltmann, J. Dunne, D. Dutta, L. El Fassi, H. Jiang, E. Kinney, N. Doshita, T. Iwata, Y. Miyachi, M. Daugherity, D. Isenhower, R. Towell, S. Watson, N. Rowlands, Y. Ngenzi, S. Pate, V. Papavassiliou, H. Yu, and F. Hossain. Probing nucleon's spin structures with polarized drell-yan in the fermilab spinquest experiment, 2019. URL https://arxiv.org/abs/1901.09994.

[16] James Davis Maxwell. *Probing Proton Spin Structure: a Measurement of g2 at Four-momentum Transfer of 2 to 6 GeV^2*. Phd dissertation, University of Virginia, Poquoson, Virginia, December 2011. B.S. Physics, Mathematics, University of Virginia, 2004; M.A. Physics, University of Virginia, 2010.

[17] Paul Gavrikov. visualkeras. https://github.com/paulgavrikov/visualkeras, 2020.

[18] Walber. Precision recall, 2021. URL https://commons.wikimedia.org/wiki/File:Precisionrecall.svg. CC BY-SA 4.0, via Wikimedia Commons.

[19] Steven Weinberg. The Making of the standard model. *Eur. Phys. J. C*, 34: 5–13, 2004. doi: 10.1140/epjc/s2004-01761-1.

[20] Fred L. Wilson. Fermi's Theory of Beta Decay. *American Journal of Physics*, 36(12):1150–1160, 12 1968. ISSN 0002-9505. doi: 10.1119/1.1974382. URL https://doi.org/10.1119/1.1974382.

[21] Hideki Yukawa. On the Interaction of Elementary Particles I. *Proc. Phys. Math. Soc. Jap.*, 17:48–57, 1935. doi: 10.1143/PTPS.1.1.

[22] Chen-Ning Yang and Robert L. Mills. Conservation of Isotopic Spin and Isotopic Gauge Invariance. *Phys. Rev.*, 96:191–195, 1954. doi: 10.1103/PhysRev.96.191.

[23] C. S. Wu, E. Ambler, R. W. Hayward, D. D. Hoppes, and R. P. Hudson. Experimental test of parity conservation in beta decay. *Phys. Rev.*, 105:1413–1415, Feb 1957. doi: 10.1103/PhysRev.105.1413. URL https://link.aps.org/doi/10.1103/PhysRev.105.1413.

[24] Sheldon L. Glashow. Partial-symmetries of weak interactions. *Nuclear Physics*, 22(4):579–588, 1961. ISSN 0029-5582. doi: https://doi.org/10.1016/0029-5582(61)90469-2. URL https://www.sciencedirect.com/science/article/pii/0029558261904692.

[25] Steven Weinberg. A model of leptons. *Phys. Rev. Lett.*, 19:1264–1266, Nov 1967. doi: 10.1103/PhysRevLett.19.1264. URL https://link.aps.org/doi/10.1103/PhysRevLett.19.1264.

[26] Abdus Salam. Weak and Electromagnetic Interactions. *Conf. Proc. C*, 680519: 367–377, 1968. doi: 10.1142/9789812795915_0034.

[27] Murray Gell-Mann. A Schematic Model of Baryons and Mesons. *Phys. Lett.*, 8:214–215, 1964. doi: 10.1016/S0031-9163(64)92001-3.

[28] G. Zweig. *An SU(3) model for strong interaction symmetry and its breaking. Version 2*, pages 22–101. 2 1964.

[29] Richard P. Feynman. Very high-energy collisions of hadrons. *Phys. Rev. Lett.*, 23:1415–1417, 1969. doi: 10.1103/PhysRevLett.23.1415.

[30] David J. Gross and Frank Wilczek. Ultraviolet behavior of non-abelian gauge theories. *Phys. Rev. Lett.*, 30:1343–1346, Jun 1973. doi: 10.1103/PhysRevLett.30.1343. URL https://link.aps.org/doi/10.1103/PhysRevLett.30.1343.

[31] H. David Politzer. Reliable perturbative results for strong interactions? *Phys. Rev. Lett.*, 30:1346–1349, Jun 1973. doi: 10.1103/PhysRevLett.30.1346. URL https://link.aps.org/doi/10.1103/PhysRevLett.30.1346.

[32] Kenneth G. Wilson. Confinement of quarks. *Phys. Rev. D*, 10:2445–2459, Oct 1974. doi: 10.1103/PhysRevD.10.2445. URL https://link.aps.org/doi/10.1103/PhysRevD.10.2445.

[33] B. Povh; C. Scholz; K. Rith; F. Zetsche. *Particles and Nuclei*. Springer, 2008. ISBN 978-3-540-79367-0.

[34] Doreen Wackeroth and Leila Belkora. Cross section. 3 2009. URL https://ed.fnal.gov/painless/pdfs/cross.pdf.

[35] Mark Thomson. *Modern particle physics*. Cambridge University Press, New York, 2013. ISBN 978-1-107-03426-6. doi: 10.1017/CBO9781139525367.

[36] Mark Thomson. *Modern particle physics*. Cambridge University Press, New York, 2013. ISBN 978-1-107-03426-6. doi: 10.1017/CBO9781139525367.

[37] Xiangdong Ji. Viewing the proton through "color" filters. *Phys. Rev. Lett.*, 91:062001, Aug 2003. doi: 10.1103/PhysRevLett.91.062001. URL https://link.aps.org/doi/10.1103/PhysRevLett.91.062001.

[38] M. Brooks et al. Seaquest with a transversely polarized target (e1039). 2016. URL https://twist.phys.virginia.edu/work/E1039proposal_final.pdf.

[39] Dennis Sivers. Single-spin production asymmetries from the hard scattering of pointlike constituents. *Phys. Rev. D*, 41:83–90, Jan 1990. doi: 10.1103/PhysRevD.41.83. URL https://link.aps.org/doi/10.1103/PhysRevD.41.83.

[40] Daniël Boer and Piet J Mulders. Time-reversal odd distribution functions in leptoproduction. *Physical Review D*, 57(12):5780–5786, 1998.

[41] B. U. Musch, Ph. Hägler, M. Engelhardt, J. W. Negele, and A. Schäfer. Sivers and boer-mulders observables from lattice qcd. *Phys. Rev. D*, 85:094510, May 2012. doi: 10.1103/PhysRevD.85.094510. URL https://link.aps.org/doi/10.1103/PhysRevD.85.094510.

[42] Davison E. Soper. Parton distribution functions. *Nuclear Physics B - Proceedings Supplements*, 53(1-3):69–80, Feb 1997. ISSN 0920-5632. doi: 10.1016/s0920-5632(96)00600-7. URL http://dx.doi.org/10.1016/S0920-5632(96)00600-7.

[43] Arthur Conover. *Kinematic Dependence of the Dilution Factor In the Spin-Quest Experiment E1039 At Fermilab*. Ms (master of science), University of Virginia, Physics - Graduate School of Arts and Sciences, 2020.

[44] J; EMC Collaboration Ashman. A measurement of the spin asymmetry and determination of the structure function g1 in deep inelastic muon-proton scattering. *Physics Letters B*, 206(2):364, Dec 1987.

[45] E. D. Bloom, D. H. Coward, H. DeStaebler, J. Drees, G. Miller, L. W. Mo, R. E. Taylor, M. Breidenbach, J. I. Friedman, G. C. Hartmann, and H. W. Kendall. High-energy inelastic $e - p$ scattering at 6 and 10 degrees. *Phys. Rev. Lett.*, 23:930–934, Oct 1969. doi: 10.1103/PhysRevLett.23.930. URL https://link.aps.org/doi/10.1103/PhysRevLett.23.930.

[46] Sidney D. Drell and Tung-Mow Yan. Massive lepton-pair production in hadron-hadron collisions at high energies. *Phys. Rev. Lett.*, 25:316–320, Aug 1970. doi: 10.1103/PhysRevLett.25.316. URL https://link.aps.org/doi/10.1103/PhysRevLett.25.316.

[47] M. Anselmino, M. Boglione, U. D'Alesio, S. Melis, F. Murgia, and A. Prokudin. Sivers effect in drell-yan processes. *Phys. Rev. D*, 79:054010, Mar 2009. doi: 10.1103/PhysRevD.79.054010. URL https://link.aps.org/doi/10.1103/PhysRevD.79.054010.

[48] John C. Collins and Davison E. Soper. Angular distribution of dileptons in high-energy hadron collisions. *Phys. Rev. D*, 16:2219–2225, Oct 1977. doi: 10.1103/PhysRevD.16.2219. URL https://link.aps.org/doi/10.1103/PhysRevD.16.2219.

[49] C. S. Lam and Wu-Ki Tung. Systematic approach to inclusive lepton pair production in hadronic collisions. *Phys. Rev. D*, 18:2447–2461, Oct 1978. doi: 10.1103/PhysRevD.18.2447. URL https://link.aps.org/doi/10.1103/PhysRevD.18.2447.

[50] C. S. Lam and Wu-Ki Tung. Parton-model relation without quantum-chromodynamic modifications in lepton pair production. *Phys. Rev. D*,

21:2712–2715, May 1980. doi: 10.1103/PhysRevD.21.2712. URL https://link.aps.org/doi/10.1103/PhysRevD.21.2712.

[51] Leszek Motyka, Mariusz Sadzikowski, and Tomasz Stebel. Lam-tung relation breaking in Z^0 hadroproduction as a probe of parton transverse momentum. *Phys. Rev. D*, 95:114025, Jun 2017. doi: 10.1103/PhysRevD.95.114025. URL https://link.aps.org/doi/10.1103/PhysRevD.95.114025.

[52] J. E. Augustin, A. M. Boyarski, M. Breidenbach, F. Bulos, J. T. Dakin, G. J. Feldman, G. E. Fischer, D. Fryberger, G. Hanson, B. Jean-Marie, R. R. Larsen, V. Lüth, H. L. Lynch, D. Lyon, C. C. Morehouse, J. M. Paterson, M. L. Perl, B. Richter, P. Rapidis, R. F. Schwitters, W. M. Tanenbaum, F. Vannucci, G. S. Abrams, D. Briggs, W. Chinowsky, C. E. Friedberg, G. Goldhaber, R. J. Hollebeek, J. A. Kadyk, B. Lulu, F. Pierre, G. H. Trilling, J. S. Whitaker, J. Wiss, and J. E. Zipse. Discovery of a narrow resonance in e^+e^- annihilation. *Phys. Rev. Lett.*, 33:1406–1408, Dec 1974. doi: 10.1103/PhysRevLett.33.1406. URL https://link.aps.org/doi/10.1103/PhysRevLett.33.1406.

[53] V. M. Aulchenko et al. New precision measurement of the J / psi and psi-prime meson masses. *Phys. Lett. B*, 573:63–79, 2003. doi: 10.1016/j.physletb.2003.08.028.

[54] Partha Bhaduri, A. Chaudhuri, and Subhasis Chattopadhyay. J/psi production in proton induced collisions at fair. 10 2011.

[55] Fermilab. Fermilab: A national laboratory for particle physics and accelerator research, 2023. URL https://www.fnal.gov/.

[56] D. G. Crabb and W. Meyer. Solid polarized targets for nuclear and particle physics experiments. *Annual Review of Nuclear and Particle Science*, 47(1): 67–109, 1997. doi: 10.1146/annurev.nucl.47.1.67. URL https://doi.org/10.1146/annurev.nucl.47.1.67.

[57] W. Meyer, K.H. Althoff, W. Havenith, O. Kaul, H. Riechert, E. Schilling, G. Sternal, and W. Thiel. Dynamic deuteron polarization in irradiated d-ammonia (nd3 and its first use in a high energy photon beam. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 227(1):35–44, 1984. ISSN 0168-9002. doi: https://doi.org/10.1016/0168-9002(84)90098-6.

[58] Jonathan Mellor. Studies and measurements of irradiated solid polarized target materials. M.s. thesis, University of Virginia, 2006.

[59] F. Bateman, M. Desrosiers, L. Hudson, B.M. Coursey, P. Bergstrom, and Stephen Seltzer. Nist accelerator facilities and programs in support of industrial radiation research. *AIP Conference Proceedings*, 680:877–880, 08 2003. doi: 10.1063/1.1619849.

[60] Tapio O. Niinikoski. *The Physics of Polarized Targets*. Cambridge University Press, 2020. doi: 10.1017/9781108567435.

[61] S.T. Goertz. The dynamic nuclear polarization process. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 526(1-2):28–42, 2004. doi: 10.1016/j.nima.2004.03.147.

[62] Z. Akbar and D. Keller. Thermal analysis and simulation of the superconducting magnet in the spinquest experiment at fermilab. In *The 18th International Workshop on Polarized Sources, Targets and Polarimetry (PoS)*, 2019. doi: 10.22323/1.379.0057.

[63] Yan et. al Pei. An elementary introduction to kalman filtering. *arXiv preprint arXiv:1710.04055*, 2017.

[64] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[65] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

[66] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[68] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):

84–90, may 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://doi.org/10.1145/3065386.

[70] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[71] Zhaohuizi Ji. *Angular distribution analysis in the Drell-Yan process with 120 GeV proton beam and LH2 (LD2) target at SeaQuest experiment*. PhD thesis, Shandong University, Jinan, China, December 2023.

[72] Leszek Motyka, Mariusz Sadzikowski, and Tomasz Stebel. Lam-tung relation breaking in ¡mml:math xmlns:mml="http://www.w3.org/1998/math/mathml" display="inline"¿¡mml:msup¿¡mml:mi¿z¡/mml:mi¿¡mml:mn¿0¡/mml:mn¿¡/mml:msup¿¡/mml:m hadroproduction as a probe of parton transverse momentum. *Physical Review D*, 95(11), June 2017. ISSN 2470-0029. doi: 10.1103/physrevd.95.114025. URL http://dx.doi.org/10.1103/PhysRevD.95.114025.